Comparison of Fair and Optimum Ridesharing Plans

BY

LUCA FOTI
B.S., Politecnico di Milano, Milan, Italy, 2015

THESIS

Submitted as partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Chicago, 2017

Chicago, Illinois

To my Mom and Sister

ACKNOWLEDGMENTS

First of all, I want to thank Prof. Ouri Wolfson who encouraged me and stimulated my interest for this work we made, giving me a lot of his precious time. Also, I want to thank Prof. Paolo Cremonesi who accepted to be my advisor for this thesis.

I want to thank my mother for her precious support she gave me and always will throughout my life and, thus, this experience. Even if there was 7-hour difference, she succeeded in giving me the strength to overcome any sort of obstacle.

I want to thank my sister, Fede, because it has been almost 9 years we have lived in different cities, but even since last year that distance became enormous, she always had time to call and support me whenever I needed.

Also, thank you to the rest of my family back in Italy that kept showing me their affection.

A thank you to all the "Poli" guys I have met here and with whom I spent a great time in Chicago (and not only); with you, time has passed too quickly since last August.

A thank you to all the "Poli" guys I left in Milan and with whom I will resume the good old routine during classes and breaks, even if it is only for a little time more (hopefully).

Last but not least, I want to thank all my childhood friends from Reggio because even, if it has bee years living far way, our friendship is still strong and everytime I see a notification from our group chat, you always make me smile.

LF

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES (continued)

FIGURE **PAGE**

LIST OF ABBREVIATIONS

3D-SR	Stable Roommates Problem with Triple Rooms
ER	Entity-Relationship
GCCF	Graph-Constrained Coalition Formation
GiST	Generalized Search Tree
ILP	Integer Linear Programming
NYC	New York City
OSM	Open Street Map
RSG	Ride Sharing Graph
RSP	Ride Sharing Plan
RTV	Request-Trip-Vehicle
RV	Request-Vehicle
SQL	Structured Query Language
SRP	Stable Roommates Problem
TV	Trip-Vehicle

SUMMARY

Services like “Uber” and “Lyft” have become widespread in large cities for everyday mobility. This is due to the ease of requesting a ride (you can simply download the mobile app and request a driver on the fly from where you are currently located to your desired destination) and to the cheapness of these services.

Recently, “Uber” and “Lyft” have embraced ride sharing opportunities: a passenger who requests a ride may decide to save a certain amount of money at the price of sharing his/her ride with someone else (which would delay his/her arrival at destination).

For what concerns passengers matching, these services attempt to optimize savings at a global level. But a possible scenario is that a passenger A is matched to passenger B, while he/she would have preferred being matched to passenger C, who, in turn, would have preferred A as ride sharing partner as well.

This introduces the concept of “fairness” in ride sharing: if a particular passenger A has not been matched to his/her top preferred passenger B, this means that passenger B preferred to be matched to some other C, and so on recursively.

Fairness is addressed by issuing the optimum plan (i.e., without considering fair shared trips but only total savings at a global level) and, at the same time, applying a compensation scheme which derives from the fair plan: in this way, if a passenger has to pay a higher fare with the optimum plan with respect to the price which would be fairer, then he/she receives a discount according to his/her savings if the fair plan would have been issued instead.

In this thesis, we compare the optimum and fair ride sharing plans in different scenarios (new requests are matched to empty taxis or to taxis with passengers on board which will have to dynamically change its route) in order to understand the % redistribution of money that is necessary to apply such a compensation scheme, when fares are based on overall distance traveled.

CHAPTER 1

INTRODUCTION

Ridesharing services have become more and more important for urban mobility, especially in big cities, because they represent a valid solution to issues such as air pollution, fuel consumption, traffic congestion.

Moreover, due to the widespread of smart phones, nowadays it is simple for a passenger to request a ride, wherever he/she is and for whenever he/she desires.

If ridesharing were exploited, there would be a decrease in the impact of the cited issues. As a consequence, since traffic would be less significant, then passengers would benefit from delays reduction, which were caused by this problem.

In general, in the scientific literature there are two different directions by which ridesharing can be addressed: optimizing some criteria such as distance traveled and travel time; providing approaches to find, for a passenger, a good partner in ridesharing in the least time possible in order to deal with real time requests.

In this thesis, we will focus on the criterion of "total saving" for the passengers.

Typically, within the context of "Uber Pool" and "Lyft Line" ridesharing services, it is possible that a passenger A is matched to passenger B, even if he/she may prefer being matched to passenger C because this will provide a higher saving: this is due to the fact that "Uber" and "Lyft" may be optimizing the matching problem at a global level, instead of considering the single passenger.

This introduces the concept of “fairness”: if a particular passenger A has not been matched to his/her top preferred passenger B, this means that passenger B preferred to be matched to some other C, and so on recursively.

1.1 Motivations and contribution

As we can see in the examples presented in section 1.6, “fair” and “optimum” ridesharing plans can be different and, currently, little has been done to analyze the difference in savings between the two plans.

Hence, the contribution of this thesis is to evaluate the “fair” plan over the “optimum” plan and to quantify the % of money which is required to be redistributed, since the optimum plan will be eventually executed but a compensation scheme derived from the fair plan is applied.

We will consider fares based on distance traveled, thus maximizing (i.e., being “fair” to) the savings in terms of money is equivalent to maximizing (i.e., being “fair” to) the savings in terms of distance traveled (as we will show in section 4.1).

We will consider different scenarios in which ridesharing plans are applied: this requires a process of standardization of how shared trips are represented.

1.2 Requests pooling

Typically in the literature, requests for ridesharing are considered with queuing based formulations ([1]), thus vehicles are matched to requests as the latter arrive in real-time.

Another option, which is the one followed in this thesis, is to group together requests issued for a specific time interval into a “pool”.

Pool (definition): it is a set of requests characterized by a time interval which ranges from “start pool time” to “end pool time” (e.g., if pool size is 5 minutes, then a possible pool starts at “10:00:00” and ends at “10:04:59”): all the requests which have been issued for a pick up within this interval (e.g., pick up time is at “10:02:00”) are added to this set.

Moreover, if in the previous pool (e.g., which starts at “9:55:00” and ends at “9:59:59”) there are some requests which are not assigned to any vehicle, then they are retained for the next pool (even if they have been issued before); otherwise, all other requests which have been assigned to some vehicle are removed from the pool.
In a pool, if two or more requests are assigned to the same vehicle, they will form a “trip”.

1.3 Static and dynamic model

There are two possible approaches to deal with requests: static and dynamic model.

Within the static model, requests which specify a pick up time within the same range interval are pooled together and then assigned to empty vehicles. Hence, vehicles which are already processing some other requests are not taken into account for this assignment (i.e., their initial route cannot be modified).
A possible scenario for this model is the following. A person’s flight has landed at 10am, but he/she needs a ride to go from the airport (hub) to his/her destination: what the passenger can do is to register to a specific pool (e.g., “10:30-10:35” pool) so that there is time to pick up his/her checked baggage. Below is a figure which represents this scenario by combining two

requests (the extension to the combination of more than two requests is straightforward):

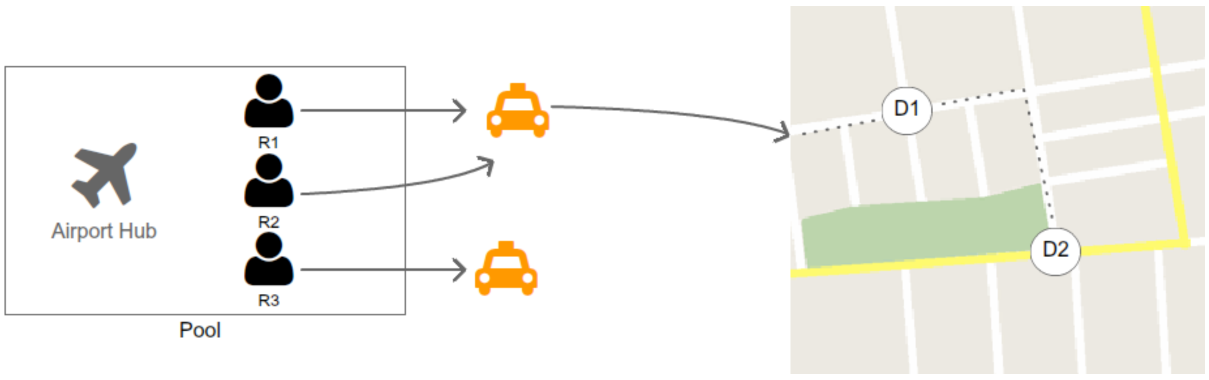

Figure 1: Static model example

In Figure 1, requests R1 and R2 are matched together and assigned to an empty vehicle which drops R1 first (D1 is the destination of R1) and then drops R2 (D2 is the destination of R2).

Within the dynamic model, new requests can be combined together and then assigned to a vehicle which may be still driving some other request, thus a new route has to be computed in order to satisfy all the requests involved.

This is a common scenario in big cities where people request a taxi from various pick up locations and, thus, vehicles route dynamically change.

Below is a figure which represents new request arrival and vehicle re-routing:

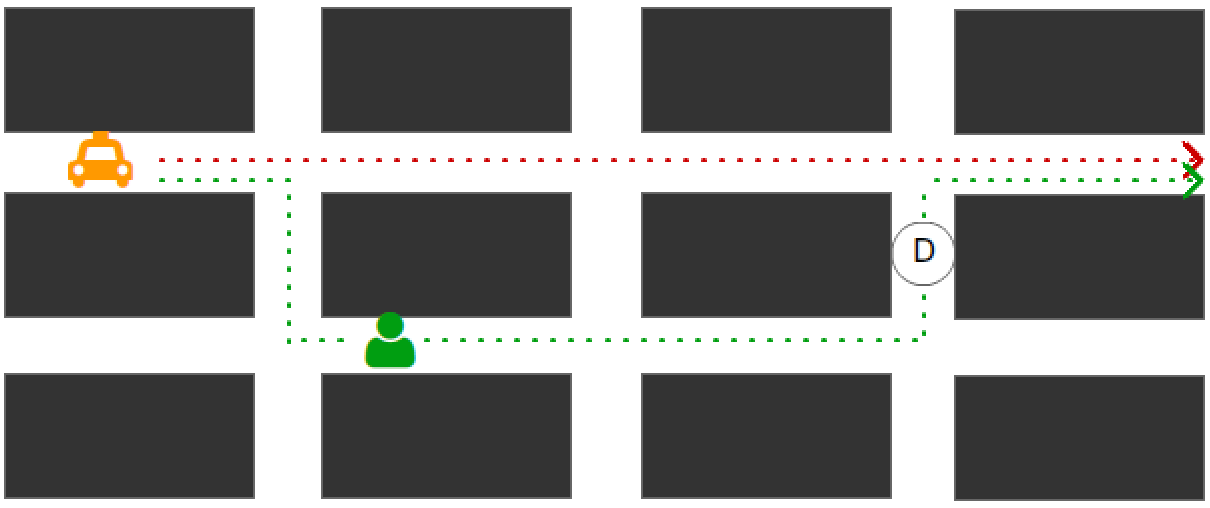

Figure 2: Dynamic model example

In Figure 2, the initial route of the vehicle is displayed with red dots (in the direction of the arrow), but since a new request is issued and is matched to this vehicle, then a re-route which includes pick up and drop off for the request is necessary: the updated route is shown with green dots and "D" represents the drop off point of the request.

In this thesis, for the comparison between the "fair" and "optimum" plans, we consider the static model alone and a combination of static and dynamic model: requests can register to a particular pool but can be assigned to already existing trips processed by some vehicle.

1.4 Ridesharing graph

Each pool of requests can be represented as a weighted graph (as also done in [2], [3]), where a node represents a request and an edge between two requests represents a potential shared trip. To each shared trip is attached a weight which symbolizes the “saving” the trip provides. See Figure 3 below for a graphical example.

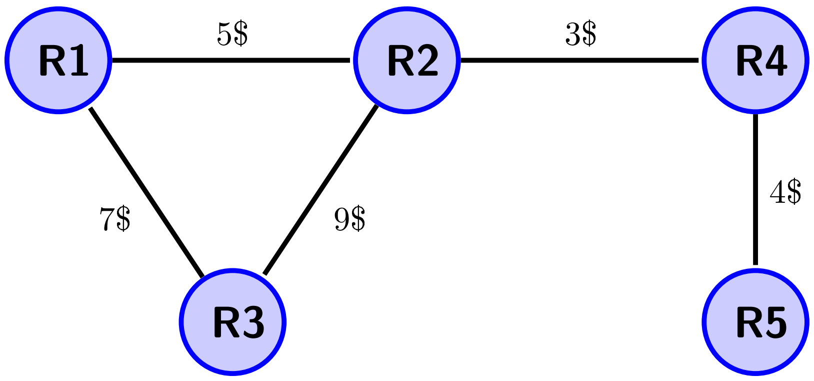

Figure 3: A ridesharing graph

If more than two requests can be combined together, then we have a ridesharing hypergraph: nodes still represent requests and hyper-edges represent a potential shared trip among the connected (two or more) requests.

In the case of the dynamic model, a non-empty vehicle can be assigned to a potential shared trip, thus the requests it is still processing (i.e., requests on board) are represented as nodes and connected by a (hyper) edge to the shared trip.

1.5 Fairness in ridesharing

In this section we introduce the concept of “fairness” in ridesharing, but in order to understand this notion, it is clearer to start from the concept of “unfairness”. We report the definitions as conceived in [2].

Unfairness (definition): “given a ride sharing graph RSG(V,E), a ride sharing plan (rsp) R is unfair to a request $v_i \in V$ if: 1) there exists another request $v_j \in V$ and the edge $e_{i,j} \in E$ is not in R; and 2) v_i and v_j both incur a higher individual saving if ridesharing with each other than with their partners in R”.

Fairness (definition): “an rsp F is fair if it is not unfair to any request”.

In order to comply with the definition of “fairness”, there are two different options to compute the individual saving in a shared trip (i.e., the amount of money saved by a particular request when ridesharing in this trip): total saving originated by shared trip (which is the weight applied to the edge connecting different requests) may be either evenly split among the single requests involved or unevenly split with a specific technique.

Evenly split (definition): total saving of a shared trip is equally split among the requests involved; the resulting plan will be "fair" to the single edge (shared trip) with respect to all the others.

Unevenly split (definition): total saving of a shared trip is not equally split among the requests involved; the resulting plan will be "fair" to the single request with respect to all the others.

A possible approach to deal with unevenly split savings is to distribute "total saving" with respect to the increase in the distance traveled, compared to the shortest path.

For instance, given two combinable requests A and B, assume that destination of A is n_1 miles away along the shortest path, and in the joint path it travels m_1 miles. For B, assume that the corresponding figures are n_2 and m_2.

Let:

$$x = \frac{m_1}{n_1}$$

$$y = \frac{m_2}{n_2}$$

Where "x" (resp. "y") represents the increase in miles traveled for request A (resp. B) with respect to its shortest path.

Thus, the following are the fractions of the overall saving for the joint trip to be assigned to the two requests:

$$Saving(A) = \frac{x}{x+y} * TotalSaving$$

$$Saving(B) = \frac{y}{x+y} * TotalSaving$$

It is worth noting that, as devised in [2], both the “fair” and the “optimum” plans have to be computed; the “optimum” plan will be eventually issued but, since it may be “unfair” to some request, a compensation scheme derived from the “fair” plan is applied.

In this thesis, with respect to all the above definitions, we assume for simplicity that there is no tie in the ridesharing graph. More specifically, within the context of evenly split savings, no edge (i.e., shared trip) in the ridesharing graph can provide the same saving; within the context of unevenly split savings, no request can have the same individual saving when participating to different shared trips (i.e., request must have a strictly ordered list of preferences in ridesharing partners).

This is due to the fact that no sufficient study has been conducted towards this direction of a “fair” approach to break the ties, which would require extensive analysis supported by thorough experiments.

1.6 Ridesharing graph matching

Given a ridesharing graph as defined in section 1.4, it is possible to produce a ridesharing plan by finding a matching in the ridesharing graph (i.e., a set of edges without common nodes).

Hence, within our context, this means that no request (node) can participate in more than one shared trip (edge): it can be either unmatched (i.e., no ridesharing) or matched only once.

1.6.1 Maximum weight matching

This type of matching produces the optimum ridesharing plan, which maximizes total savings at a global level and does not take into consideration "fairness" to requests.

In Figure 4, four requests participate in ridesharing and the edges selected with this matching are colored in green: total saving is 15$.

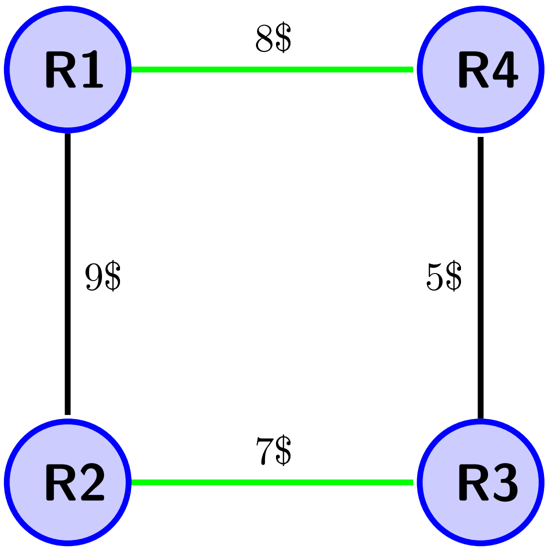

Figure 4: Maximum weight matching of a ridesharing graph

1.6.2 Evenly split fair matching

If total saving for a shared trip is evenly split among the involved requests, then a “fair” matching would be the following (edges selected for the matching are colored in green):

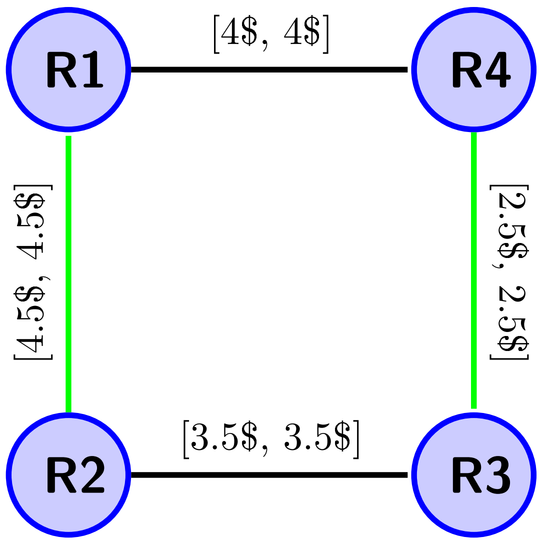

Figure 5: Evenly split fair matching of a ridesharing graph

On each edge (A, B) between requests A and B, a label which encloses two values in square brackets is applied: the value closest to request A (resp. request B) indicates the individual saving of request A (resp. request B) in this shared trip.

We can see that requests R1 and R2 both prefer being matched with each other because this results in higher saving for them (4.5$) with respect to the other options. Moreover, R3 (resp. R4) would prefer having R1 (resp. R2) as partner in ridesharing, but since it would be unfair to R1 (resp. R2) to participate in some other shared trip, then it is fair to combine R3 and R4 together.

It is worth noting that, with the same ridesharing graph, the (evenly split) "fair" plan is different from the "optimum" plan (Figure 4): in this case, total saving is 14$.

1.6.3 Unevenly split fair matching

If total saving for a shared trip is split unevenly among the involved requests, then a "fair" matching would be the following (green edges represent the edges in the solution):

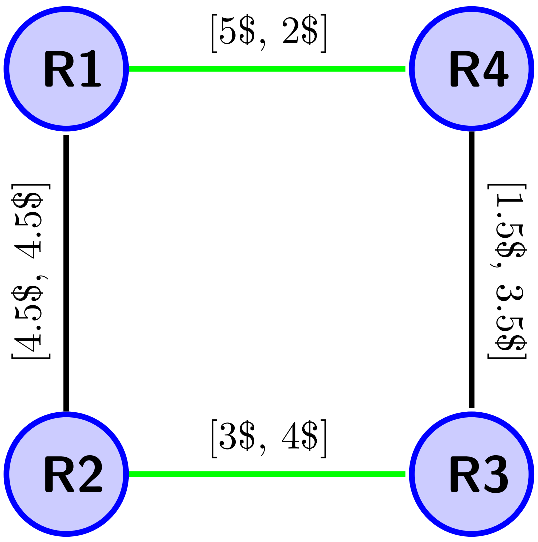

Figure 6: Unevenly split fair matching of a ridesharing graph

Same notation used for Figure 5 holds here.

We can see that R1 and R4 are partners in ridesharing since they both prefer being matched with each other; R2 would prefer ridesharing with R1, but this would be unfair to R1, thus it is eventually matched with R3.

It is worth noting that, with the same ridesharing graph, the (unevenly split) “fair” plan is different from the “fair” plan with the assumption of splitting evenly the savings among the requests (Figure 5). Moreover, it is also noteworthy that a ridesharing plan with unevenly split savings may not be always possible, which means that Nash equilibria cannot be found:

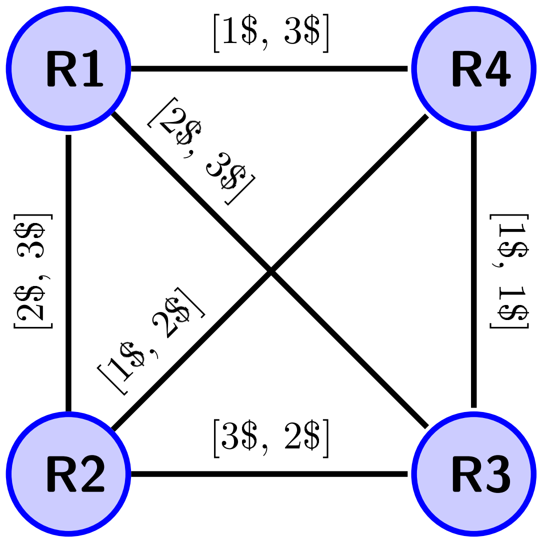

Figure 7: Unevenly split fair matching of a ridesharing graph with no solution

Same notation used for Figure 5 holds here.

Given the ridesharing graph shown in Figure 7, consider a plan which matches R1-R4 and R2-R3; this plan is unfair to R1, because both R1 and R3 would save more if partnering together with respect to what they save with their partners in this plan. Similarly, if R2 or R3 is matched to R4 rather than R1, the plan would be unfair to the partner of R4.

1.7 Related work

As far as static model is concerned, [4] devises an unweighted ridesharing graph and provides experimental results in Manhattan area which supports the idea that, even with a static implementation and only merging two requests at a time, huge benefits derive in terms of % shared trips. It is also shown that if a weight (e.g., total travel time) is applied to each edge of the ridesharing graph, then it is possible to minimize overall travel time.

In addition, also [5] considers the static case, but with requests originating from the same hub ("LaGuardia" airport) and with destination in New York City, still adopting an unweighted ridesharing graph. They proposed an approach which combines multiple drop-off points ridesharing and slugging (i.e., walking for the purpose of ridesharing): they demonstrated that this combination reduces the total number of trips to 25%-40%.

In regard to dynamic model, [3] proposes an any-time optimal approach which is capable to find the optimum solution to the problem of minimizing travel delays, but due to time needs, a suboptimal solution is found. They consider ridesharing for any vehicle capacity: results in terms of % shared trips are shown for vehicles with 2, 4 and 10 passenger seats.

1.8 Thesis structure

The following chapters of this thesis are organized as follows:

- *Chapter 2 - Data organization*, which describes the database schemas and how they can be effectively used for taxi ridesharing purposes, addressing optimization issues;
- *Chapter 3 - Ridesharing algorithms*, which presents the algorithm for both the optimum plan and the fair plan in specific case studies;
- *Chapter 4 - Experiments settings*, which includes the settings for the experiments and states any assumption made;
- *Chapter 5 - Results analysis*, which shows plots aiming at comparing the algorithms presented in chapter 3 with respect to the settings provided in chapter 4;
- *Chapter 6 - Conclusions and future work*, which concludes this thesis and provides some perspective for future works.

CHAPTER 2

DATA ORGANIZATION

2.1 Overview

The DBMS which has been employed is “PostgreSQL 9.6” with “PostGIS” spatial extension, in order to allow geometrical types and purposefully represent “road intersections” and “road segments” in geographic longitude and latitude coordinates.
We show in Figure 8 the Entity-Relationship diagram for our database: rectangles indicate entities; entity attributes are listed inside the rectangle (primary key is underlined); edges indicate relations; given an edge between “X” and “Y” entities, multiplicity of “Y” is indicated at “X” endpoint and vice-versa.

Schemas description is provided:

- *Road Intersection*: indicates the intersection between two streets and is represented by a point in the space (longitude and latitude coordinates), stored in the attribute “geom”; spatial indices are created on “ID” (B-tree) and on “geom” (GiST);
- *Road Segment*: is a fraction of a street and connects two adjacent “road intersections”; is labeled with “travel time” (minutes to drive from “source intersection” to “target intersection”) and “distance miles” (to compute distance saved in trips); is represented

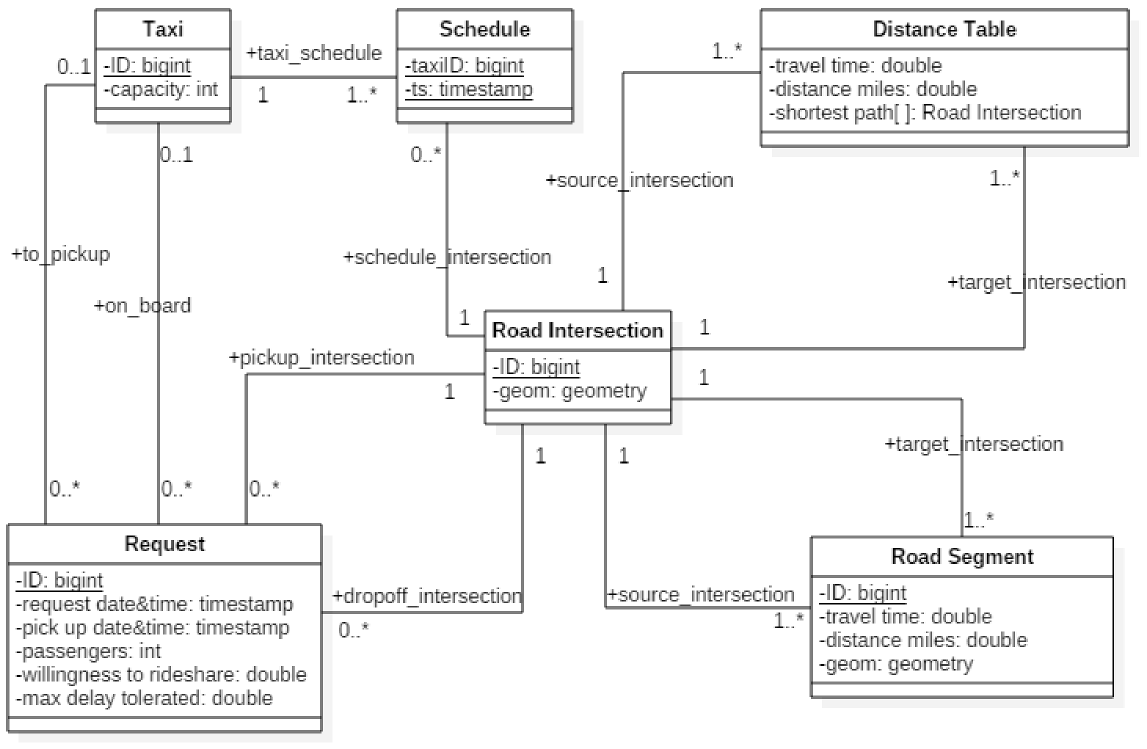

Figure 8: ER diagram

by a line in the space, stored in the attribute "geom"; together with "Road Intersection" schema, represents "Road Network";

- *Distance Table*: "source intersection" and "target intersection" represent the primary key; stores the records related to the shortest path between any two "road intersections" in the road network; overall "travel time", overall "distance miles" and all hops in "shortest path" are retained;

- *Request*: contains information about request issued time and actual pick up time; can be either “on board” or “to pick up” for only one vehicle or not assigned at all; “pick up intersection” and “drop off intersection” are assigned based on a mapping from actual locations to existing “road intersections”; “willingness to rideshare” indicates the probability for the request to participate in ridesharing; “maximum delay tolerated” indicates the fraction of the time to get to destination without ridesharing (i.e., the shortest path) such that the request will participate in ridesharing only if, in the shared trip, request drop off is delayed by at most this fraction; spatial index on “request date&time” (B-Tree);
- *Taxi*: represents a vehicle in the system; can have “requests” either already on board or to pick up; its current location is represented by the first record in its “schedule”, thus, for later computations aimed at checking constraints satisfiability, the interval between “current time” and “timestamp” associated to the first record in “schedule” is also considered;
- *Schedule*: is a weak entity for “Taxi” and is considered a separate entity (instead of being a relationship between “Taxi” and “Road Intersection”) because “timestamp” is a fundamental attribute, since each taxi cannot be located in two different “road intersections” at the same time; thus, contains, for each “Taxi”, its route through the “road intersections” ordered by related “timestamp”; every time a new pool starts, it is possible to delete records for old schedules (i.e., a “taxi” has already traversed some “road intersections”, thus it is not necessary to keep that information anymore).

2.2 Road network: tailoring OSM data

"Open Street Map" (OSM) project aims at creating maps of the world with individual contribution and free employment of the available data.

But, for our purposes, it is not possible to employ OSM data as they are: we need to refine them in order to retrieve, in particular, "road intersections" and "road segments" for a specific region.

First of all, it is worth noting that, for each street, OSM provides an ordered set of nodes which belong to that street: these nodes can be either intersections or not.

The approach which has been followed here is to check, for any node belonging to a street, if it belongs also to some other street: if this is the case, then that node is a road intersection.

Moreover, in this way bridges and underpasses are properly handled: OSM takes into consideration, beyond longitude and latitude, also altitude and therefore would not regard streets at different levels as intersected.

After retrieving all the road intersections, we need to connect them with road segments in order to generate a full road network. Thus, streets are first split into single edges which connect two adjacent nodes (either intersections or not); after that, consecutive edges are merged together into a road segment in order to connect two adjacent road intersections.

It is worth noting that this process retains the information related to travel time and distance in miles: travel time (respectively, distance in miles) of a road segment is equal to the sum of travel times (respectively, distances in miles) of each combined edge.

Moreover, the constructed road network is portable: "road intersections" and "road segments" tables can be exported in any format and computations on the graph can be performed with any preferred implementation.

As a consequence, distance table construction for a given road network can be performed with any shortest path computation implementation. Here we show the road networks constructed and used for the static model (sections 3.1-3.2) and for the dynamic model (section 3.3):

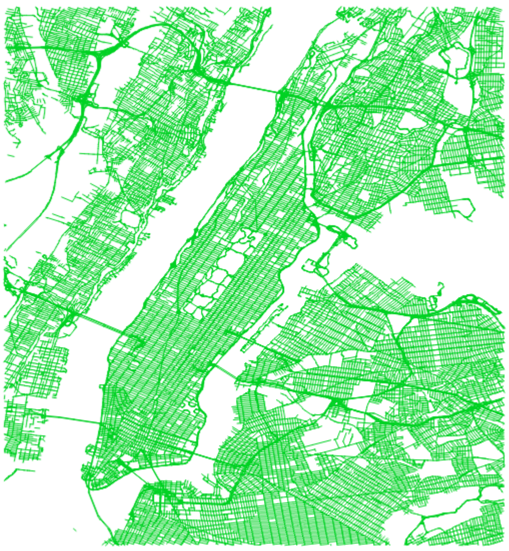

Figure 9: Static model: New York City road network

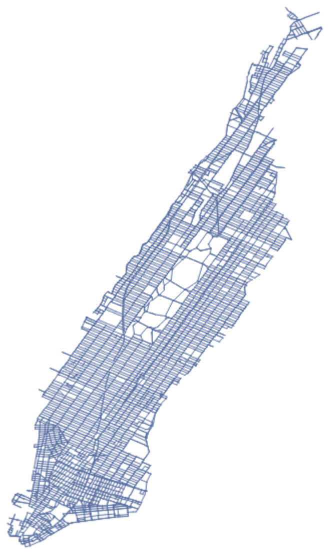

Figure 10: Dynamic model: Manhattan road network

2.3 SQL queries

The following queries can be executed with "PostgreSQL 9.6" and performance is sped up by creating proper indexes as suggested in section 2.1.

2.3.1 Longitude/latitude coordinates to road intersection mapping

Each request has actual pick up and drop off coordinates, expressed in longitude and latitude: for our purposes, we would like to map pick up and drop off locations to existing "road intersections".

Thus, given longitude "x_long" and latitude "y_lat" coordinates in the space, it is possible to map this geometric point to the closest "road intersection":

TABLE I: SQL QUERY: MAPPING LONGITUDE/LATITUDE COORDINATES TO ROAD INTERSECTION

```
SELECT id
FROM road_intersections
ORDER BY geom <-> st_setsrid(st_makepoint(x_long, y_lat), 4326)
LIMIT 1;
```

In particular, "$< - >$" geometric operator means "distance between" two geometries. But there exist different spatial reference systems (e.g., Cartesian and Geographic coordinate systems) in which distance calculation is different: distance between two points on a plane is different than distance between the same two points on a spheroid (e.g. the Earth). Thus, we need to set our spatial reference system to geographic coordinate system ("st_setsrid" function), which employs longitude and latitude: in particular, this reference system is identified by "4326" value.
In addition, "st_makepoint" function creates a point in the specified reference system with coordinates "x_long" and "y_lat".
As a result, the above SQL query orders road intersections by increasing distance to the new point generated: only the first one (i.e., the closest road intersection) is retrieved.

2.3.2 Pool of requests retrieval

At the beginning of a new pool, requests which have been issued within this time interval need to be gathered together and efficiently retrieved from the database. Thus, with an index based on “request_date&time”, given a pool which spans from “start_pool_time” to “end_pool_time”:

TABLE II: SQL QUERY: RETRIEVE REQUESTS FOR A SPECIFIC POOL

```
SELECT *
FROM requests
WHERE request_datetime >= ‘start_pool_time’
AND request_datetime < ‘end_pool_time’;
```

2.3.3 Distance table look up

This is the most frequent query which is executed either to check constraints satisfiability or to update the schedule of a taxi. In order to retrieve the information related to the shortest path between two road intersections (respectively identified by “source_id” and “target_id”), we have:

TABLE III: SQL QUERY: DISTANCE TABLE LOOK UP

```
SELECT *
FROM distance_table
WHERE source = ‘source_id’ AND target = ‘target_id’;
```

For what concerns the static model context, our distance table stores only the distance between the considered hub node and any other intersection (i.e., “source_id” is the ID of the considered hub node); in addition, the information needed is: time and miles to travel to reach destination from the hub node.

As opposed to the static model, for the dynamic model our distance table stores information between any two intersections; this information includes: time and miles to travel to reach destination from source; ordered sequence of hops (i.e., intersections) in the shortest path to destination.

CHAPTER 3

RIDESHARING ALGORITHMS

The idea is to provide a degree of commonality among the algorithms which produce an optimum and a fair ridesharing plan, and this can be achieved with the standardization of the ridesharing graph.
These algorithms can be applied to two different models:

- *Static*: new taxi requests are grouped together in a pool of fixed size (e.g., 5 minutes) and merged rides are generated taking into account only pending requests in that pool;
- *Dynamic*: new taxi requests are attempted to be merged to already existing taxi rides, otherwise a new taxi driver will be dispatched.

In order to outline the case studies oriented to the comparison between the optimum plan and the fair plan, it is necessary to understand whether they are the same or not in the unweighted ridesharing graph case: if they are, there is nothing to compare and we can move directly to the weighted case. In the following, we prove that this is actually the case.
In the case of an unweighted ridesharing graph (RSG), “fairness” definition (described in section 1.5) does not have to be modified if we assume to apply the same weight to each edge: each request can be merged to any of its neighboring requests, because all the ridesharing options result in the same total saving.

The following is the theorem we want to prove.

Theorem: in the case of an unweighted RSG, the optimum ridesharing plan is also fair.

(Proof)

In order to prove this theorem, it is useful to prove first that any ridesharing plan constructed with Algorithm NE4.1 ([2] and reported in section 3.1.2) is fair.

In fact, since all edges have the same weight, step 2.a) of the algorithm can be seen as "pick a random edge in the remaining RSG", which results in any random ridesharing plan.

Since Theorem 2 ("for an evenly split RSG, the rsp F computed by Algorithm NE4.1 is fair") in [2] has been proven, then we can infer that any random ridesharing plan constructed from an unweighted RSG, with Algorithm NE4.1, is fair.

By proving this, the optimum plan is only one among all the random ridesharing plans, and since all of them are fair, it follows that the optimum plan is also fair.

In the following subsections, ridesharing algorithms for each case study are presented.

3.1 Static model: combination of at most two requests originating in single hub

In this case, only requests which are picked up in a specific location are considered to be merged: ridesharing is allowed to combine only two requests at a time and the resulting trip will be assigned to an empty taxi located in the same hub.

Since, in this case study, only two requests can be merged at a time, the resulting ridesharing graph will be an undirected weighted graph.

3.1.1 Optimum matching algorithm

The optimum plan can be computed with any maximum weight matching: edges in the final solution maximize the sum of weights in the ridesharing graph while considering the constraint that no request is incident to more than one edge present in the solution.

Below is the pseudo-code of the algorithm:

TABLE IV: STATIC MODEL (MERGING AT MOST TWO REQUESTS): MAXIMUM WEIGHT MATCHING ALGORITHM

Input: Ridesharing graph RSG = (R, T)
Output: The set S of (either combined or not) requests
1. S = R 2. Build a maximum weight matching M_{max} on RSG 3. **for each** $T_{i,j}$ in M_{max} **do** 4. S = S $\cup$ $\{R_{i,j}\}$; S = S $\setminus$ $\{R_i, R_j\}$ 5. **return** S

The set S is initialized with the set R of all the requests in a pool. Thus, given a ridesharing graph RSG with requests in R as nodes and edges $T_{i,j}$ in T (an edge $T_{i,j}$ exists if requests R_i and R_j can be combined) to which a weight is applied, then it is possible to compute a maximum weight matching M_{max} on RSG. For any edge $T_{i,j}$ in M_{max}, the combined trip $R_{i,j}$ is included in the solution S, whereas the single requests R_i and R_j are excluded.

As suggested in [4], Edmonds matching algorithm ([6]) for the weighted case has been employed and it yields $O(n^2 log\ n)$ time complexity.

In particular, "Lemon" C++ library has been used because it provides an implementation of the aforementioned maximum weight matching algorithm.

3.1.2 Evenly split fair matching algorithm

In this case, finding Nash equilibria, given a ridesharing graph RSG with no edges labeled with the same weight, is always possible (as demonstrated in [2]).

A fair ridesharing plan is computed iteratively by combining two requests at a time connected by the heaviest edge in RSG, and removing them from RSG (along with their outgoing edges), until there are still edges available in RSG.

This is meaningful because if we split the weight of the edges in an even way, then the heaviest edge $T_{i,j}$ implies that request R_i prefers being matched with request R_j and vice versa: the same holds for the following iterations.

Thus, the fair rsp F is computed with the following algorithm (NE4.1 in [2] and reported here):

TABLE V: STATIC MODEL (MERGING AT MOST TWO REQUESTS): EVENLY SPLIT FAIR MATCHING ALGORITHM

Input: Ridesharing graph RSG = (R, T)

Output: Ridesharing plan rsp F

```
1. F = {}
2. while there are still edges in RSG do
    (a) Retrieve the heaviest edge $T_{i,j}$ in the remaining RSG
    (b) F = F ∪ $T_{i,j}$; Remove requests $R_i$ and $R_j$ and their incident edges from RSG
```

Time complexity of the above algorithm is *O(n log n)*, where n is the number of nodes in RSG (see [7]).

3.1.3 Unevenly split fair matching algorithm

If weight on each edge of the ridesharing graph RSG is not split evenly, then the algorithm devised in the previous section does not work anymore, because it is not necessarily fair within this context.

To each pair of combinable requests is applied a weight (total savings), but different fractions are related to the single request involved: this fraction is called “individual saving”.

In this case, [2] introduces an algorithm to compute a fair ridesharing plan, which will use the solution found to the “Stable Roommates Problem” (SRP). It has been used an implementation which employs constraint programming and whose documentation can be found in [8];

in particular, it solves an extended SRP: “SRP with incomplete lists”, which means that the ridesharing graph does not have to be complete and, therefore, requests do not have to specify a preference for an unconnected request.

The algorithm works as follows. Each request R_i in RSG sorts its neighbors (possible partners in the ridesharing plan) by individual saving in descending order; a solution to the SRP is found in $O(n^2)$, where n is the number of requests (see [9]).

It is worth noting that, as shown in section 1.6.3, a solution to the SRP cannot always be found. Thus, if there is no solution for the algorithm, then the optimum plan payment scheme will be issued.

TABLE VI: STATIC MODEL (MERGING AT MOST TWO REQUESTS): UNEVENLY SPLIT FAIR MATCHING ALGORITHM

Input: Ridesharing graph RSG = (R, T)

Output: The set S of (either combined or not) requests

```
1. S = R
2. Build a SRP $M_{SRP}$ on RSG
3. if $M_{SRP}$ has no solution
4.     Apply maximum weight algorithm (Table IV)
5. else
6.     for each $T_{i,j}$ in $M_{SRP}$ do
7.     S = S ∪ {$R_{i,j}$}; S = S \ {$R_i$, $R_j$}
8.     return S
```

3.2 Static model: combination of “k” requests originating in single hub

In this case, only requests which are picked up in a specific location are considered to be merged: ridesharing is allowed to combine “k” requests at a time (with “k” between 2 and taxi maximum capacity) and the resulting trip will be assigned to an empty taxi located in the same hub.

Since, in this case study, even more than two requests can be merged at a time, the resulting ridesharing graph will be an undirected weighted hyper-graph.

3.2.1 Optimum matching algorithm

Optimum matching in a hyper-graph is a NP-complete problem, but there are some approximations mentioned in [4] which are solvable in polynomial time. However, any approximation (even if closer to the exact optimum solution) may not be enough to understand the real difference between the optimum and the fair plans.

Since our aim is to compare the fair plan to the exact optimum plan, a naive approach is to try any combination of edges in the ridesharing hyper-graph and, among the feasible combinations, check which one provides the highest weighted matching. By “feasible combination” we mean that no request appears in more than a single hyper-edge present in the combination.

The main issue of this approach is that it is time and memory consuming. Thus, a cleverer approach is required to retrieve the same results while saving both time and memory.

This approach employs a “branch and bound” technique and is inspired to the solution to “Graph-Constrained Coalition Formation” (GCCF) problem given in [10] and [11]. Hyper-edges can be seen as single “coalitions” of requests they connect, whose value is “total savings”: the objective is to find the coalition of hyper-edges which maximizes “total savings”.

3.2.2 Evenly split fair matching algorithm

Also in this case, finding Nash equilibria, given a ridesharing hyper-graph RSH with no edges labeled with the same weight, is always possible (as demonstrated in [2]).

A ridesharing plan F is fair if there is no hyper-edge in the ridesharing hyper-graph, absent in the solution, in which all the incident requests would have a higher individual saving than in F. Since hyper-edge total saving is evenly split among the related requests, then the individual saving is computed as total saving divided by the number of incident requests.

A fair *rsp* is computed equivalently as stated in 3.1.2.: hyper-edges are sorted by individual saving in decreasing order and, iteratively, the heaviest hyper-edge in the remaining RSH is selected and its related requests are removed from RSH (along with their outgoing hyper-edges), until there are still hyper-edges available in RSH.

This is meaningful because if we split the weight of the hyper-edges in an even way, then the heaviest hyper-edge t in T (in terms of individual saving) implies that all the related requests prefer being matched with each other: the same holds for the following iterations.

It is worth noting that, at some iteration of step 2.a, it is possible that the algorithm will select a hyper-edge whose total weight is lower than another hyper-edge still present in RSH: this is because its individual saving is higher due to a lower number of incident requests. This scenario cannot happen in 3.1.2, instead.

Thus, the fair rsp F is computed with the following algorithm:

TABLE VII: STATIC MODEL (MERGING MORE THAN TWO REQUESTS): EVENLY SPLIT FAIR MATCHING ALGORITHM

Input: Ridesharing hyper-graph RSH = (R, T)

Output: Ridesharing plan rsp F

```
1. F = {}
2. while there are still hyper-edges in RSH do
    (a) Retrieve the edge $t$ with the highest individual
        saving in the remaining RSH
    (b) F = F ∪ {$t$}; Remove requests connected to $t$ and
        their incident edges from RSH
```

3.2.3 Unevenly split fair matching algorithm

If we split the weight of hyper-edges in the ridesharing hyper-graph in an uneven way, then the problem of finding a fair rsp, if there exists one, becomes NP-complete: this can be demonstrated by reduction from “Stable roommates problem with triple rooms” (3D-SR); see [12] for demonstration.

Thus, some approximations should be devised to deal with this approach.

However, as shown in section 5.1, within the context of combining only two requests at a time originating from a single hub, there is not a valuable difference between the evenly split and the unevenly split fair plans: as a consequence, it is reasonable to compare the optimum plan to the evenly split fair plan only, because it is computationally tractable, whereas the unevenly split case is intractable.

3.3 Dynamic model: requests assignment to already existing rides

In this case, new requests are combined to already existing rides: these taxis can be either already processing some other requests (i.e., they already have passengers on board) or be empty; in the former option, taxi schedule is required to be re-routed.

3.3.1 Optimum matching algorithm

This algorithm is an adaptation of the one devised in [3].

Due to limited computational budget for real time analysis, [3] provides some heuristics, in order to make this problem tractable, such as: filtering the number of edges in the RV graph; setting a time out for each vehicle considered in the RTV graph construction. These two graphs will be accurately described shortly.

Moreover, since it satisfies the any-time property, then the algorithm increments an initial solution and thus provides a suboptimal solution: if there were enough time, then it is guaranteed that the optimum solution would be found.

However, our main interest is to compare the fair plan to the exact optimum plan; as a result, no heuristic will be employed in order to retrieve the optimum solution.

Given a pool of requests and a set of vehicles with their current state, it is first checked which requests can be pair-wise combined (i.e., there exists a virtual vehicle, located in one of the two requests pick up intersection, which can process both requests while satisfying the maximum delay tolerated constraint); then, actual vehicles are considered and checked if they can process a single request from the pool. Thus, in the graph (called RV graph), vehicles and requests are the nodes, whereas an edge between two requests means that they can be combined and an edge between a vehicle and a request means that the vehicle can process the request.

The resulting graph is further examined by analyzing its cliques: each complete sub-graph which contains a vehicle is a potential trip and is checked whether it is feasible indeed.

Newly formed trips are then assigned to vehicles with respect to the optimum solution provided by an ILP problem, as later described.

Thus, the algorithm follows three steps.

- **Pairwise request-vehicle RV graph construction**

  RV graph gives an overview of which requests can be merged and assigned to which vehicle.

  In this graph, each request in the current pool is a node, and each vehicle is also a node. An edge is drawn between two requests A and B if a virtual vehicle, located in either A's or B's pick up location, can process them both while satisfying the maximum delay tolerated constraint.

  An edge is drawn between a request r and a vehicle v if v can pick up r at its origin location while satisfying the constraints of vehicle capacity and maximum delay tolerated for r and any other request still on board.

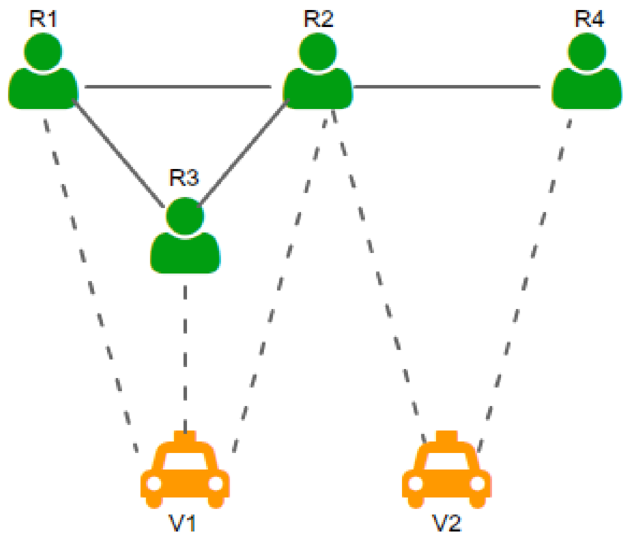

Figure 11: Dynamic model: RV graph example

In Figure 11 we provide an example of RV graph: it shows a combination of four requests and two vehicles.

In order to retain only the most promising vehicles and pruning certain edges for later faster computations, “Dual Side Taxi Searching” idea (which has been devised in [13]) is followed: for each request, we consider only those vehicles which are currently close enough to its pick up intersection and will be close enough to its drop off intersection. Current location of a vehicle is tracked as described in section 2.1: we retain only the future route of a vehicle, thus current location is the earliest record in the schedule. Analogously, given a timestamp *ts* at which a request is expecting to be dropped off (computed upon the shortest time from its pick up to drop off intersections), a vehicle future location at ts is its location immediately non-after ts as given in the schedule. Thus, before checking which vehicles can satisfy delay constraints for their on board requests and for a new request r, it is useful to first select only those vehicles which are currently in a specific location from which they can pick up request r while satisfying the constraint of maximum delay tolerated; secondly, within this set of vehicles, we only retain those whose future location at the earliest drop off time of request r (computed based on the shortest time to arrive at its destination) can reach request r destination within the maximum delay tolerated. For instance, assume we have a request r issued at “10:00:00” and its earliest drop off time (shortest time to destination) is “10:10:00”. A vehicle v can be checked if it is connected to request r if: from its current location at “10:00:00” it can pick up r within maximum delay tolerated, and; from its future location at “10:10:00” it can reach the destination of r within maximum delay tolerated.

The process of adding an edge between any two requests takes $O(n^2log(i^2))$, where “n” is the number of requests and “i” is the number of road intersections: for each combination of two requests, an edge is decided to connect them in $O(log(i^2))$ time, that is the indexed access time to the distance table which contains “i^2” rows. In particular, in our case, “i” for Manhattan is 3,933.

The process of adding an edge between a vehicle v and a request r takes $O(m\ n\ log(i^2))$, where “m” is the number of vehicles ,”n” is the number of requests and “i” is the number of road intersections (same reasoning for logarithmic look up applies). Moreover, vehicle capacity is assumed to be a small constant, so considering permutations of the schedule for a vehicle is bounded and small with respect to the number of vehicles and requests. Thus, the overall time complexity, assuming that vehicles are higher in quantity than requests are, is $O(m\ n\ log(i^2))$. This is the worst case when all the vehicles are close enough (i.e., are selected with the aforementioned concept of "Dual Side Taxi Searching") to all requests in the pool.

In order to improve overall performance, we have parallelized among the requests, to check whether two distinct requests can be combined, and among the vehicles, to compute which requests a vehicle can process.

- **Request-trip-vehicle RTV graph construction**

  The second step of the algorithm is to explore the cliques of the RV graph, that is, complete subgraphs of the RV graph such that one of their nodes is a vehicle.

  First of all, all feasible trips have to be listed.

  Each clique represents a potential trip T, that is a set of new requests which either may or may not be processed by the vehicle in the clique. Since this step can be exponential in the worst case, it is possible to specify a timeout per each vehicle to explore its related cliques (i.e., 0.2 seconds), which leads to sub-optimality of the solution but removes longer trips.

  This step proceeds incrementally by adding to the set of feasible trips all the single requests with their incident vehicles; after that, trips of size 2 are considered: if two requests can be combined together, then we can check which vehicles can process them both while satisfying capacity and delay constraints. After that, from this initial set of trips, it is possible to generate feasible trips of size “k”, where “k” is less than or equal to vehicle capacity; for a given vehicle, there exists a potential trip of size “k” if all of its sub-trips of size “k-1” are actually feasible: for this potential trip, then, it is checked whether the given vehicle can process it and, at the same time, can satisfy capacity and delay constraints. It is worth noting that a request r may appear in multiple feasible trips: the constraint for which r can appear at most once in the optimal plan is formalized in the ILP of next step.

Thus, in the RTV graph, there are three type of nodes: requests, trips, vehicles. Beyond the previously computed trips, also a “null” trip is added in order to specify the case in which a request cannot be processed by any vehicle.

There is an edge between a request r and a trip T if T contains r; each request r is then connected to the “null” trip.

There is an edge between a trip T (“null” trip is not considered) and a vehicle v if v can process T and on board requests while satisfying capacity and delay constraints.

Moreover, for each of the edges between trips and vehicles, we retain information about the new schedule and, also, a weight is applied: “total savings” if trip T were assigned to vehicle v: if multiple routes can satisfy the constraints, then the one with highest total savings is chosen.

This is different than what stated in [3], because, in this reference, the authors’ aim is to minimize the sum of delays (which is the weight they are considering).

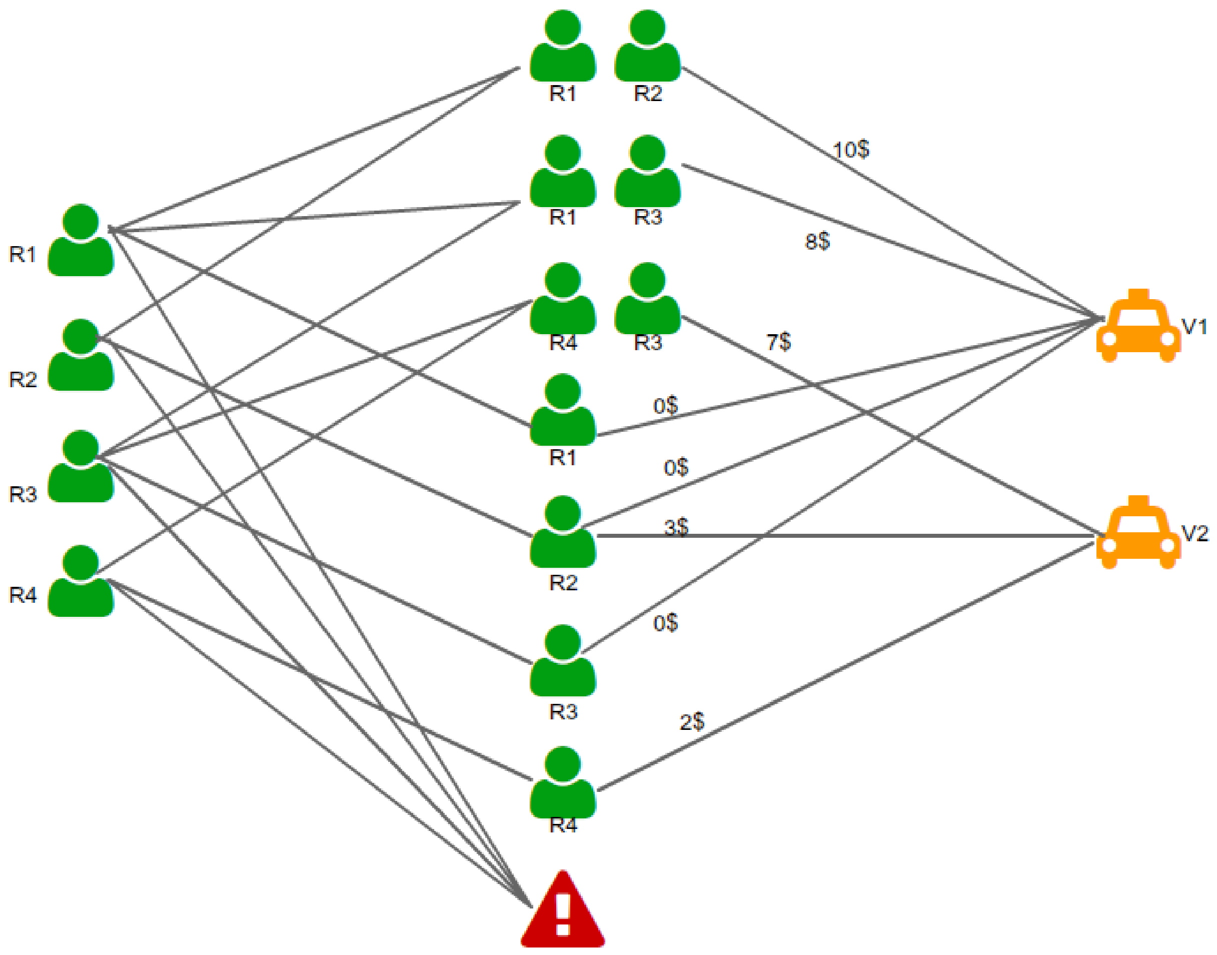

Figure 12: Dynamic model: RTV graph example

In Figure 12 we can see a possible result drawn from RV graph displayed in Figure 11; cliques in the RV graph which resulted in not being feasible are: R1-R2-R3-V1 and R2-R3-V1.

As we can see, on the left part of the graph, the four requests are connected to the trips (plus the "null" trip) which contain them. Moreover, between each trip and related vehicle there is a “total saving” applied: assuming that V1 is empty, while V2 is not, we can notice that some assignments (e.g. trip {R1} to V1) provide no saving: this is due to the fact that, since V1 is empty and R1 is assigned to this vehicle, there is no benefit deriving from ridesharing.

In order to improve overall performance, we have parallelized among the vehicles to compute all the feasible trips which can be processed by the same vehicle.

- **Optimal assignment**

  This last step, given the RTV graph with feasible trips and related vehicles, computes an optimal assignment of vehicles to trips: this is formalized as an Integer Linear Program (ILP).

  This optimization problem is fed with an initial solution which can be computed greedily as follows. First of all, we sort the edges in the RTV graph between trips and vehicles in decreasing order. After that, it is possible to select these sorted edges from the heaviest to the lightest and insert them in the greedy solution if both hold: trip does not contain requests which have already been added to the greedy solution; vehicle is not already assigned in the greedy solution.

  Here we provide the ILP formulation:

TABLE VIII: DYNAMIC MODEL: ILP FORMULATION

1. Initial guess: $\sum_{greedy}$
2. $\sum_{optim} := \arg\max_{\mathcal{X}} \mathcal{C}(\mathcal{X})$,
   where $\mathcal{C}(\mathcal{X}) = \sum_{i,j \in \mathcal{E}_{TV}} c_{i,j}\epsilon_{i,j} + \sum_{k=\{0,\dots,n\}} c_{k0}\chi_k$
3. s.t. $\sum_{i \in \mathcal{I}_j^V} \epsilon_{i,j} \leq 1,\ \forall v_j \in \mathcal{V}$
4. $\sum_{i \in \mathcal{I}_k^R, j \in \mathcal{I}_i^T} \epsilon_{i,j} + \chi_k = 1,\ \forall r_k \in \mathcal{R}$

There are two binary variables in the ILP problem: $\epsilon_{i,j}$ is set to 1 if trip "i" is assigned to vehicle "j", 0 otherwise; χ_k is set to 1 if, in the solution, there is no trip which contains request "k" that has been assigned to any vehicle, 0 otherwise.

Costs $c_{i,j}$ are applied to $\epsilon_{i,j}$ and they represent the weight of the edge of the RTV graph which connects trip "i" and vehicle "j"; negative costs c_{k0} are applied to χ_k in order to penalize ignored requests. We denote the set of variables as $\mathcal{X}$.

In the above formulation: $\mathcal{E}_{TV}$ indicates the set of indexes {i, j} such that an edge between trip "i" and vehicle "j" exists; "n" is the number of requests; $\mathcal{I}_j^V$ indicates the indexes "i" for which an edge between trip "i" and vehicle "j" exists; $\mathcal{I}_k^R$ indicates the indexes "i" for which an edge between request "k" and trip "i" exists; $\mathcal{I}_i^T$ indicates the indexes "j" for which an edge between trip "i" and vehicle "j" exists.

Constraints of this ILP problem are: each vehicle can be assigned to at most one trip;

each request can be either assigned to a vehicle or not assigned at all.

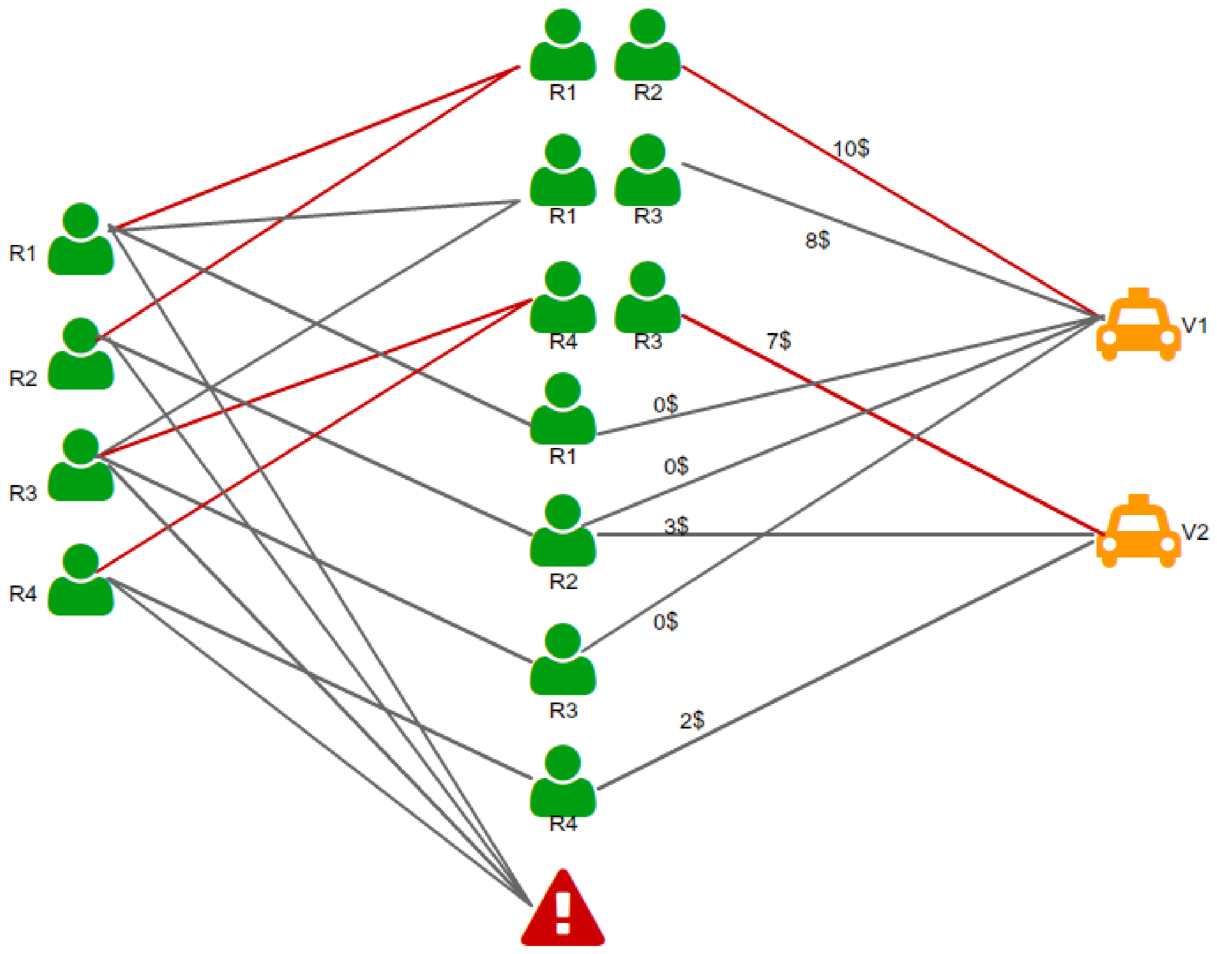

Figure 13: Dynamic model: optimal assignment example

In Figure 13 we present the solution to the problem shown in Figure 12.

In [3], the aim is to minimize the sum of delays, while in our case we want to maximize total savings; the objective function is modified accordingly: since we provide different costs, the only modification to apply is that the problem is now a maximization problem. It is worth noting that a penalty c_{k0} (negative value) has to be assigned in order to force the ILP to prefer a solution which assigns a trip "i" to vehicle "j" with related cost $c_{i,j}$ equals to 0.

In order to solve this problem, we have employed "Mosek" solver whose ILP optimization is inherently parallelized.

3.3.2 Evenly split fair matching algorithm

Given the RTV graph constructed as explained in section 3.3.1, only its sub-graph containing trips, vehicles and their edges needs to be analyzed (that we will call TV graph).

First of all, it is useful to prove the equivalence between the aforementioned sub-graph and the hyper-graph used for the evenly split fair matching algorithm for combining more than two requests (section 3.2.2).

What we would like to have is a hyper-graph with requests as nodes and that its hyper-edges represent a trip (among the connected requests); weight on the hyper-edge is "total savings" for the related trip assigned to a specific vehicle.

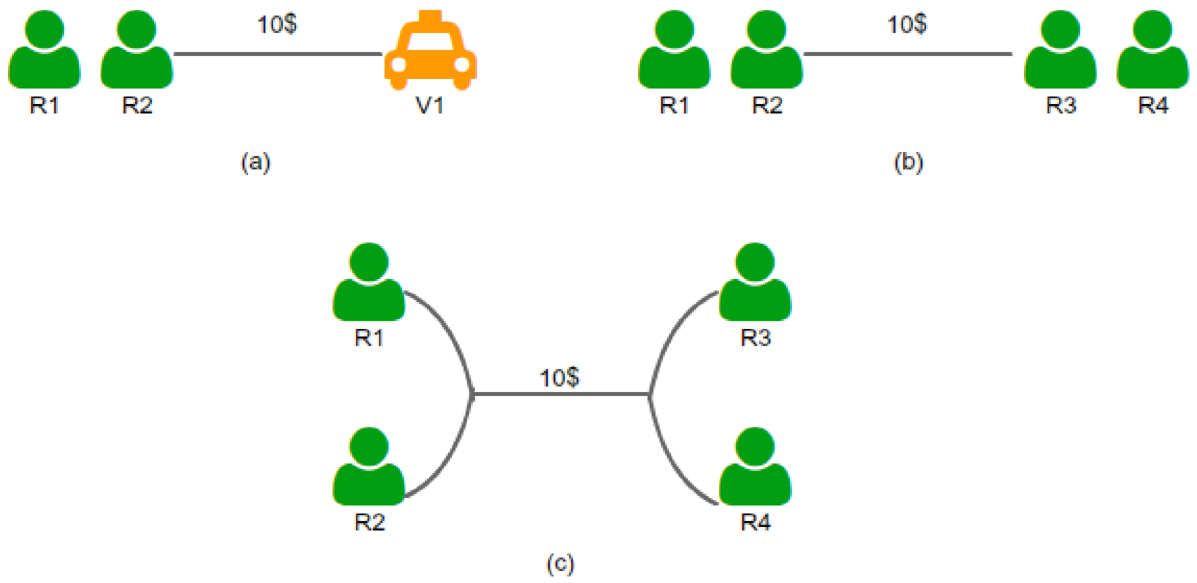

Figure 14: Dynamic model: TV graph standardization to ridesharing graph

Figure 14 graphically shows this translation for one edge of the TV graph.

Starting from one edge of the TV graph (Figure 14.a), it is possible to split each vehicle into n nodes, where n is the number of requests on board (Figure 14.b); at the same time, it is possible to split each trip into m nodes, where m is the number of combined requests for that trip. Finally, all these split nodes are connected by a hyper-edge with the same weight as stored in the RTV graph (Figure 14.c).

Thus, it is easy to check that from the TV graph representation it is possible to derive the ridesharing hyper-graph.

As a result, the algorithm to compute an evenly split fair plan is equivalent to the one devised in section 3.2.2.: edges of TV graph are sorted by individual saving in decreasing order and, iteratively, the heaviest edge in the remaining TV graph is selected and both the related vehicle and all the trips containing requests of the selected trip are removed from TV graph (along with their incident edges), until there are still edges available in TV graph.

Notice that individual saving is computed taking into account also requests on board of the vehicle.

TABLE IX: DYNAMIC MODEL: EVENLY SPLIT FAIR MATCHING ALGORITHM

Input: RTV sub-graph TV = ($T \cup V$, E)
Output: Ridesharing plan rsp F
```
1. F = {}
2. while there are still edges in TV do
    (a) Retrieve the edge $t$ with the highest individual saving in the remaining TV graph
    (b) F = F $\cup$ $\{t\}$; remove trips containing requests in $t$ and vehicle related to $t$ and their incident edges from TV graph
```

3.3.3 Unevenly split fair matching algorithm

After we translate the TV graph into a ridesharing hyper-graph (as shown in Figure 14), then we can split the weight of hyper-edges in an uneven way.

But the problem of finding a fair rsp, if there exists one, becomes NP-complete as already stated in section 3.2.3. Thus, some approximations should be devised to deal with this approach. However, in section 5.3, we present results related to the comparison between evenly and unevenly split savings when only two requests can be processed by a vehicle at the same time. The algorithm used in this context is the same as the one devised in section 3.1.3.

We have noticed, analogously to what shown in section 5.1 for the static model, that there is not a valuable difference between the evenly split and the unevenly split fair plans: as a consequence, it is reasonable to compare the optimum plan to the evenly split fair plan only, because it is computationally tractable, whereas the unevenly split case is intractable.

CHAPTER 4

EXPERIMENTS SETTINGS

In this section, how experiments have been set up is explained.

Firstly, general settings which are common to all the experiments are presented.

Secondly, we describe the specific settings related to static model experiments (single source cases for both combining at most two requests at a time and for more than two requests at a time).

Thirdly, we describe the specific settings related to dynamic model experiments.

4.1 General settings

New York City road network is extracted from "Open Street Map" data and only road intersections are retained. Thus, it is useful to generate road segments, connecting two different road intersections, which hold cumulative information (e.g., distance in miles and in time) of the streets they are a union of. The result is a directed graph with road intersections as nodes and road segments as edges.

With respect to "travel times", each road segment is first assigned a travel time with respect to the maximum speed on that type of road. In order to have a more realistic idea of the actual speed, as suggested in [5], a congestion fraction *cf* is applied to the maximum speed (*cf* range is (0.0, 1.0]), if we assume that the congestion fraction is the same for every road segment. Thus, driving time on a road segment is given by:

$$Driving\ time = \frac{road\ segment\ length}{maximum\ speed\ on\ road\ segment * cf}$$

We define a request r with the following tuple: pick up location, drop off location, request time (at which request is issued), latest acceptable pick up time, effective pick up time, expected drop off time, earliest possible time at which drop off location can be reached, number of passengers. Our aim is to provide results which are not related to a specific payment scheme, thus we apply to each merged trip a cost function that represents "miles saved".

The aforementioned cost function will be translated into dollars ($) so that it is possible to compute "total saving" for a merged trip: in this translation, we will consider only "cost per mile" fares for a NYC cab (2.50$/mile), without taking into account initial charge or slow traffic surcharges. See [14] for further information: passengers pay 0.50$ for each 1/5 mile, so the overall cost per mile is 2.50$.

The raw translation from "total miles traveled" to "total savings" has been proven to be effective: below we provide the proof.

Translation Mileage-Savings

Definitions:

$$sm_m = \text{total amount of money saved if requests were merged}$$

$$m_u = \text{total amount of money if requests were not merged}$$

$$C = \text{cost/mile}$$

$$sd_m = \text{total mileage saved if requests were merged}$$

$$d_u = \text{total mileage if requests were not merged}$$

(Proof)

$$\textit{\% fare reduction} = \frac{sm_m}{m_u} = \frac{C * sd_m}{C * d_u} = \frac{sd_m}{d_u} = \textit{\% mileage reduction}$$

The proof is guided by the fact that a cost C per mile has to be applied in order to compute a taxi trip fare, and this cost is assumed to be constant if fare is distance-based.

Direct consequence of the translation from "% reduction in mileage" to "% reduction in fare" is that we can conduct experiments with respect to distance traveled abstracting from fares, if they are assumed to be computed with constant cost per mile.

For the experiments, New York City taxi requests dataset has been employed. It stores over four years of taxi operations in NYC. For each trip, it provides, among the others: pick up and drop off locations; pick up and drop off times; passengers on board; actual travel time and distance. See reference [15] for more detailed information about this dataset.

Since, in the aforementioned dataset, date and time at which a request is issued is not present (but only time at which that request has been picked up), then we assume that "request date and time" is the same as "pick up date and time".

Origins and destinations of a single request are mapped to the closest road intersection in the road network considered: requests from database which have the same mapped origin and destination are filtered out.

Taxi capacity is normally set to 4 passenger seats, unless differently specified. As a consequence, requests can be combined if the sum of their passengers is less than or equal to taxi capacity.

4.2 Static model settings

The NYC road network of the area considered is a directed graph which consists of 59,792 intersections and 72,557 road segments; see Figure 9 to visualize the area of NYC which has been analyzed: it includes Manhattan and surroundings, with LaGuardia airport as well.

For this case study, only requests originating in LaGuardia airport are considered.

In order to speed up the experiments, the shortest path between LaGuardia road intersection and any other road intersection is pre-computed and stored into a distance table: shortest path between any two other road intersections is computed at run time.

It is assumed that taxi fleet in LaGuardia can always satisfy the demand of requests and all of them are empty.

In this case study, total distance saved for a combination of n requests is given by:

$$\textit{Total distance saved} = \left(\sum_{i=\{1,\ldots,n\}} miles_{sp,i}\right) - \arg\min_{j}(miles_{mp,j}),$$

where $miles_{sp,i}$ indicates the miles to be traveled by request i from LaGuardia airport to its destination, following the shortest path, and $miles_{mp,j}$ indicates total miles to be traveled with trip combination j (given m possible combinations): we choose the shortest combination in miles in order to maximize distance saved for that trip.

Results presented in sections 5.1 (combination of at most two requests) take into account pools of requests issued in 2013 (each day from 7am to midnight) from LaGuardia to the previously defined NYC area. Overall, there have been analyzed about two millions requests.

For what concerns the combination of more than two requests at a time, 100 random pools of requests issued in 2013 have been analyzed.

For these experiments, we consider the following parameters: willingness to ride share (passengers willingness to take part into ride sharing); maximum delay tolerated (devised as a fraction of the shortest time to arrive at destination, given by the shortest path from LaGuardia); pool size.

We consider pools of requests of varying size (i.e., 5 to 10 minutes) in which all new requests are included. For instance, if the pool size is 5 minutes, then all the new requests issued between "11:00:00" and "11:04:59" will be placed in the same pool.

It is worth noting that, since maximum delay tolerated is only a fraction of the shortest time to arrive at destination, then two or more requests can be checked if they can be combined regardless of the congestion fraction used. A proof for the simple case of the combination of two requests (which can be easily extended to more than two requests) follows.

If we consider two requests A and B originating in LaGuardia and for which request A is dropped off first, then it is necessary to check the maximum delay constraint on request B only:

$$t_{LaG,D(A)} + t_{D(A),D(B)} - t_{LaG,D(B)} \leq t_{LaG,D(B)} * \textit{max delay tolerated},$$

where D(A) (resp. D(B)) indicates drop off intersection for request A (resp. request B) and $t_{i,j}$ indicates the actual time (considering traffic) from intersection "i" to intersection "j".

Now, we can make the same congestion fraction cf (computed in some way) explicit to all times:

$$\frac{ft_{LaG,D(A)} + ft_{D(A),D(B)} - ft_{LaG,D(B)}}{cf} \leq \frac{ft_{LaG,D(B)} * max\ delay\ tolerated}{cf},$$

where $ft_{i,j}$ indicates the fastest time (i.e., considering maximum speed) from intersection "i" to intersection "j".

As we can notice, it is possible to eliminate the congestion fraction and the inequality will still hold even if only maximum speed is used.

4.3 Dynamic model settings

The full Manhattan road network is a directed graph which consists of 3,933 road intersections and 8,400 road segments; only the following road classes have been taken into account: primary, secondary, tertiary, residential, unclassified, road, living street. Other classes have been filtered out because they are rarely related to trips pickup and drop off ([4]). See Figure 10 to visualize the area of interest.

The aforementioned road network is also strongly connected: from each road intersection, it is possible to reach any other intersection. This is because we want to avoid scenarios in which vehicles are stuck in limited areas of the network or requests cannot be satisfied because there does not exist a path for vehicles to pick them up.

For this case study, only requests originating and terminating in Manhattan are considered. In particular, we have considered a full simulation on date January 25th 2013, from 10am to noon (enough to reach a steady state): it is a Friday, thus a weekday but also close to the weekend, in order to have a clearer idea of the trend. Moreover, in this time span, about 40,000 requests have been issued.

According to what reported in [16], in NYC the average number of passengers per request is

1.3: thus, we assume that each request involves one passenger only.

Requests which cannot be processed within maximum delay tolerated from the time they have been issued are then ignored and removed from the pool of requests to be processed.

In order to speed up the experiments, the shortest path and travel time between any two road intersections in Manhattan are pre-computed and stored in a look up table.

Vehicles are initialized as empty and with a random location at a specific road intersection. During the simulation, if it occurs that a vehicle v is not assigned to any request and does not have any passengers on board, then it is issued a schedule from its current road intersection to a random road intersection.

Considering a trip T which can be assigned to a vehicle v, "total distance saved" for the trip is given by:

$$\textit{Total distance saved} = miles_{ir} - (\sum_{r \in T} miles_{sp,r}) - \arg\min_{j}(miles_{nr,j}),$$

where $miles_{ir}$ indicates the miles for initial route of vehicle v (due to on board requests), $miles_{sp,r}$ indicates the shortest path miles for request r, $miles_{nr,j}$ indicates the miles for new route combination j: if there are multiple feasible combinations which satisfy all the requests involved, then the one which provides the shortest updated route (and, thus, the largest total distance saved) is chosen.

Notice that, for the initial and the updated routes, we do not consider miles traveled when vehicle is empty because those miles would have been traveled in any case to pick up a new request.

For the experiments, we have used a congestion fraction of 0.7, which means that there is a little less than medium traffic in Manhattan. Moreover, we have analyzed pools of 30 seconds because, in Manhattan, on average there about 167 requests each 30 seconds (in the two hours interval considered) and, thus, it is a considerable number.

CHAPTER 5

RESULTS ANALYSIS

After the optimum and fair plans are computed, we can issue them both separately to analyze the differences between them in terms of distance traveled (that is equivalent to total savings), assuming that no compensation scheme is applied to the optimum plan.

Following the structure of chapter 3, we will analyze the static model case first (both for combining at most two requests at a time and for combining more than two requests at a time) and then the dynamic model.

5.1 Static model: combination of at most two requests originating in single hub

5.1.1 Comparison between optimum and evenly split fair plans

The comparison in % mileage reduction between the optimum plan and the evenly split fair plan is investigated with respect to the variation of willingness to ride share, maximum delay tolerated and pool size (in minutes). In the end, ridesharing graph samples for the two plans are provided to highlight the differences.

- **Willingness to ride share**

  In Figure 15, it is possible to visualize the % mileage reduction with the optimum plan with pool size equals to 5 minutes and maximum delay tolerated equals to 10%, studying the results to the variation of willingness to ride share. We can notice that % mileage reduction increases almost linearly as willingness to ride share increases.

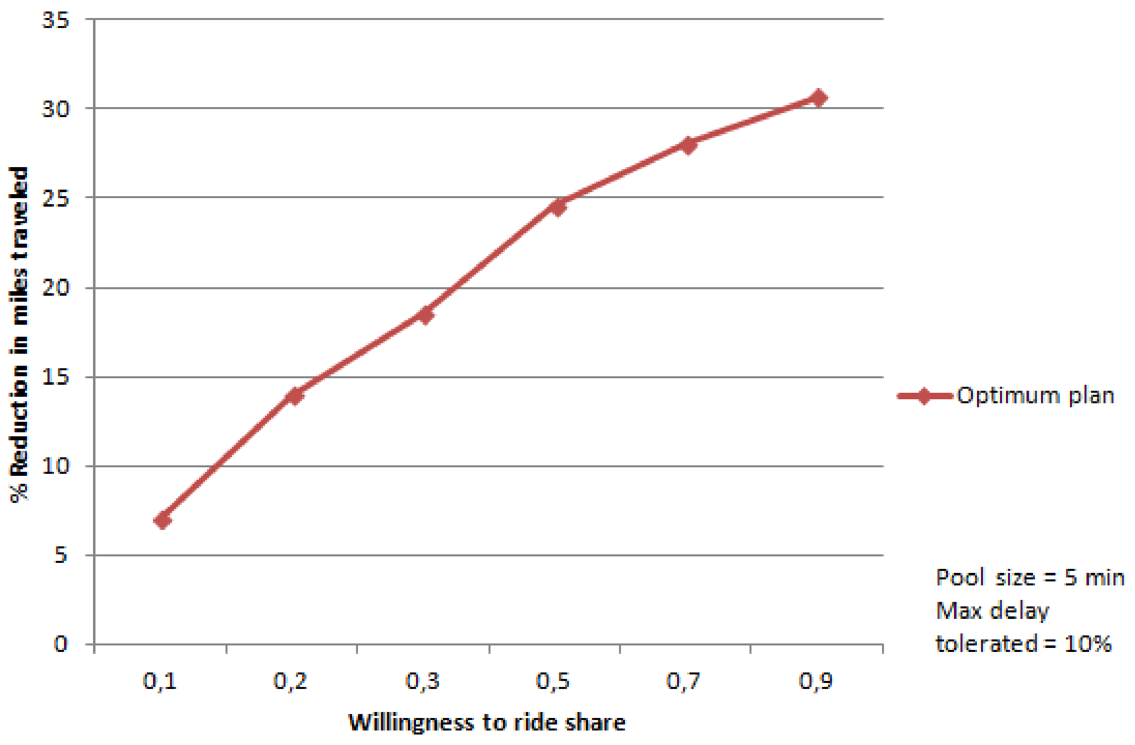

Figure 15: Static model: % mileage reduction with the optimum plan (willingness to ride share)

Figure 16 provides a direct comparison between the optimum and the evenly split fair plans with respect to willingness to ride share: it presents how often the increase in terms of % mileage reduction of the optimum plan with respect to the evenly split fair plan is less than 5%, between 5% and 15%, more than 15%. There may also be no increase in the two plans, thus we distinguish the existence of a solution that is exactly the same ("No Increase" in the plot) and the non-existence of a solution ("Not applicable" in the plot), which means that the analyzed ridesharing graph had an empty set of edges (i.e., no request in the pool could have been merged).

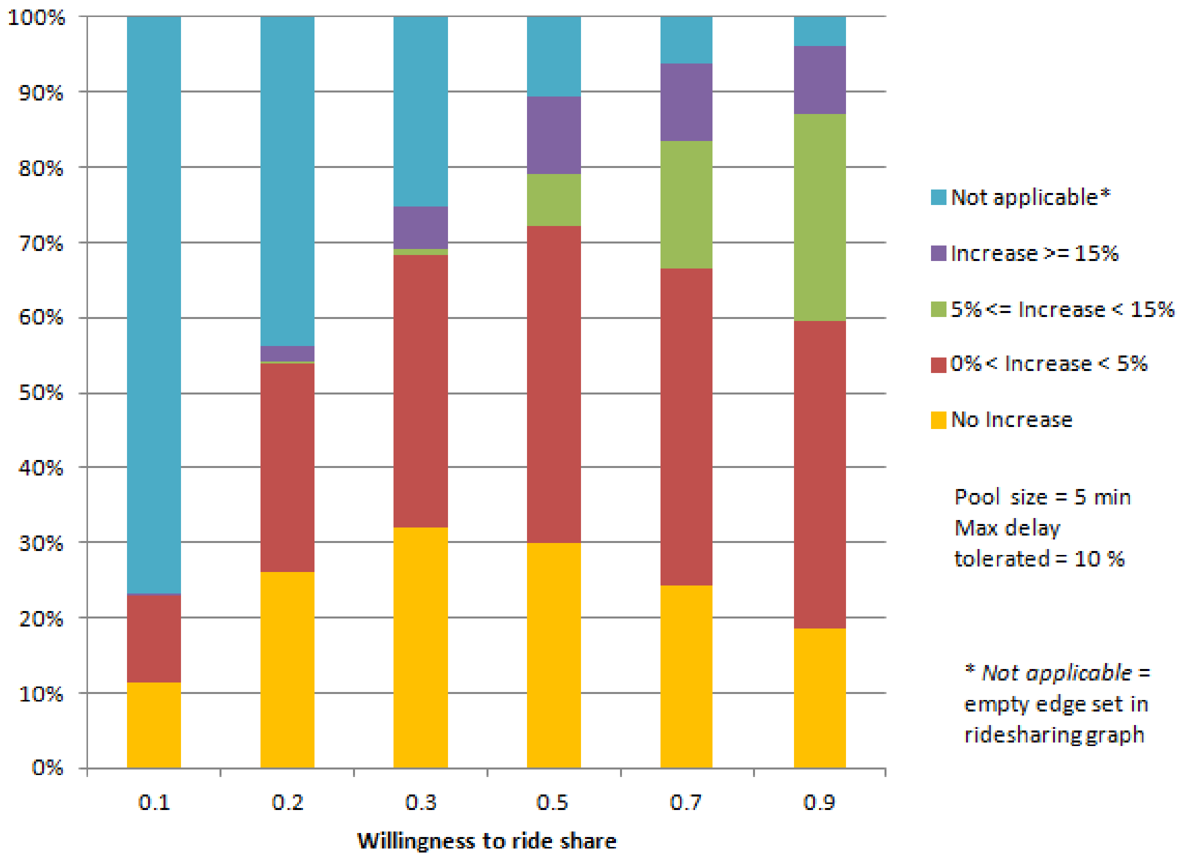

Figure 16: Static model: optimum plan % increase over the fair plan (willingness to ride share)

- **Maximum delay tolerated**

  In Figure 17, it is possible to visualize the % mileage reduction with the optimum plan with pool size equals to 5 minutes and willingness to ride share equals to 90%, studying the results to the variation of maximum delay tolerated. Also in this case, as for willingness to ride share variation, % mileage reduction grows almost linearly when maximum delay tolerated increases.

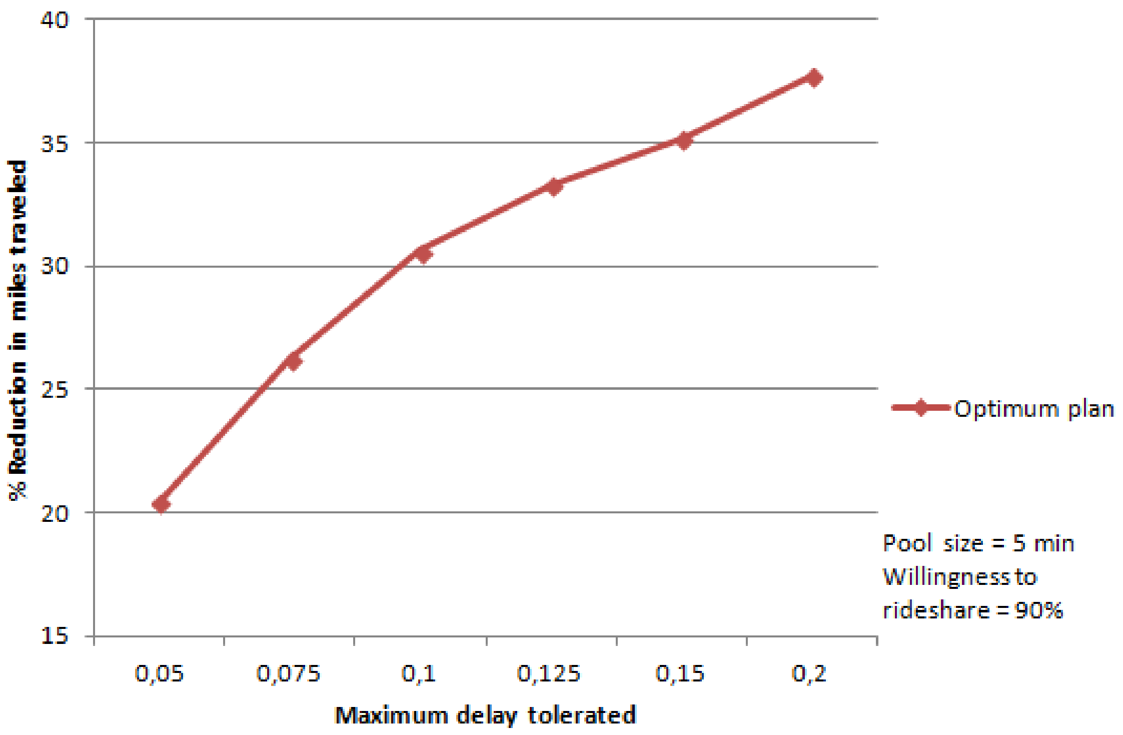

Figure 17: Static model: % mileage reduction with the optimum plan (maximum delay tolerated)

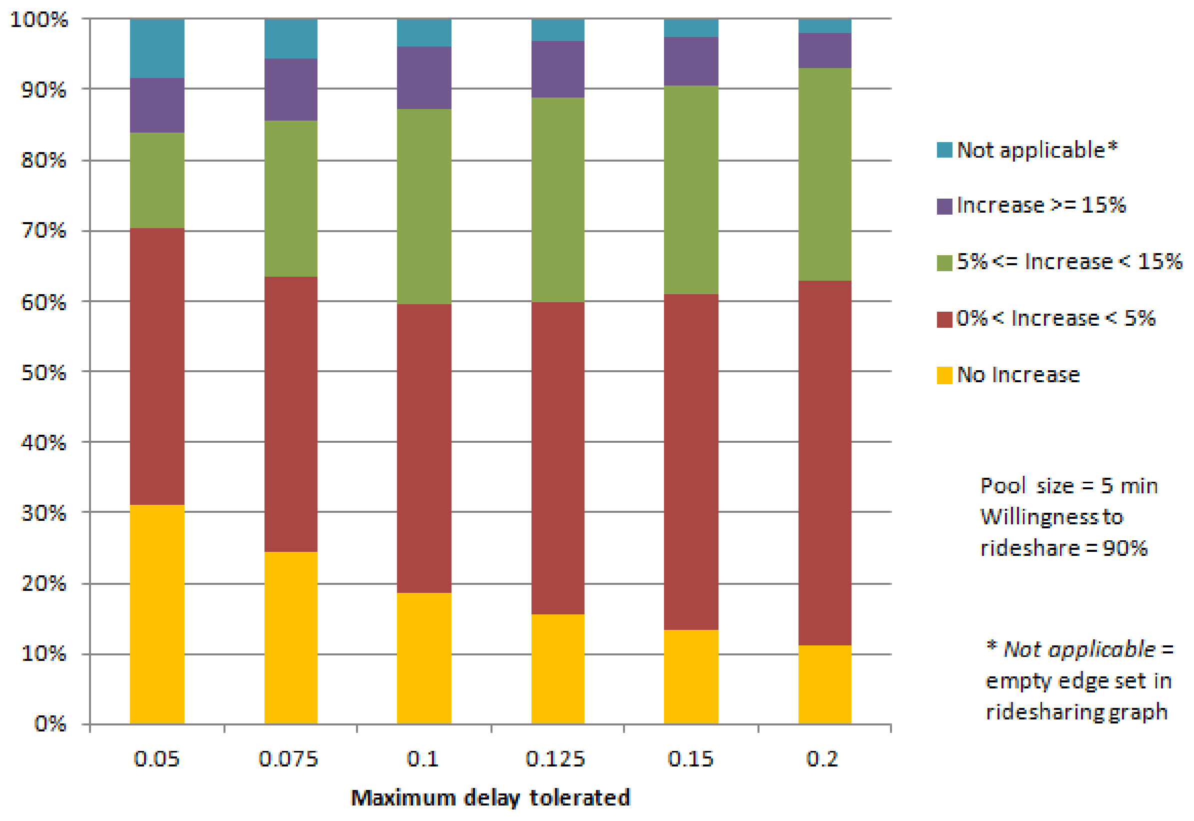

Figure 18: Static model: optimum plan % increase over the fair plan (maximum delay tolerated)

Equivalently to what shown by Figure 16 for willingness to ride share, Figure 18 provides a comparison between the optimum and the evenly split fair plans with respect to maximum delay tolerated: it presents how often the increase in terms of % mileage reduction of the optimum plan with respect to the evenly split fair plan is less than 5%, between 5% and 15%, more than 15%. There may also be no increase in the two plans, thus we distinguish the existence of a solution that is exactly the same ("No Increase" in the plot) and the non-existence of a solution ("Not applicable" in the plot), which means that the analyzed ridesharing graph had an empty set of edges (no request in the pool could have been merged).

- **Pool size**

  In Figure 19, we visualize the % mileage reduction in the optimum plan compared to no-ridesharing, as a function of the pool size; willingness to ride share equals to 90% and maximum delay tolerated equals to 10%.

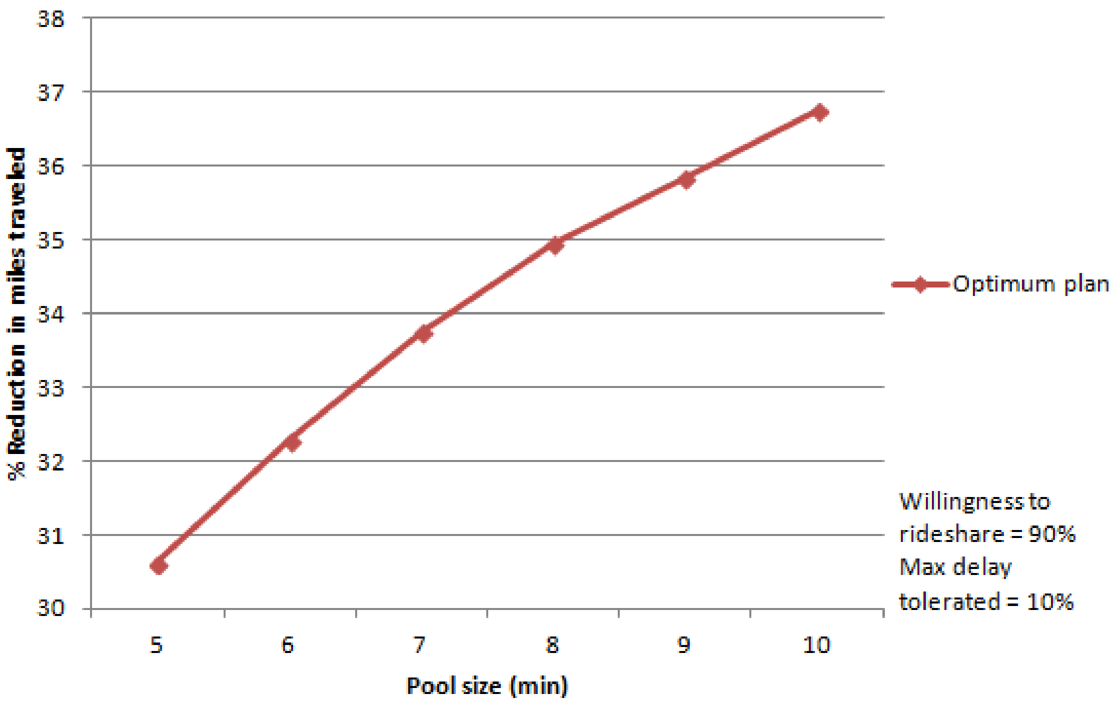

Figure 19: Static model: % mileage reduction with the optimum plan (pool size)

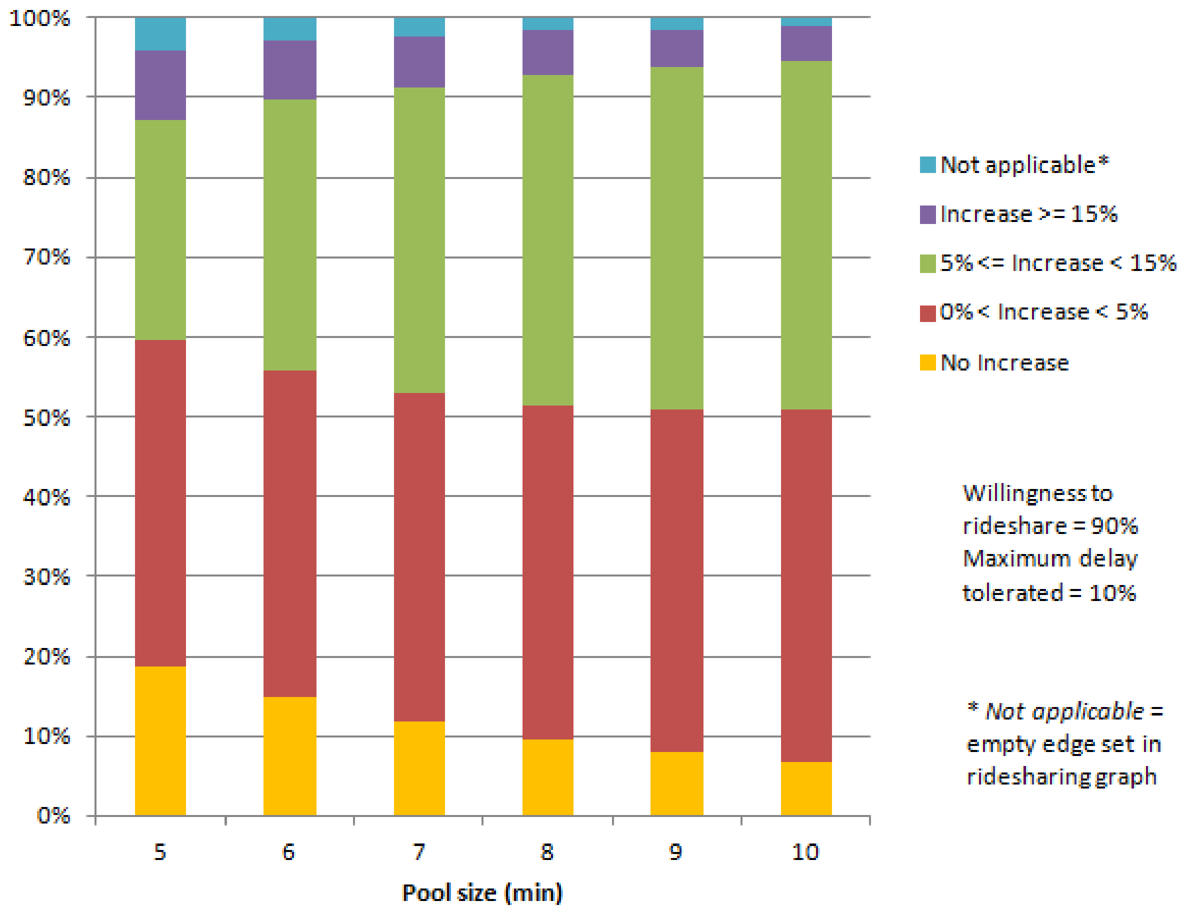

Figure 20: Static model: optimum plan % increase over the fair plan (pool size)

Figure 20 provides a comparison between the optimum and the evenly split fair plans with respect to pool size: it presents how often the increase in terms of % mileage reduction of the optimum plan with respect to the evenly split fair plan is less than 5%, between 5% and 15%, more than 15%. There may also be no increase in the two plans, thus we distinguish the existence of a solution that is exactly the same ("No Increase" in the plot) and the non-existence of a solution ("Not applicable" in the plot), which means that the analyzed ridesharing graph had an empty set of edges (no request in the pool could have been merged).

- **Ride sharing graphs**

  We illustrate, for some ridesharing graphs (RSGs), the maximum weight matching and the (evenly split) fair matching.

  The settings for the RSGs are: willingness to rideshare = 90%; maximum delay tolerated = 10%; pool size = 5 minutes.

  In a visual representation of an RSG: nodes are identified by request IDs; edges are labeled with a weight which represents the distance saved if the two related requests were merged; edges which belong to the solution provided by a specific algorithm are colored in red.

  In the following pages, we show two cases:

  1. The optimum and the evenly split fair plans provide the same solution;
  2. The optimum plan provides a greater % reduction with respect to the evenly split fair plan.

1st *case: the optimum and the (evenly split) fair plans are the same. In this case, the RSG is disconnected and consists of 4 components.*

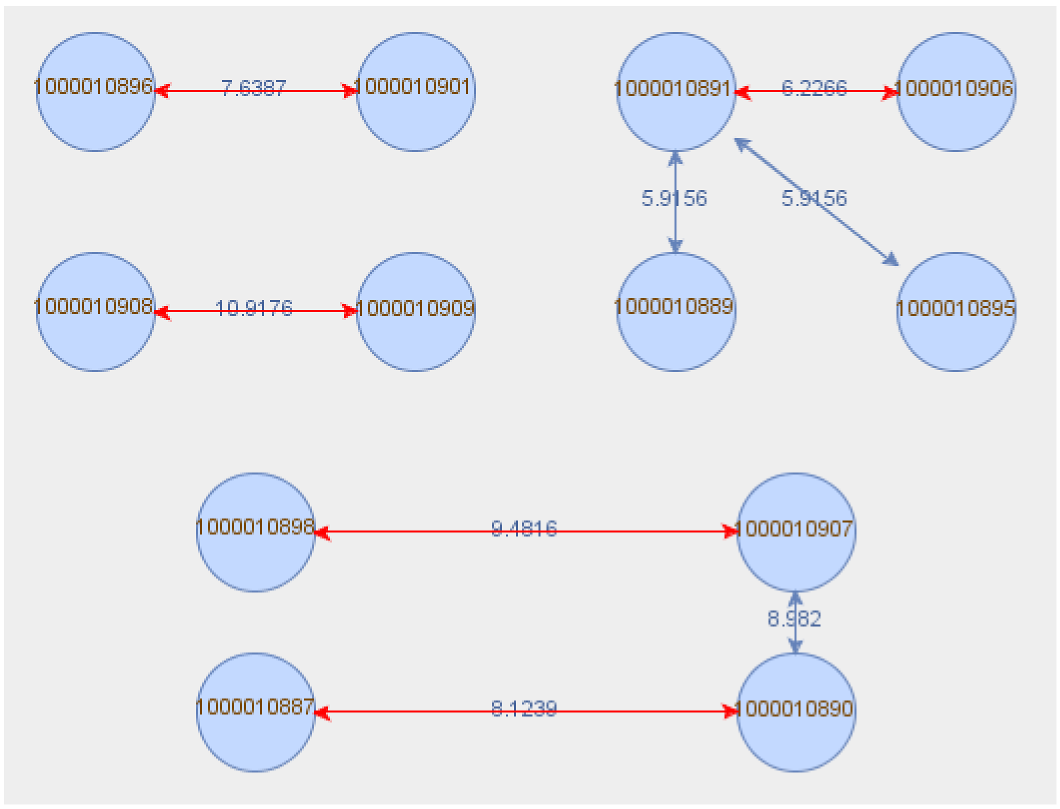

Figure 21: Static model: optimum and evenly split fair plans provide the same solution

2^{nd} case: the optimum plan provides a solution with higher distance saved (62.3522) than the evenly split fair plan does (54.7702). Their respective % mileage reductions are 31.1% and 28.05%. In this case, the RSG is disconnected and there are two components.

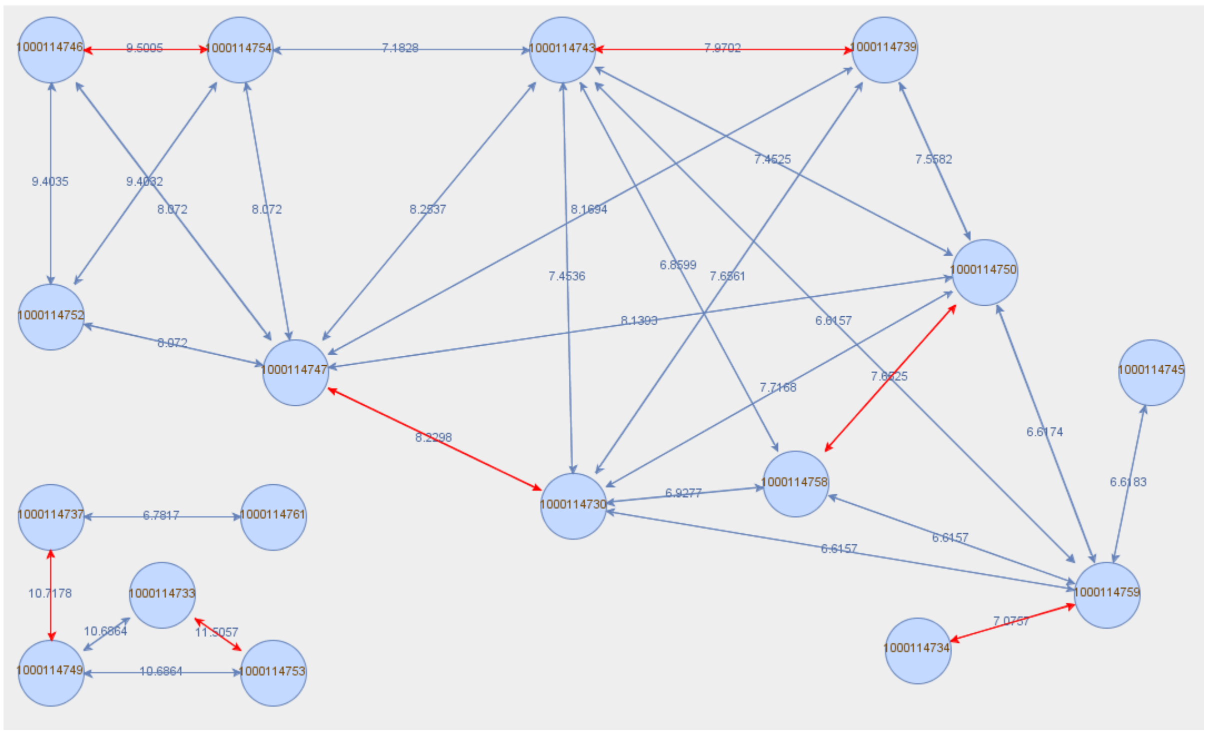

Figure 22: Static model: optimum fair plan provides greater % reduction than the evenly split fair plan

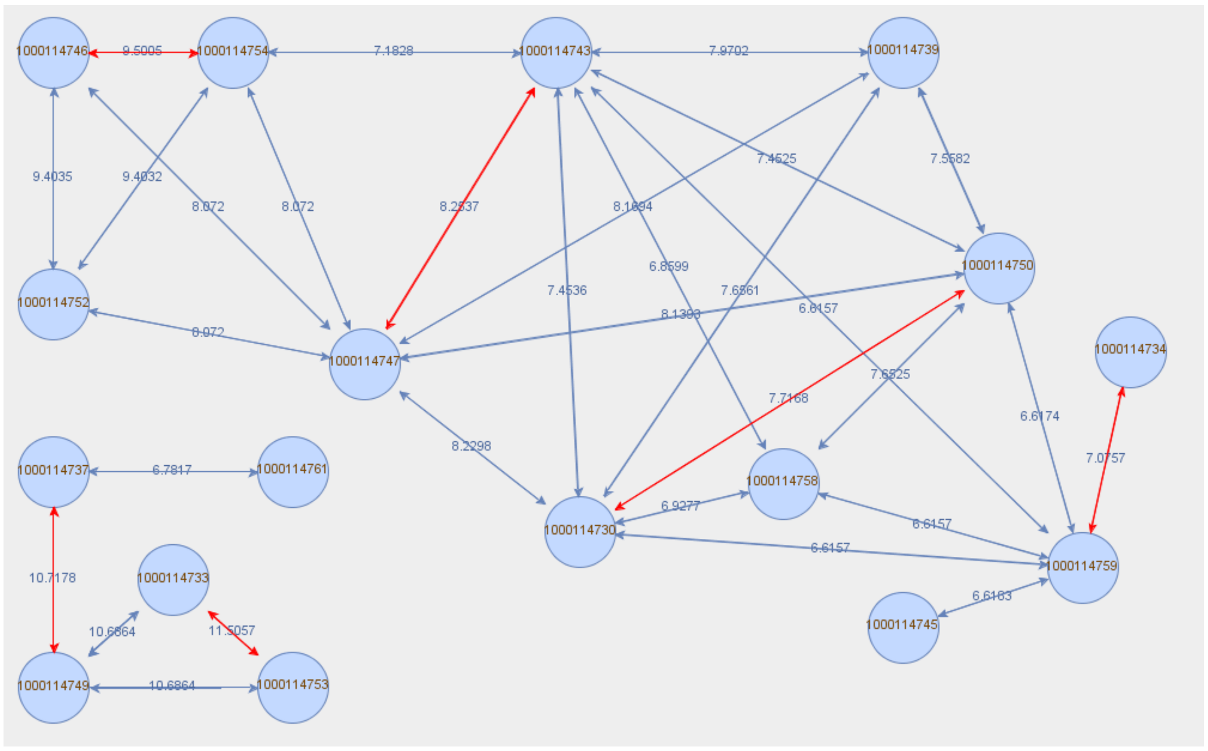

Figure 23: Static model: evenly split fair plan provides smaller % reduction than the optimum plan

5.1.2 Comparison between evenly and unevenly split fair plans

After comparing the optimum and evenly split fair plans, we have conducted other experiments aimed at comparing the evenly and unevenly split fair matching algorithms.

The difference between the unevenly and evenly split fair plans is calculated by considering the amount of money (or, in our case, miles traveled effectively) each request will have to pay in the two cases: a positive value means that each request will pay, in total, less with the evenly split fair plan; a negative value means that each request will pay, in total, less with the unevenly split fair plan.

But in order to draw significant conclusions on the difference in payment scheme between the two plans, it is more appropriate to analyze what fraction of total miles traveled without ride sharing this difference constitutes.

Thus, this is the formula for each analyzed pool:

$$\%\ \textit{difference} = \frac{miles_{usfp} - miles_{esfp}}{miles_{nr}} * 100\%,$$

where $miles_{usfp}$ indicates the miles traveled with the unevenly split fair plan, $miles_{esfp}$ indicates the miles traveled with the evenly split fair plan and $miles_{nr}$ indicates the miles traveled without ride sharing.

% difference between the evenly split and unevenly split fair plans is investigated with respect to the variation of willingness to ride share, maximum delay tolerated and pool size (in minutes). Also, since the unevenly split fair plan may not exist, the trend of how often this occurs is presented. In the end, ridesharing graph samples for the two plans are provided to highlight the differences.

- **Willingness to ride share**

  In Figure 24 the comparison between evenly and unevenly split fair plans, with respect to willingness to ride share, is provided: % average difference and standard deviation are presented.

  According to what standard deviation in Figure 24 outlines, % difference can be at most at around 1.6%: this is negligible and supports the idea that, for experiments in section

5.2, it is worth comparing the optimum plan to the evenly split fair plan only (which is tractable with respect to the unevenly split fair plan computation).

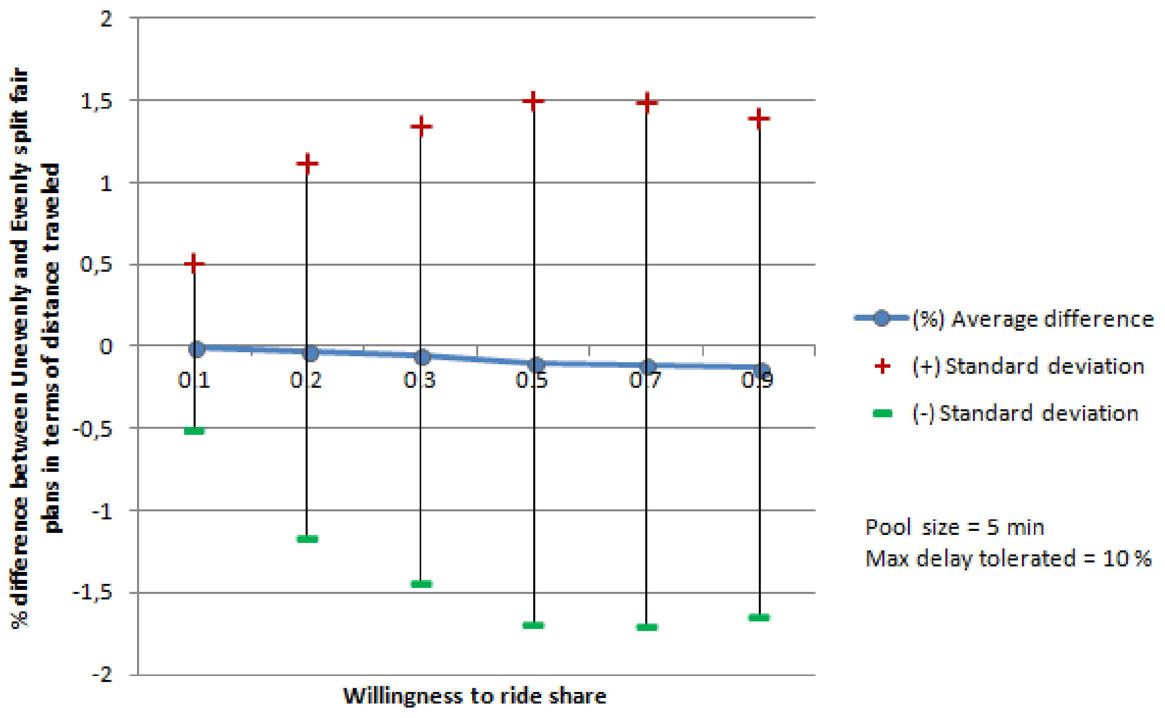

Figure 24: Static model: comparison between unevenly and evenly split fair plans (willingness to ride share)

Since a solution to the SRP (used to compute the unevenly split fair plan) may not always be retrieved, the % of this case occurrences is presented with respect to willingness to ride share (Figure 25).

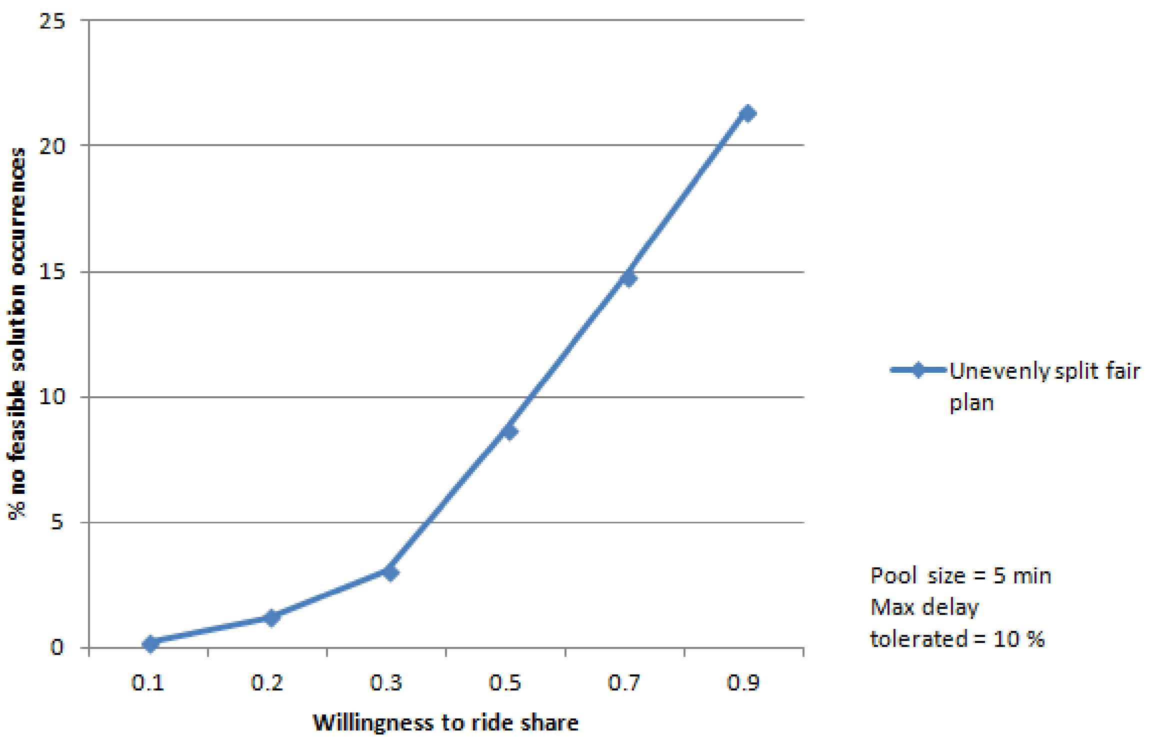

Figure 25: Static model: % no solution to the SRP (willingness to ride share)

As we can observe, % of no solution to the SRP increases exponentially as willingness to rideshare increases.

- **Maximum delay tolerated**

  In Figure 26 the comparison between evenly and unevenly split fair plans, with respect to maximum delay tolerated, is provided: % average difference and standard deviation are presented. According to what standard deviation in Figure 26 outlines, % difference can be at most at around 1.8%: this is negligible and supports the idea that, for experiments in section 5.2, it is worth comparing the optimum plan to the evenly split fair plan only (which is tractable with respect to the unevenly split fair plan computation).

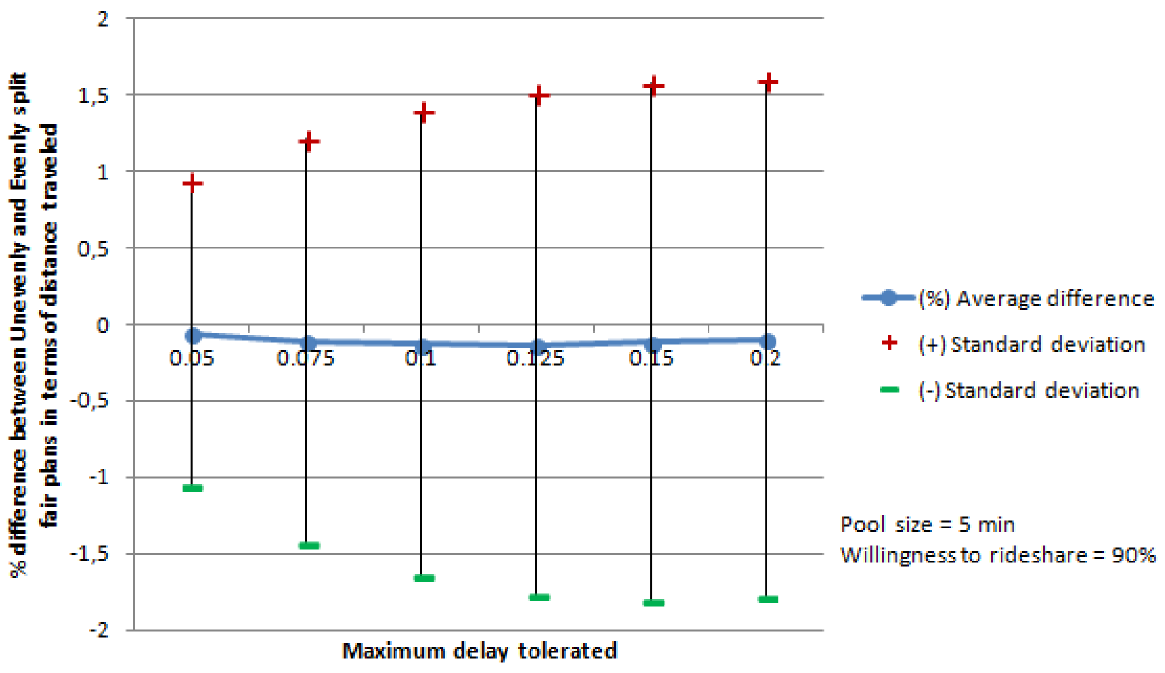

Figure 26: Static model: comparison between unevenly and evenly split fair plans (maximum delay tolerated)

Since a solution to the SRP (used to compute the unevenly split fair plan) may not always be retrieved, the % of this case occurrences is presented with respect to maximum delay tolerated (Figure 27).

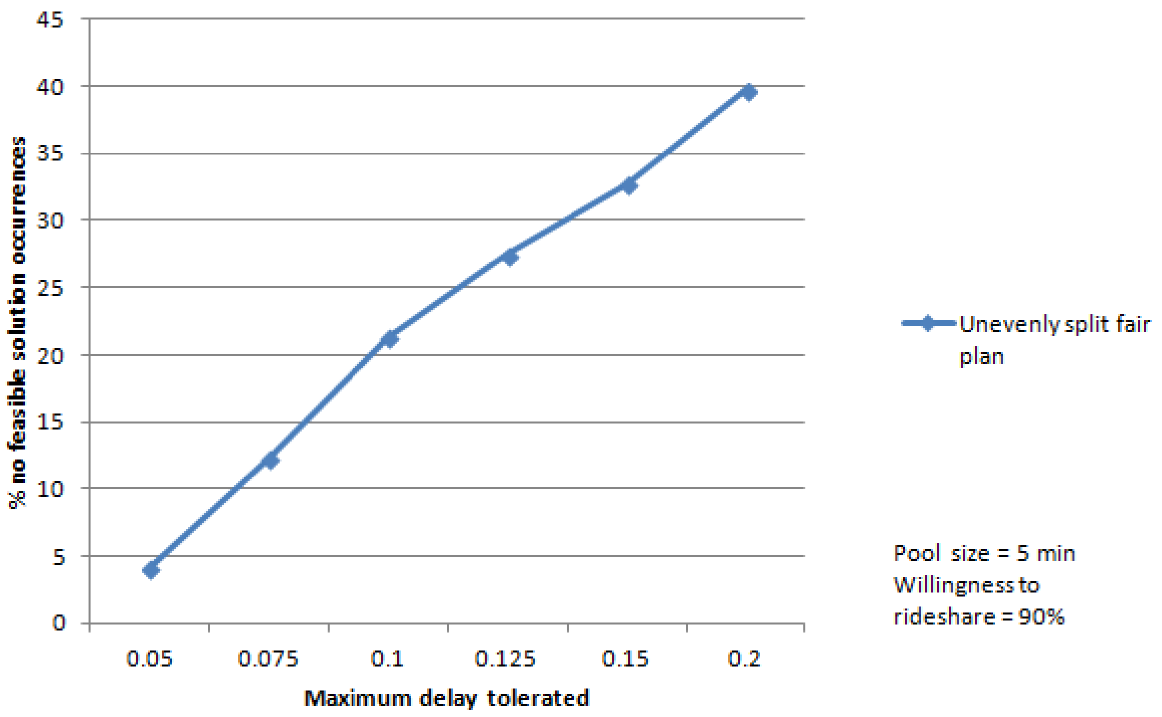

Figure 27: Static model: % no solution to the SRP (maximum delay tolerated)

As we can observe, % of no solution to the SRP increases almost linearly as maximum delay tolerated increases.

- **Pool size**

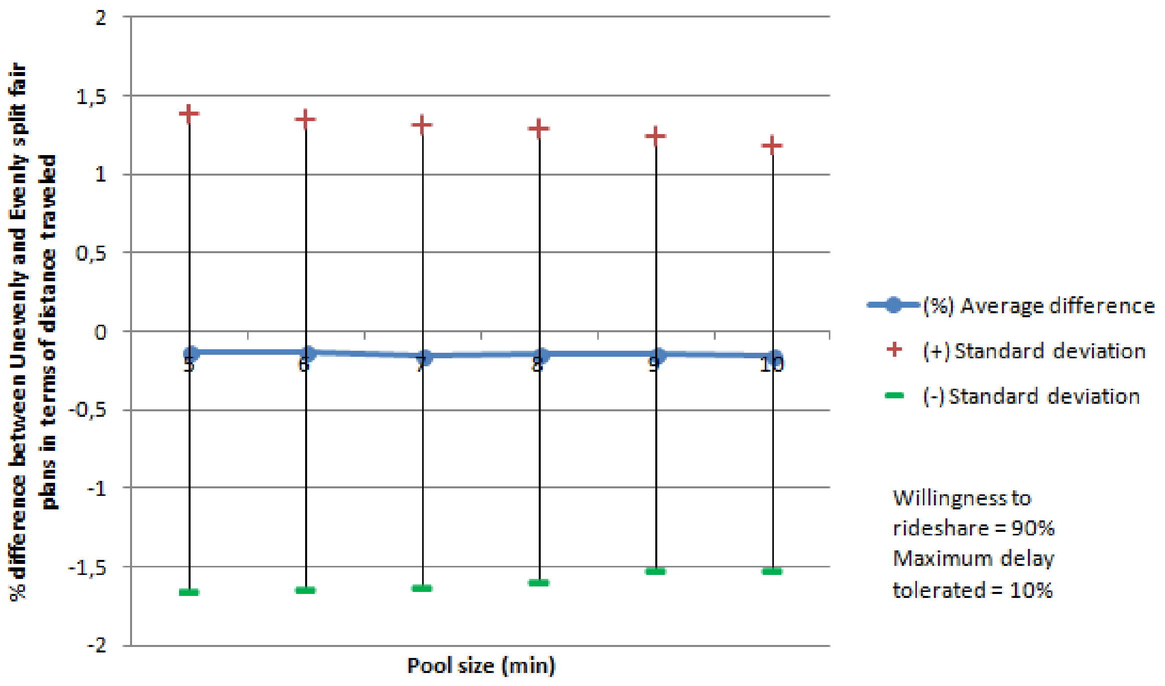

Figure 28: Static model: comparison between unevenly and evenly split fair plans (pool size)

In Figure 28 the comparison between evenly and unevenly split fair plans, with respect to pool size, is provided: % average difference and standard deviation are presented.

According to what standard deviation in Figure 28 outlines, % difference can be at most at around 1.6%: this is negligible and, along with the equivalent plots with respect to willingness to ride share and maximum delay tolerated, supports the idea that, for experiments in section 5.2, it is worth comparing the optimum plan to the evenly split fair plan only (which is tractable with respect to the unevenly split fair plan computation).

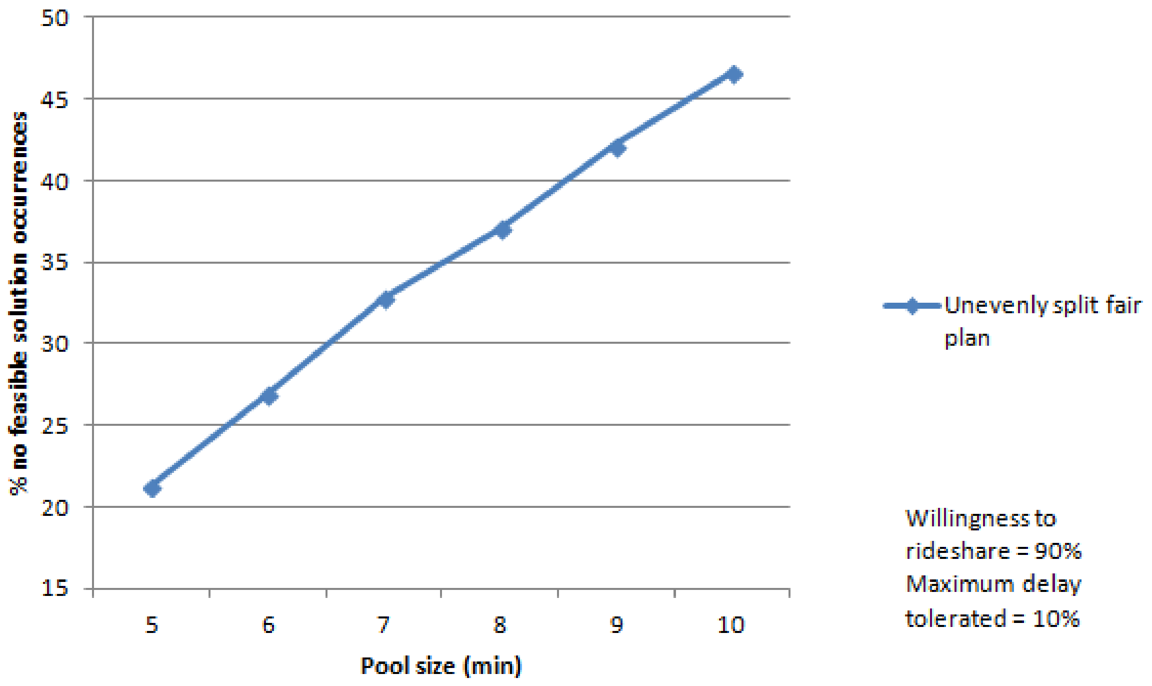

Figure 29: Static model: % no solution to the SRP (pool size)

Since a solution to the SRP (used to compute the unevenly split fair plan) may not always be retrieved, the % of this case occurrences is presented with respect to pool size (Figure 29).

As we can observe, % of no solution to the SRP increases almost linearly as pool size increases.

- **Ride sharing graphs**

  We illustrate, for some ridesharing graphs (RSGs), the evenly and unevenly split fair matching. The settings are: willingness to rideshare = 90%; maximum delay tolerated = 10%; pool size = 5 minutes.

  In a RSG: nodes are identified by request IDs; edges which belong to the solution provided by a specific algorithm are colored in red.

  For the evenly split fair RSG, edges are labeled with a weight which represents the distance saved if the two related requests were merged. For the unevenly split fair RSG, directed edges are used only for clarity purposes. An edge going from node A to node B is labeled with "X-Y" which indicates that "X" is the distance saved by request A if the edge were taken and that "Y" is the distance saved by request B if the edge were taken. It is worth noting that the sum between "X" and "Y" corresponds to the weight assigned to the same edge in the evenly split fair RSG.

1st *case: the evenly split fair plan provides a solution with higher distance saved (38.2609) than the unevenly split fair plan does (30.1579). Their respective % mileage reductions are 30.197% and 23.803%. In this case, the RSG is disconnected and there are two components.*

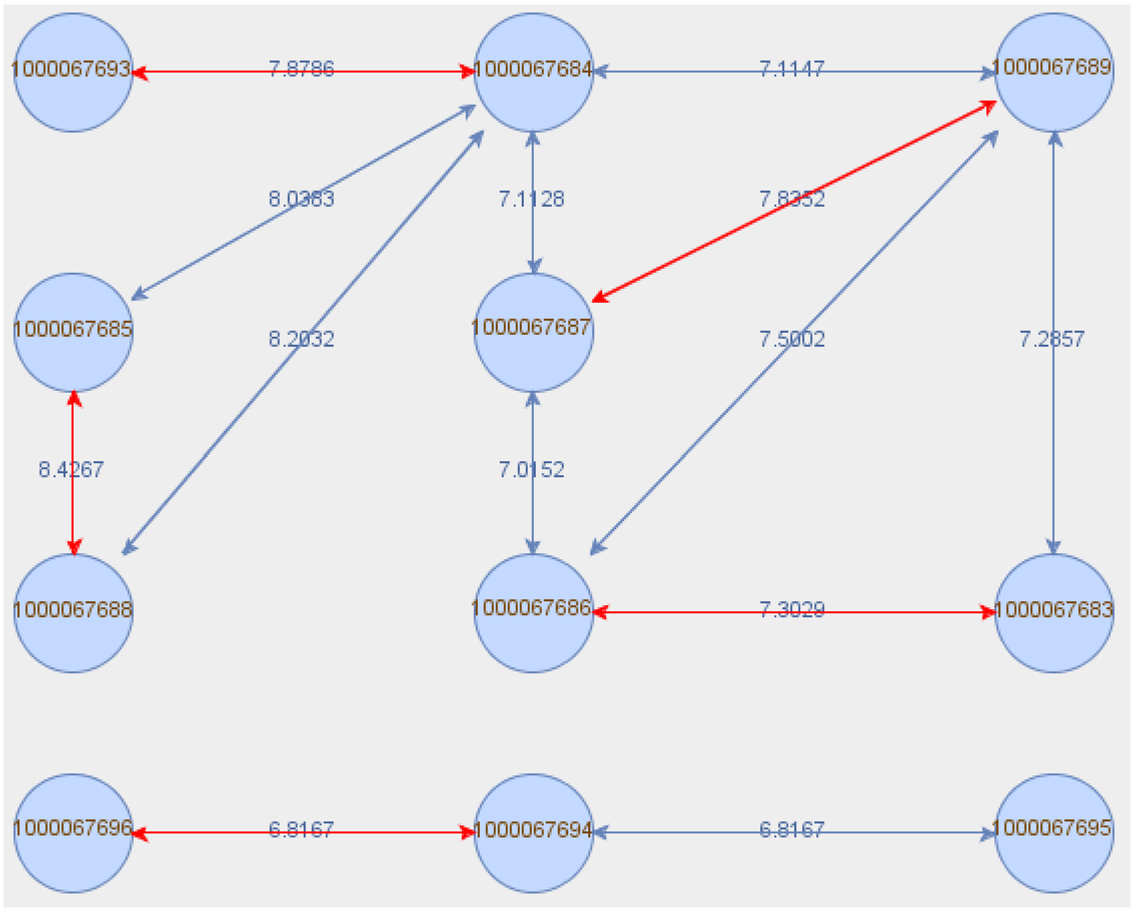

Figure 30: Static model: evenly split fair plan provides greater % reduction than the unevenly split fair plan

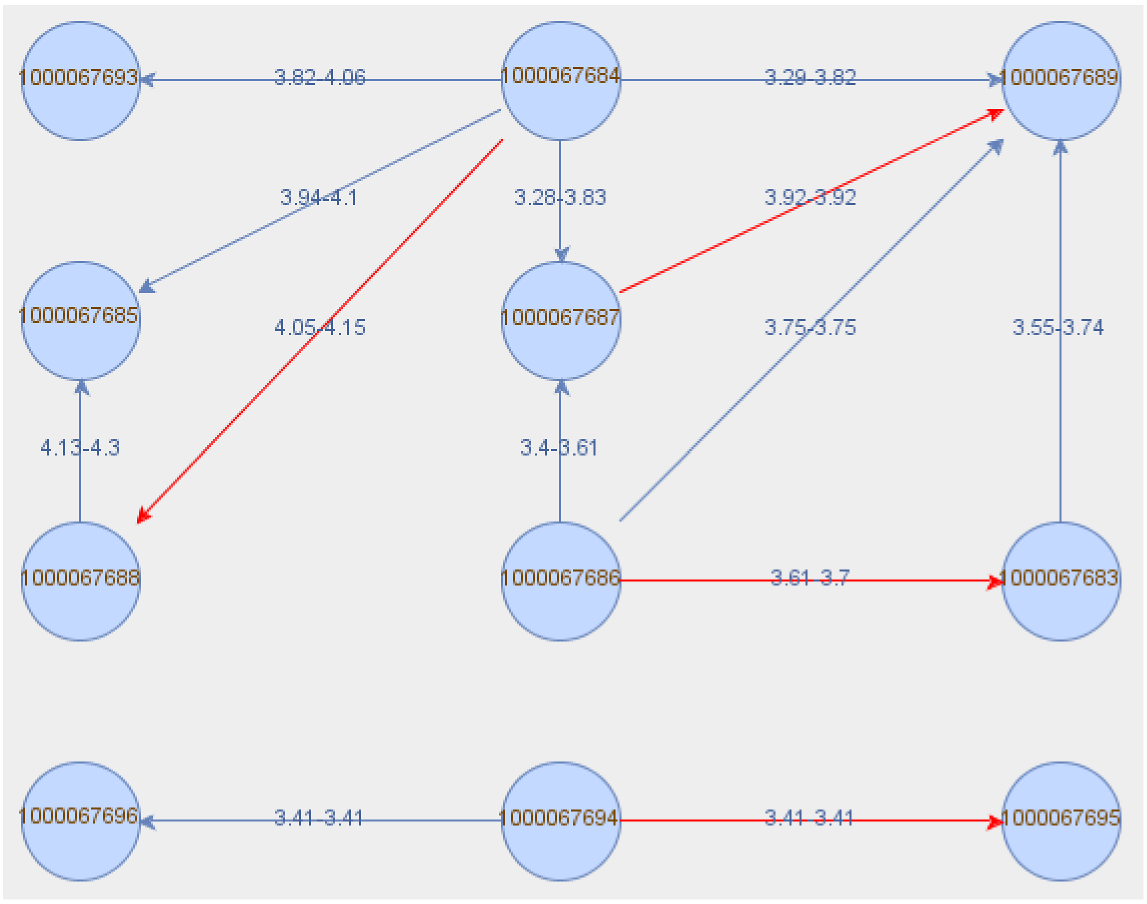

Figure 31: Static model: unevenly split fair plan provides smaller % reduction than the evenly split fair plan

2nd case: the unevenly split fair plan provides a solution with higher distance saved (35.614) than the evenly split fair plan does (28.7601). Their respective % mileage reductions are 22.196% and 17.925%. In this case, the RSG is disconnected and there are two components.

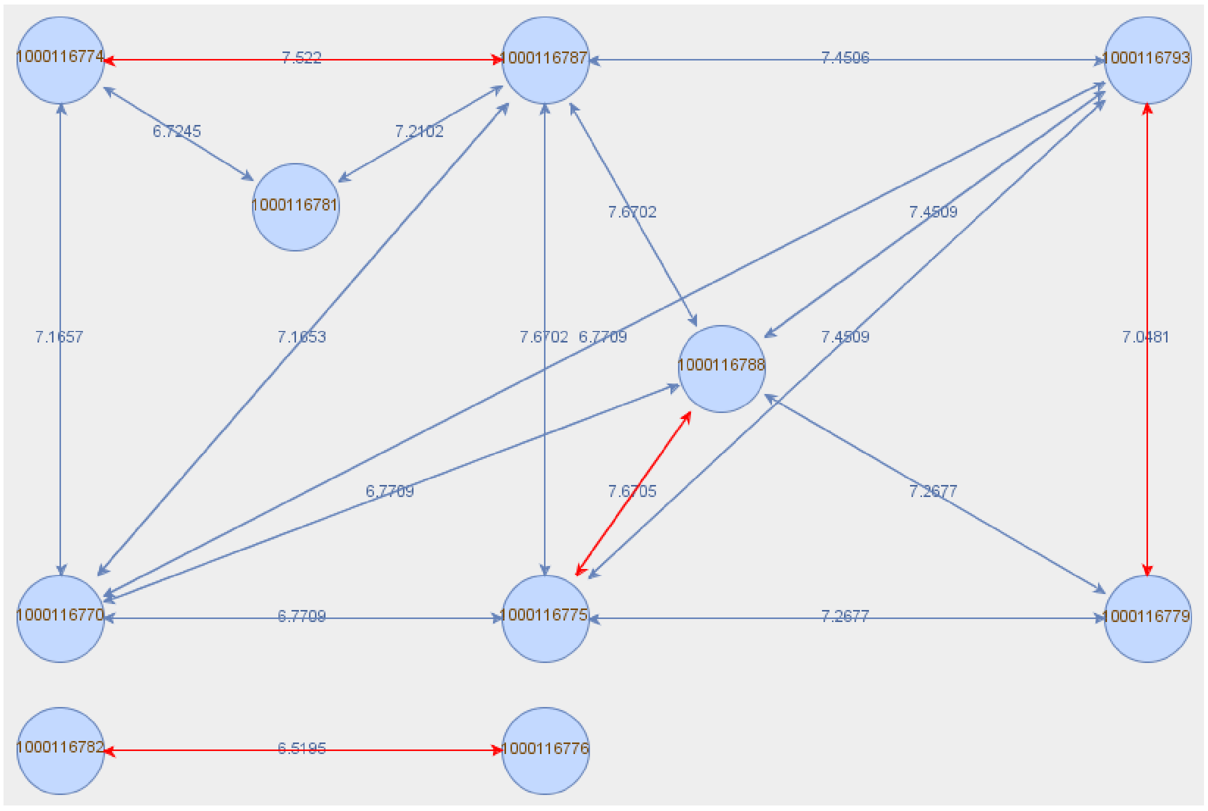

Figure 32: Static model: evenly split fair plan provides smaller % reduction than the unevenly split fair plan

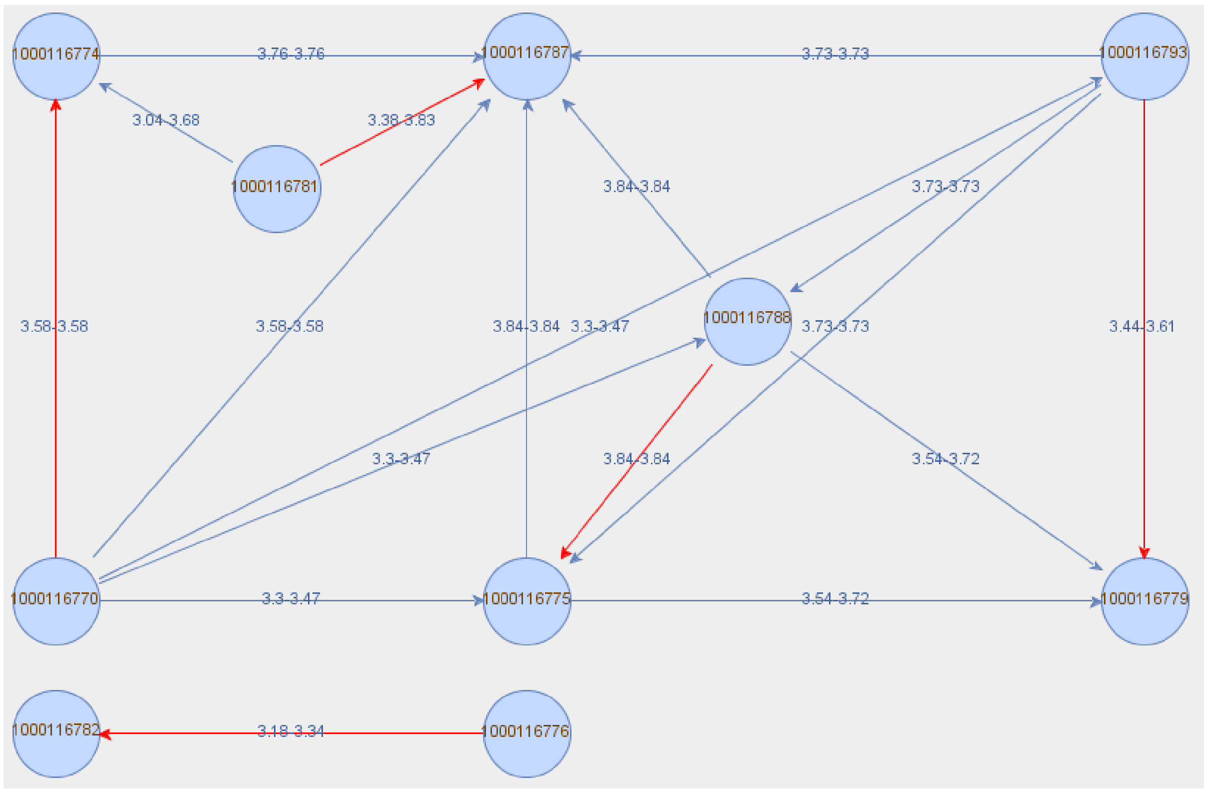

Figure 33: Static model: unevenly split fair plan provides greater % reduction than the evenly split fair plan

In the examples provided, we can notice that the uneven splits of the distance saved for a shared trip are almost even. This implies that the assumption of split evenly the distance saved among the involved requests is effective, at least within this context. As a result, it is reasonable to conduct experiments for combining more than two requests with respect to the evenly split fair plan only (section 5.2).

5.2 Static model: combination of "k" requests originating in single hub

So far, we have presented the results regarding the comparison between the optimum and (evenly and unevenly split) fair plans, but in the case where only two requests may be merged at a time.

Thus, in this section we analyze the case in which more than two requests can be processed by the same taxi, within the static model context, considering the variation of maximum number of passengers allowed on board.

It is worth noting that if a taxi can have 4 passengers on board, this does not mean that it is possible to combine any four requests: each request is related to a particular number of passengers, thus a taxi can process at the same time requests whose overall number of passengers is 4 (and it is four requests in the worst case).

In Figure 34, we visualize the % mileage reduction in the optimum plan compared to no-ridesharing, as a function of the maximum number of passengers; willingness to ride share equals to 90%, maximum delay tolerated equals to 10% and pools of 5 minutes are considered.

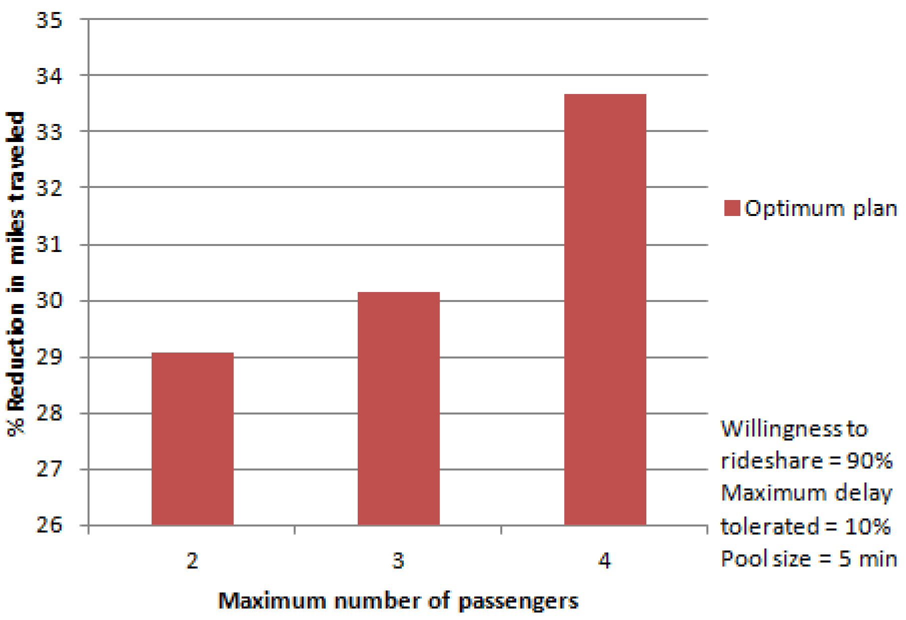

Figure 34: Static model (combine more than two requests): % mileage reduction with the optimum plan

Figure 35 compares the optimum and the evenly split fair plans with respect to maximum number of passengers: it presents how often the increase in terms of % mileage reduction of the optimum plan with respect to the evenly split fair plan is less than 5%, between 5% and 15%, more than 15%. There may also be no increase in the two plans, thus we distinguish the existence of a solution that is exactly the same ("No Increase" in the plot) and the non-existence of a solution ("Not applicable" in the plot), which means that the analyzed ridesharing graph had an empty set of edges (no request in the pool could have been merged).

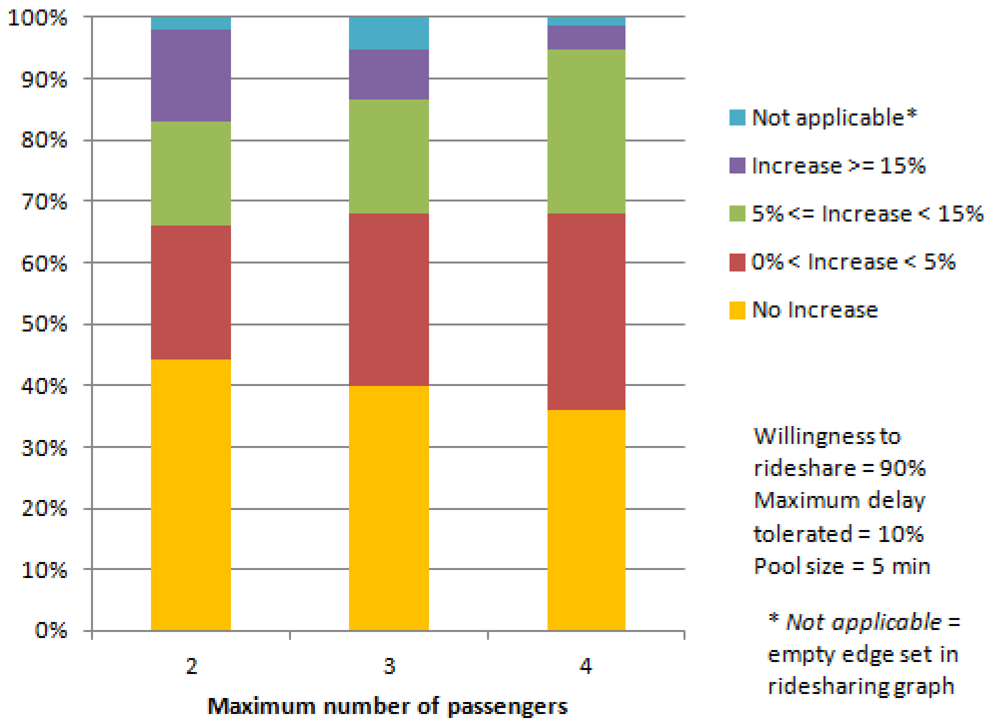

Figure 35: Static model (combine more than two requests): optimum plan % increase over the fair plan

It is interesting to compare the results shown in Figure 34 and Figure 35 to those related to combining at most two requests at a time (Figure 15 and Figure 16), with the following settings: maximum number of passengers is 4; willingness to ride share is 90%; maximum delay tolerated is 10%; pool size is 5 minutes.

We can notice that there is no clear difference between merging at most two requests at a time and merging as many requests a taxi can process: important benefits for the static model can be reached even with the option of merging at most two requests.

5.3 Dynamic model: requests assignment to already existing rides

5.3.1 Comparison between optimum and evenly split fair plans

In this section we analyze the benefits of ridesharing within the dynamic model (in which requests may be assigned to a taxi with some passengers on board) with respect to total savings. We have considered the variation of maximum delay tolerated; constants are the following: willingness to rideshare is 90%; pool size is 30 seconds; overall number of taxis is 5,000.

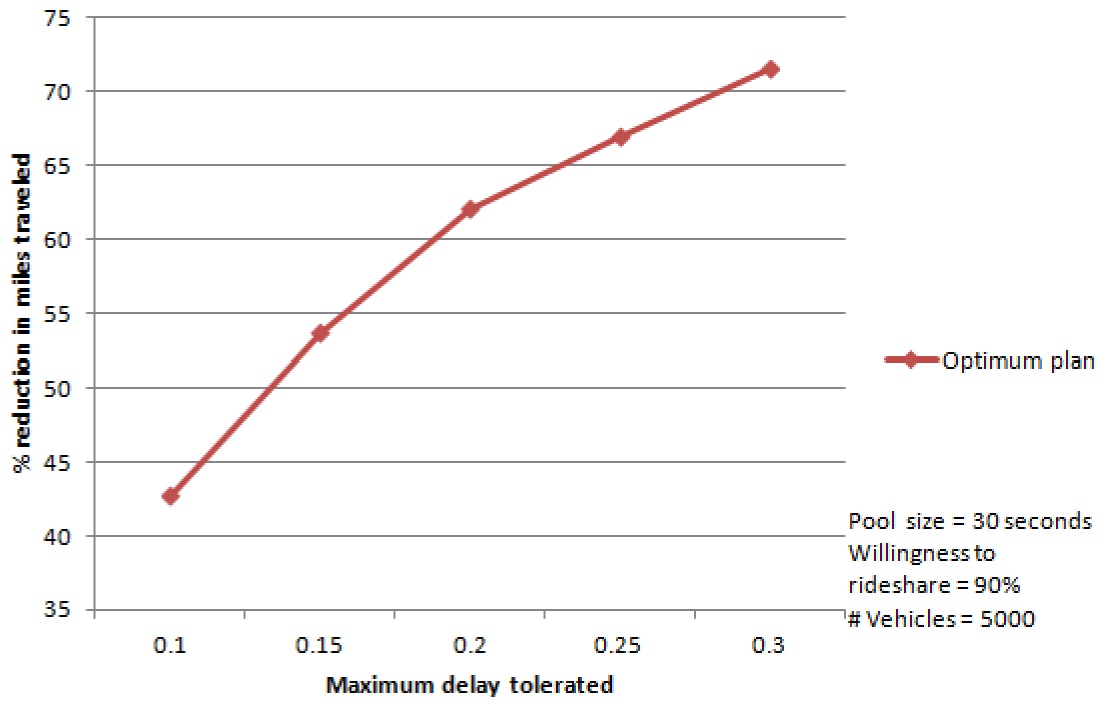

Figure 36: Dynamic model: % mileage reduction with the optimum plan (maximum delay tolerated)

In Figure 36, we can notice that, as maximum delay tolerated increases, then more opportunities for ridesharing are possible and, as a result, more miles (and therefore dollars) are saved and with huge benefits with respect to the static model. In fact, in Figure 17 we can see that, for instance, with maximum delay tolerated set to 20%, we have that % mileage reduction is about 38%, while in the dynamic model (Figure 36) it is about 62%.

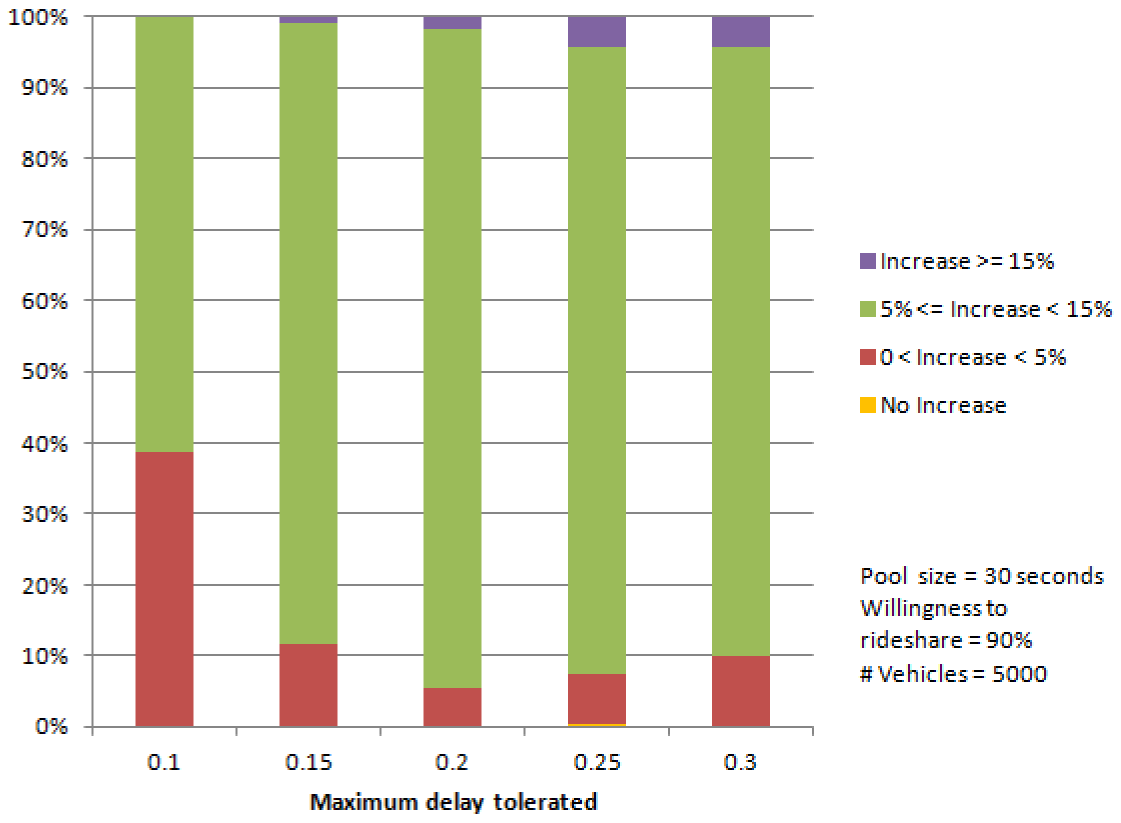

Figure 37: Dynamic model: optimum plan % increase over the fair plan (maximum delay tolerated)

Figure 37 provides a direct comparison between the optimum and the evenly split fair plans: it presents how often the increase in terms of % mileage reduction of the optimum plan with respect to the evenly split fair plan is less than 5%, between 5% and 15%, more than 15%, or there is no increase (i.e., they both provide the same solution).

As we can notice, the evenly split fair plan almost never provides the same solution as the optimum plan does: in particular, the latter often increases % mileage reduction between 5%-15% with respect to the former.
It is worth noting that, even if increasing maximum delay tolerated improves opportunities for ridesharing, this results into longer trips and more requests being processed at the same time: thus, new requests may not find a close vehicle to process them as well. In Figure 38 we show this trend by considering the variation of maximum delay tolerated.

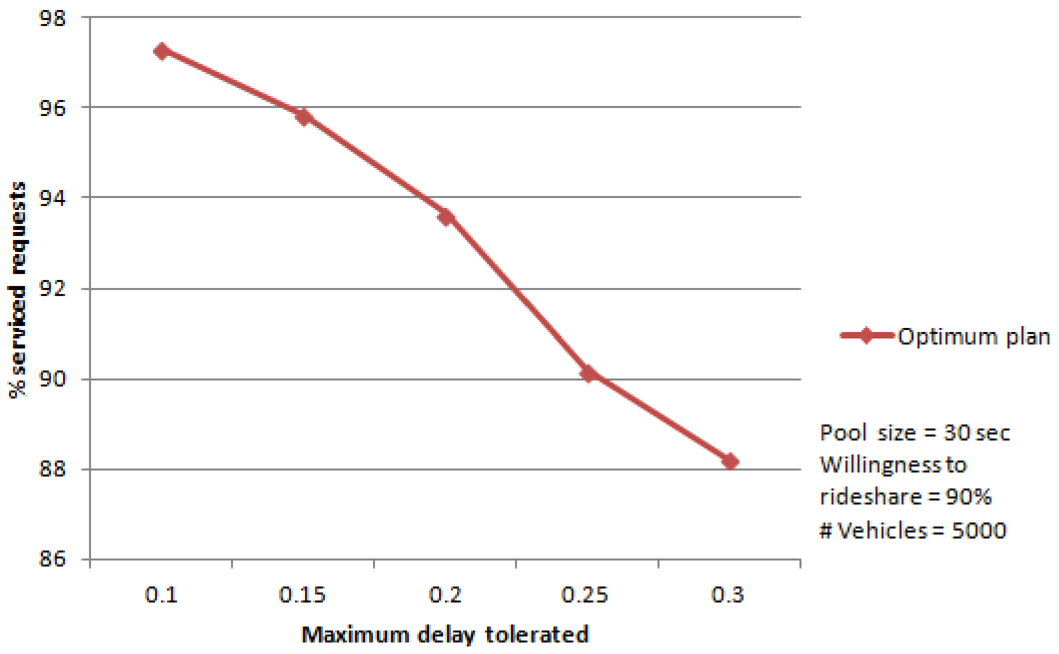

Figure 38: Dynamic model: % serviced requests with the optimum plan (maximum delay tolerated)

We have investigated whether those not serviced requests would bring a considerable increase in the % mileage reduction: thus, we have increased the number of vehicles to 6,000 and 7,000. In Figure 39, we can see that % reduction is almost the same in the three cases analyzed; Figure 40 shows that this holds also for the evenly split fair plan.

In Figure 41 we show that % serviced requests effectively increases as we provide more vehicles for the simulation.

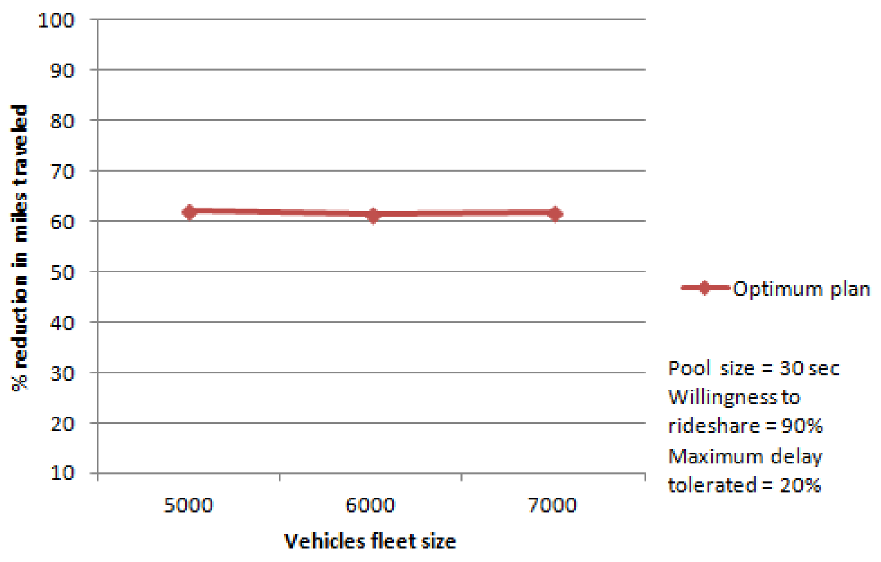

Figure 39: Dynamic model: % mileage reduction with the optimum plan (fleet size)

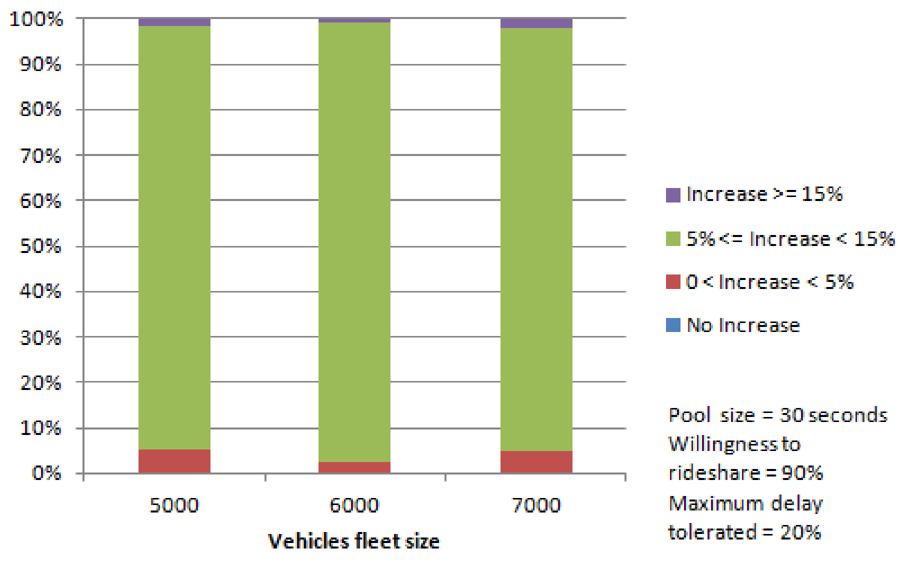

Figure 40: Dynamic model: optimum plan % increase over the fair plan (fleet size)

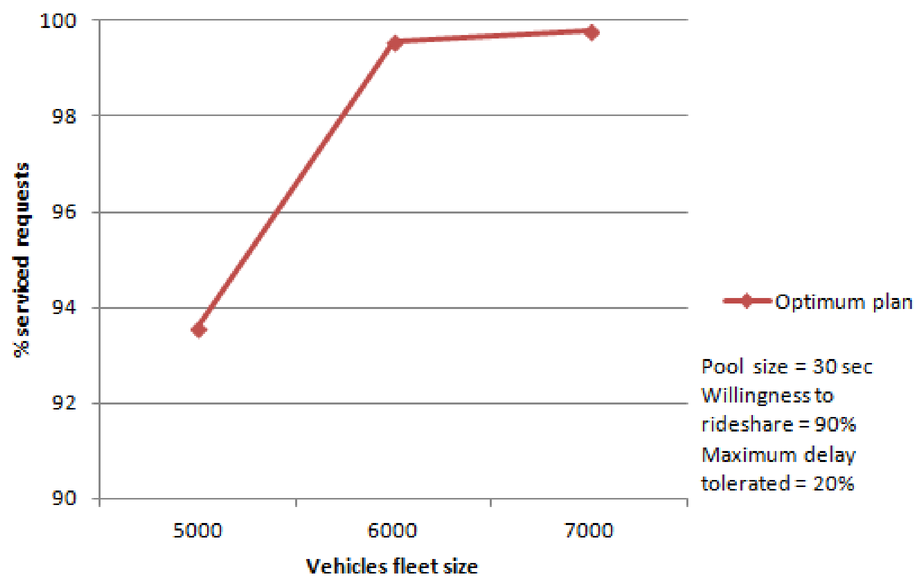

Figure 41: Dynamic model: % serviced requests with the optimum plan (fleet size)

As a result, increasing the number of vehicles will straightforwardly increase the % of satisfied requests, but this does not improve overall % mileage reduction.

5.3.2 Comparison between evenly and unevenly split fair plans

We compare the evenly and unevenly split fair plans with the same metrics described in section 5.1.2.

We have analyzed the case in which only two requests can be merged and processed by the same vehicle at the same time: for this reason, we have increased fleet size to 7,500 in order to satisfy the demand.

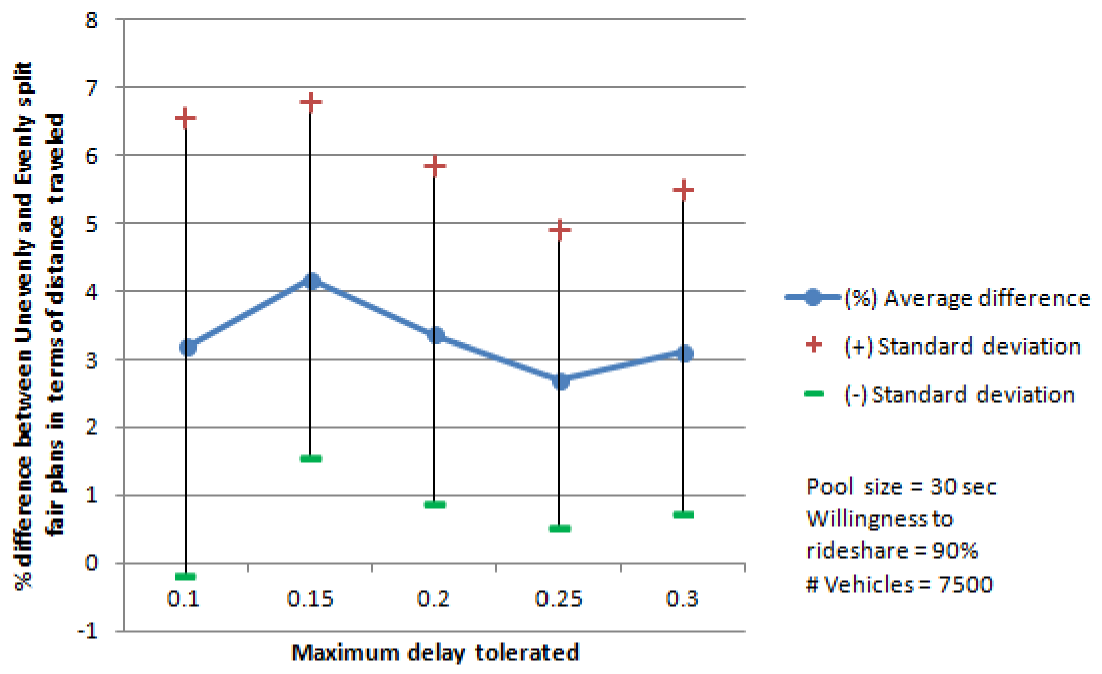

Figure 42: Dynamic model: comparison between unevenly and evenly split fair plans

In Figure 42 the comparison between evenly and unevenly split fair plans, with respect to maximum delay tolerated, is provided: % average difference and standard deviation are presented.

According to what standard deviation in Figure 42 outlines, % difference can be at most at around 7%: there is not a clear difference with respect to what presented in section 5.1.2. As a consequence, it is worth comparing the optimum plan to the evenly split fair plan only (which is tractable with respect to the unevenly split fair plan computation), also within the dynamic model.

Since a solution to the SRP (used to compute the unevenly split fair plan) may not always be retrieved, the % of this case occurrences is presented with respect to maximum delay tolerated (Figure 43).

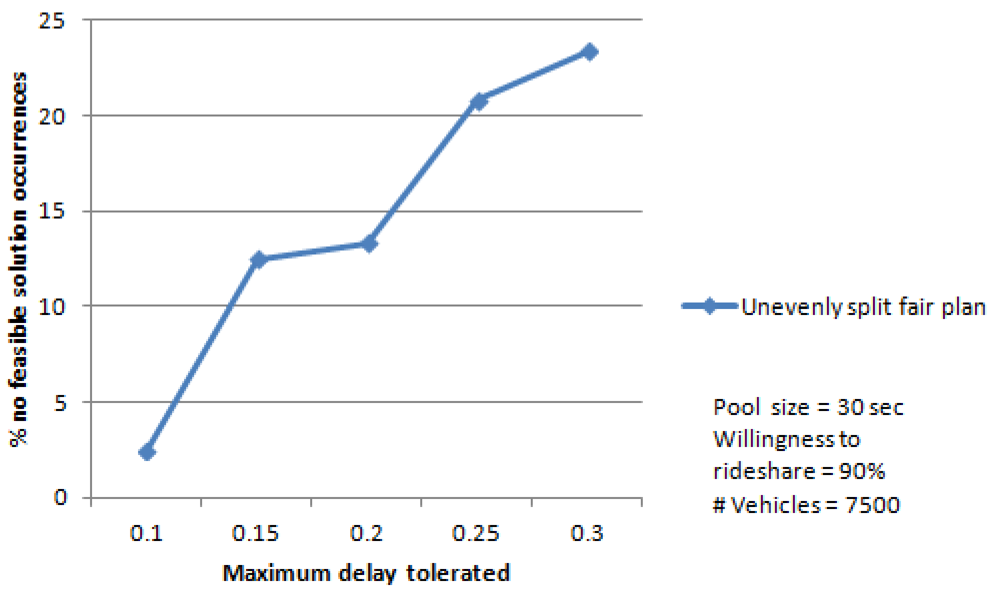

Figure 43: Dynamic model: % no solution to the SRP

As we can see, % no solution occurrences increases as the pools become larger (by increasing maximum delay tolerated).

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

We have introduced the concept of "fairness" in ride sharing. Savings for a passenger may be either evenly or unevenly split with respect to total saving the shared trip he/she participates in provides.

We have demonstrated with experiments that, in both the static and dynamic models, the two splits do not provide very different results in terms of total savings. As a consequence, the even split is preferred since the algorithm which deals with it is greedy and can be applied also to the case of combining more than two requests at a time. Moreover, the unevenly split fair matching algorithm is NP-complete in the case of combining more than two requests, but also in the easier case of combining at most two requests, a solution is not guaranteed.

With respect to the evenly split fair plan, we have studied the % redistribution of money required, since the optimum plan will always be issued eventually. We have discovered that in the static model almost all the times a redistribution of money is needed, in particular more than 5% and between 0% (excluded) and 5% are the ranges which occur more often and are equally probable (Figure 16, Figure 18, Figure 20); in the dynamic model, the redistribution of money lies in the 5%-15% range almost all the times (Figure 40) this means that, within this model, on average, more money needs to be redistributed compared to the static model.

6.2 Future work

Here we provide some options for future works.

First of all, in this thesis we have analyzed taxi fares which are computed on distance traveled only: thus, it is interesting to consider some other metrics (e.g., time traveled).

Another work can be to implement a new mobile application which allows to rank requests in a pool in order to provide a fair assignment. This can be done either manually or automatically: in the former case, “Uber” or “Lyft” can make pools of requests public, thus passengers can provide a list of preferences with respect to other passengers present in the pool (taking into account travel delay but also the characteristics of the other passengers); in the latter case, pools of requests are not public, but when submitting a ride sharing request, a passenger can specify a quantitative value for various parameters in order to compute an overall weight which takes into account, for example, money saved, travel time delay, the ride sharing partner characteristics.

Thus, fairness in ridesharing has been devised only with respect to money saved, but this definition can be extended to some other parameters which contribute in computing the overall weight for a shared trip.

CITED LITERATURE

1. Zhang, R. and Pavone, M.: Control of robotic mobility-on-demand systems: a queueing-theoretical perspective. The International Journal of Robotics Research, 35(1-3):186–203, 2016.

2. Wolfson, O. and Lin, J.: Fairness versus optimality in ridesharing. to appear in Proc. of Mobile Data Management (MDM) 2017.

3. Alonso-Mora, J., Samaranayake, S., Wallar, A., Frazzoli, E., and Rus, D.: On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. Proceedings of the National Academy of Sciences, page 201611675, 2017.

4. Santi, P., Resta, G., Szell, M., Sobolevsky, S., Strogatz, S. H., and Ratti, C.: Quantifying the benefits of vehicle pooling with shareability networks. Proceedings of the National Academy of Sciences, 111(37):13290–13294, 2014.

5. Lin, J., Sasidharan, S., Ma, S., and Wolfson, O.: A model of multimodal ridesharing and its analysis. In Mobile Data Management (MDM), 2016 17th IEEE International Conference on, volume 1, pages 164–173. IEEE, 2016.

6. Galil, Z.: Efficient algorithms for finding maximum matching in graphs. ACM Computing Surveys (CSUR), 18(1):23–38, 1986.

7. Bespamyatnikh, S. N.: An optimal algorithm for closest-pair maintenance. Discrete & Computational Geometry, 19(2):175–195, 1998.

8. Prosser, P.: Stable roommates and constraint programming. In International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems, pages 15–28. Springer, 2014.

9. Gusfield, D. and Irving, R. W.: The stable marriage problem: structure and algorithms. MIT press, 1989.

10. Bistaffa, F., Farinelli, A., Cerquides, J., Rodríguez-Aguilar, J. A., and Ramchurn, S. D.: Algorithms for graph-constrained coalition formation in the real world. arXiv preprint arXiv:1612.04299, 2016.

CITED LITERATURE (continued)

11. Bistaffa, F., Farinelli, A., and Ramchurn, S. D.: Sharing rides with friends: a coalition formation algorithm for ridesharing. 2014.

12. Iwama, K., Miyazaki, S., and Okamoto, K.: Stable roommates problem with triple rooms. In Proc. 10th KOREA-JAPAN Joint Workshop on Algorithms and Computation (WAAC 2007), pages 105–112, 2007.

13. Ma, S., Zheng, Y., and Wolfson, O.: Real-time city-scale taxi ridesharing. IEEE Transactions on Knowledge and Data Engineering, 27(7):1782–1795, 2015.

14. Nyc taxi and limousine commission. `http://www.nyc.gov/html/tlc/html/passenger/taxicab_rate.shtml`. Accessed: 2017-04-21.

15. Swoboda, A. J. T.: New york city taxicab transportation demand modeling for the analysis of ridesharing and autonomous taxi systems, 2015. B.S. thesis.

16. Bloomberg, M. and Yassky, D.: New york city 2014 taxicab factbook (new york city taxi and limousine commission, new york. `http://www.nyc.gov/html/tlc/downloads/pdf/2014_taxicab_fact_book.pdf`, 2014. Accessed: 2017-04-21.

VITA

NAME	Luca Foti
EDUCATION	
	Bachelor's Degree in Computer Science and Engineering,
	Sep 2015, Politecnico di Milano, Italy
	Pursuing Master of Science Degree in Computer Science and Engineering, Politecnico di Milano, Italy
LANGUAGE SKILLS	
Italian	Native speaker
English	Full working proficiency
	2012 - CAE examination (grade B)
	2015 - TOEFL iBT examination (105/120)
	A.Y. 2015/16. Lessons and exams attended exclusively in English
	A.Y. 2016/17 One Year of study abroad in Chicago, Illinois
SCHOLARSHIPS	
Spring 2017	Research Assistantship (RA) position (20 hours/week) with full tuition waiver plus monthly stipend
WORK EXPERIENCE AND PROJECTS	
Mar 2015 - Jun 2015	Thesis project for B.S. at Politecnico di Milano.
	It consisted of implementing a multiplayer board game in Java and allowing people to register and play together via socket and RMI connections.
Mar 2015 - Jun 2016	"High Performance Processors and Systems" project course at Politecnico di Milano: interactive floorplanning for FPGAs.
Mar 2015 - Jun 2016	"Hypermedia and Web Applications" project course at Politecnico di Milano: website and mobile application development for an Italian telephony company.