Taxi and Ride Sharing: A Dynamic Dial-a-Ride Problem with Money as an Incentive

Douglas O. Santos[a], Eduardo C. Xavier[a]

[a]*Institute of Computing, University of Campinas, 13083-852, Campinas, Brazil*

Abstract

This paper deals with a combinatorial optimization problem that models situations of both dynamic ride-sharing and taxi-sharing. Passengers who want to share a taxi or a ride, use an app to specify their current location, destination and further information such as the earliest departure time, the latest arrival time and the maximum cost they are willing to pay for the ride. Car owners also specify their origin, destination, the leaving time and the maximum accepted delay. Taxi drivers report their location and the time they will start and end the service. All drivers need to define a price per kilometer. The problem is to compute routes, matching requests to vehicles in such a way that ride-sharing is allowed as long as some restrictions are satisfied, such as: the capacity of the vehicle, maximum trip cost of each passenger, maximum delay, etc. The problem is dynamic since new requests arrive on-line and routes can be modified in order to attend them. To solve this dynamic problem, the day is divided in time periods. For each period, an instance of a static problem is created and solved by a greedy randomized adaptive search procedure (GRASP). Experiments with instances based on real data were made to evaluate the heuristics and the proposed method. In our simulations with taxis, passengers paid, on average, almost 30% less than they would pay on private rides.

Keywords: Taxi-sharing, Ride-sharing, Dial-a-Ride , Heuristics, GRASP

1. Introduction

The Dial-a-Ride problem is a well known problem in the literature, see Cordeau & Laporte (2007), and we study in this paper a version of this problem that models situations of both ride-sharing and taxi-sharing where people would be willing to share rides if they could save money with that. We call this problem by Dynamic Dial-a-Ride Problem with Money as an Incentive (DARP-M).

Ride-sharing is important for the sustainable development. According to Sims et al. (2014), greenhouse gas emissions from the transport sector have more than doubled since 1970, and have increased at a faster rate than any other energy end-use sector, being responsible for approximately 23% of the total of energy-related CO_2 emissions. Ride-sharing may also reduce the traffic congestion which is a serious problem in metropolitan areas. Moreover, it directly benefits people, reducing the costs with transportation and the

*Corresponding author: Eduardo C. Xavier
Email addresses: douglasantos@fb.com (Douglas O. Santos), eduardo@ic.unicamp.br (Eduardo C. Xavier)
This work was partially supported by Fapesp and CNPq

waiting time to get a ride or a taxi, because sometimes, the supply of vehicles is not enough for the big demand of passengers.

We consider the existence of a framework that allows users to share taxis and their own cars. The framework consists of two components:

- An app used by passengers to specify source and destination, time to be served, maximum time to be delivered, number of people, and how much they are willing to pay for the ride (in case of a taxi ride this should be the taxi cost of a sole trip). The app is also used by car owners and taxi drivers, who need to specify the following: source, destination, leaving time, maximum allowable time to be at destination, price per kilometer and the vehicle capacity.

- A server that receives all information, on demand, and matches vehicles to requests, computing routes for each vehicle.

The goal of this paper is to model a realistic situation of dynamic ride-sharing as a combinatorial optimization problem, and also to propose a method to solve the problem in large scale.

We consider ride-sharing and taxi-sharing as a unique problem, modeled in the following manner: at each instant of time, there is a set of passengers needing to travel from a source to a destination point, and there is a set of vehicles, each one having a source and a destination. Passengers have constraints that need to be considered. The constraints are: an earliest departure time, a latest arrival time, the number of passengers that will travel together and the maximum value they are willing to pay for the ride. Vehicles can also have an earliest departure time and a latest arrival time. They also must have a maximum capacity and a price per kilometer. The ride cost for each passenger must be calculated in a fair way. The cost of each part of the route is divided equally among all passengers aboard. The problem is to compute routes for each vehicle, where each route is a sequence of source and destination points, satisfying all the constraints and maximizing a multi-criteria objective function. The objective function consists of maximizing the number of attended requests and minimizing the total value paid by all passengers. The problem is called Dynamic Dial-a-Ride Problem with Money as an Incentive (DARP-M).

Notice that ride-sharing and taxi-sharing have some differences. In the case of ride-sharing, usually a private car owner has a specific trip route and he can accept to give someone a ride if this person is going to a close location of his final destination. In the case of taxi sharing, the taxi driver does not have a specific route unless it is attending some other passenger. Taxi drivers do not have time restrictions either, while in the case of ride sharing, the private car owner can have very strict time restrictions. Also, the prices for taxis are fixed while in the case of ride-sharing each driver can specify a different charge. Despite some differences, it is possible to see that our model can deal with both problems.

Note that, the framework can impose other constraints, such as the one presented in Santos & Xavier (2013), which uses a social network and allows people to choose if they want to share a ride only with friends or friends of friends; in Tao (2007), people can choose to share only with people of the same sex; in Lalos

et al. (2009), there is a central responsible for motoring all vehicles to see if they are following a pre-specified route. Moreover, frameworks presented in the literature can be adapted to consider the DARP-M.

1.1. Related Work

There are other works in the literature that deals with ride-sharing problems. Most of them are derived from the Dial-a-Ride Problem, see Cordeau & Laporte (2007), which is a version of the Pickup and Delivery Problem (PDP), see Parragh et al. (2008a). The main difference between the problem of this work and other ride-sharing problems is the fact that the DARP-M has cost constraints which do not allow people to pay more for a shared ride than they would pay for a private one. In Herbawi & Weber (2012), a problem called Ridematching Problem with Time Windows (RMPTW) is presented. RMPTW is very similar to a Dial a Ride Problem (DARP), but with a different objective function and some additional constraints. The proposed method alternates between a genetic algorithm and an insertion heuristic to solve static snapshots of the problem. In Agatz et al. (2011), another kind of dynamic ride-sharing problem is presented, but it is a simpler version, as each driver may pass through only one pickup and delivery location. They modeled the problem as a maximum-weight bipartite matching problem and solved it using the optimization software CPLEX. In Ma et al. (2013), a dynamic taxi-sharing problem is defined and the proposed method solves directly the dynamic version using insertion heuristics. A framework is also presented. In Horn (2002), a software system to manage the deployment of a fleet of demand responsive passenger vehicles, including shared taxis, is described. The proposed method solves the problem with insertion heuristics and local searches. A recent survey describing recent advances in the field of dynamic ride-sharing is presented in Agatz et al. (2012).

1.2. Our Contributions

In this work we study a new variant of the Dial a Ride Problem where cost restrictions must be taken into consideration. This new problem models situations of taxi sharing and ride sharing where people would be interest in participating if they had economic advantages. For that, the problem imposes that shared trips must cost less than what users are willing to pay.

In order to solve the static problem we propose a new GRASP heuristic with Path Relinking. We performed several tests with variations of the GRASP heuristic, obtaining one that performs much better than our previous heuristic for taxi-sharing presented in Santos & Xavier (2013).

We also consider the dynamic version of the problem. In this case requests are buffered during sometime and then a static version of the problem is constructed and solved using the GRASP heuristic. Some modifications must be made in the heuristic, since the route for a vehicle might have already been set on a previous iteration, and because of that some passengers might have already been served or are still being served. One crucial point to solve the dynamic problem is to compute shortest paths between all source and destination points in a very efficient way. All points are assumed to be in a graph representing the map of a city. In our solution, we used Contraction Hierarchies, see Geisberger et al. (2008), which was hundred of

times faster to compute many-to-many shortest paths than a simple approach using Dijkstra shortest-path algorithm.

We made computational experiments to asses the performance of our heuristics, using instances that simulate taxi activities in the city of São Paulo. In order to do that, we used the real graph obtained from Open Street Map [2]. In our simulations, passengers paid, on average, almost 30% less than they would have paid on private rides.

1.3. Text Organization

In Section 2, we define the Dial-a-Ride Problem with Money as an Incentive. Its textual description is presented in Subsection 2.1 and its mathematical model is presented in Subsection 2.2. This section also shows the proof of NP-Hardness (Subsection 2.3) and DARP-M's dynamic version (Subsection 2.4). In Section 3, we propose a method to solve the dynamic problem. In Subsection 3.1, we present GRASP heuristics that solve the static problem and are necessary for the proposed method. Section 4 shows the results of our computational experiments. At last, in Section 5, some conclusions are presented.

2. The Dial-a-Ride Problem with Money as an Incentive

This section describes the optimization problem called Dial-a-Ride Problem with Money as an Incentive (DARP-M). It represents realistic situations of ride-sharing. The problem is general and can be used to represent taxi-sharing or even a situation where there are taxis and car owners, who want to share their vehicles to reduce trip costs. We present the textual description (Section 2.1) and the mathematical model (Section 2.2). We also prove that DARP-M is a NP-Hard problem (Section 2.3) and Section 2.4 shows what changes when the dynamic version is considered.

2.1. Description

The DARP-M receives a set N with n requests of passengers and a set M with m vehicles that are available for ride-sharing.

Each request $i \in N$ has a source point i^+, a destination point i^-, an earliest departure time e_i, a latest arrival time l_i, the number of passengers p_i and also the maximum value, $s_i > 0$, passengers are willing to pay for the ride (in most of the cases s_i is the cost of a private ride). In order to be served, the request i needs to have all passengers p_i traveling together from i^+ to i^- in only one vehicle. Also, they must leave their source point not before e_i and must arrive not after l_i. Moreover, they cannot pay more than s_i.

Each vehicle $k \in M$ has a source point k^+, a destination point k^-, a maximum capacity of passengers q_k, an earliest departure time e_k and a latest arrival time l_k.

Let $V = \{i^+, i^- | i \in N\}$ be a set of all source and destination points of all n requests and $W = V \cup \{k^+, k^- \, | k \in M\}$ be the set of all $2n + 2m$ points, including the vehicles points. Even if two points represent

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

the same location, they will be considered two distinct points. Each point $w \in W$ has a service duration d_w, which is the time taken to do a pickup or delivery operation, and the number of passengers p_w that is going to get in or get off the vehicle. If w is a destination point, p_w will be negative. For vehicle points we have $p_w = 0$, so we do not consider the driver as a passenger.

Moreover, each vehicle k has two matrices $|V \cup \{k^+, k^-\}| \times |V \cup \{k^+, k^-\}|$ denoted by T^k and C^k that indicate respectively the travel time and cost, between all pairs of points in $V \cup \{k^+, k^-\}$.

The route $R_k = \{u_1, u_2, ..., u_z\}$ of a vehicle k is a sequence of points sorted by the time they will be visited. The first and the last point of the route, u_1 and u_z, are the source k^+ and the destination point k^-. Other points correspond to the points of passengers. If a request i is being served by k then it must have i^+ and i^- in R_k, and i^+ must be before i^-.

For each point u_j in a route R_k we define the time $B^k_{u_j}$ as the time when the point u_j will be served. We have $B^k_{u_1} = e_k$ and for $j > 1$, $B^k_{u_j} \geq B^k_{u_{j-1}} + t^k_{u_{j-1},u_j} + d_{u_{j-1}}$. And also, it must respect the time window. So if $u_j = x^+$, $B^k_{u_j} \geq e_x$ and if $u_j = x^-$, $B^k_{u_j} \leq l_x$. For each point in the route we also need to set its load $L^k_{u_j}$, that indicates the number of passengers in k right after point u_j is served. So, for $j > 1$, $L^k_{u_j} = L^k_{u_{j-1}} + p_{u_j}$ and for $j = 1$, $L^k_{u_1} = 0$. $L^k_{u_j}$ must be less than or equal to q_k.

At last, the cost of each ride needs to be calculated. In a shared trip, the cost of going from u_j to u_{j+1}, where u_{j+1} is immediately after u_j in R_k, is divided equally among all passengers inside the vehicle. So let S^k_{i,u_j} be the cost that passengers of request i must pay, if they are inside vehicle k, when it goes from u_j to u_{j+1}. We have that, $S^k_{i,u_j} = \frac{c^k_{u_j,u_{j+1}} \cdot p_i}{L^k_{u_j}}$, where $c^k_{u_j,u_{j+1}}$ is the cost of going from u_j to u_{j+1} in the vehicle k. If we sum every cost of a request i, we will get S_i which must be at most s_i.

To summarize, the Dial-a-Ride Problem with Money as an Incentive is to build a route for each vehicle, subject to some constraints:

Precedence: The source point of a request must be served before the destination one.

Capacity: Each vehicle has a maximum capacity of passengers.

Time Window: Each request and vehicle has a time window that must be satisfied.

Shared Cost: Each request defines a maximum cost to be payed for a shared ride.

No transfer: A request cannot be served by more than one vehicle.

Demand: All passengers of the same request must travel together.

Moreover, we need to optimize two criteria.

1. Maximize the number of served requests.
2. Minimize the sum of the costs of all served requests.

Section 2.2 shows the objective function that takes those two criteria into account.

2.2. Mathematical model

This section shows the mathematical model for the DARP-M.

We have four kinds of decision variables.

- $x^k_{u,v} = \begin{cases} 1, & \text{if point } v \text{ is immediately after point } u \text{ in the route of vehicle } k. \\ 0, & \text{otherwise.} \end{cases}$
- B^k_u = the time of the beginning of the service at point u in the route of vehicle k. If u does not belong to R_k, this variable will be ignored.
- L^k_u = the number of passengers inside vehicle k, right after it served point u. If u does not belong to R_k, this variable will be ignored.
- $S^k_{i,u}$ = cost to be payed by request i, in case when its passengers are inside vehicle k when it goes from point u to the next point in its route. If i is not served by k or if u is not in R_k, this variable will be ignored.

Next, we present the model:

$$\max \sum_{i \in N} \sum_{k \in M} \sum_{v \in V} \left(x^k_{i^+,v} - \lambda \frac{S^k_{i,v}}{s_i} \right) \tag{1}$$

subject to:

$$\sum_{k \in M} \sum_{v \in V} x^k_{i^+,v} \leq 1 \qquad \forall i \in N \tag{2}$$

$$\sum_{v \in V \cup \{k^-\}} x^k_{k^+,v} = 1 \qquad \forall k \in M \tag{3}$$

$$\sum_{v \in V \cup \{k^+\}} x^k_{v,k^-} = 1 \qquad \forall k \in M \tag{4}$$

$$\sum_{v \in V \cup \{k^+\}} x^k_{v,u} - \sum_{v \in V \cup \{k^-\}} x^k_{u,v} = 0 \qquad \forall u \in V, \forall k \in M \tag{5}$$

$$\sum_{v \in V} x^k_{i^+,v} - \sum_{v \in V} x^k_{v,i^-} = 0 \qquad \forall i \in N, \forall k \in M \tag{6}$$

$$x^k_{u,v} \in \{0,1\} \qquad \forall k \in M, (u,v) \in V \cup \{k^+\} \times V \cup \{k^-\} \tag{7}$$

$$(x^k_{u,v} = 1) \Rightarrow B^k_v \geq B^k_u + t^k_{u,v} + d_u \qquad \forall k \in M, \forall u, v \in V \cup \{k^+, k^-\} \tag{8}$$

$$e_i \leq B^k_{i^+} \leq B^k_{i^-} \leq l_i \qquad \forall i \in N, \forall k \in M \tag{9}$$

$$e_k \leq B^k_{k^+} \leq B^k_{k^-} \leq l_k \qquad \forall k \in M \tag{10}$$

$$(x^k_{u,v} = 1) \Rightarrow L^k_v = L^k_u + p_v \qquad \forall k \in M, \forall u, v \in V \cup \{k^+, k^-\} \tag{11}$$

$$0 \leq L^k_u \leq q_k \qquad \forall k \in M, \forall u \in V \tag{12}$$

$$L^k_{k^+} = L^k_{k^-} = 0 \qquad \forall k \in M \tag{13}$$

$$(x^k_{u,v} = 1) \wedge (B^k_{i^+} \leq B^k_u < B^k_{i^-}) \Rightarrow S^k_{i,u} = \frac{c^k_{u,v} p_i}{L^k_u} \qquad \forall k \in M, \forall i \in N, \forall u \in V, \forall v \in V \tag{14}$$

$$\sum_{k \in M} \sum_{v \in V} S^k_{i,v} \leq s_i \qquad \forall i \in N \tag{15}$$

$$S^k_{i,v} \geq 0 \qquad \forall i \in N, \forall k \in M, \forall v \in V \tag{16}$$

The objective function maximizes the number of served requests and minimizes the total cost. In order to clarify, we rewrite it using two auxiliary variables. Let y_i be a binary variable which is equal to one, if

and only if, i is served. Let S_i be the total cost to be payed by the passengers of request i. The objective function is: maximize $\sum_{i \in N} \left(y_i - \lambda \frac{S_i}{s_i} \right)$. Note that $y_i = \sum_{k \in M} \sum_{v \in V} x^k_{i^+,v}$ and $S_i = \sum_{k \in M} \sum_{v \in V} S^k_{i,v}$. Also note that s_i is greater than or equal to S_i. The smaller $\frac{S_i}{s_i}$ is, the bigger are the savings. We use a constant $\lambda \in [0, 1]$ to balance the second criterion.

Constraint (2) guarantees that each request i is served at most once, by only one vehicle. Constraints (3) and (4) ensure that each vehicle k has its source and destination points in its route. Constraint (5) is a flow conservation constraint, each point $u \in V$ must have its in-degree equal to its out-degree, which is at most one. And constraint (6) guarantees that the source and destination points of a request will be served by the same vehicle, if the request is served.

Constraints (8), (9) and (10) are for the B variables, while (11), (12) and (13) are for the L variables.

All those constraints, (2)-(13), are also presented in the Pickup and Delivery Problem with Time Windows (PDPTW) (Parragh et al. (2008a), Parragh et al. (2008b)).

Constraints (14), (15) and (16) are related to the shared costs, and they are not presented in any other problem in the literature. Constraint (14) calculates the value that passengers must pay in each part of the route. Constraint (15) guarantees that the total cost of each request is not greater than the maximum allowed. When a variable $S^k_{i,u}$ is ignored because i is not served by k or because u is not in R_k, it is always optimal to have its value set to zero.

2.3. NP-Hardness

In this section we show that DARP-M is a NP-Hard Problem, using a version of the Hamiltonian path problem where the start and end vertices of the path are fixed. Let $G = (V, E)$ be a complete undirected graph with costs $w_{u,v} > 0$ for each edge $(u, v) \in E$. The problem is to check if there is an Hamiltonian path, in G, starting at u^+ and ending at u^-, with cost not greater than Y, where $u^+, u^- \in V$ and $u^+ \neq u^-$.

The decision version of the DARP-M is to check if there is a feasible solution with an objective function value greater than or equal to Z. Notice that the NP-hardness of the DARP problem was shown in Baugh Jr. et al. (1998), via a reduction from the TSP to the DARP problem, where in this case in both problems the objective is to find a cycle (route) of minimum total cost. However, in the DARP-M, the objective function is quite different, and edges costs are not part of it, since the main objective is to attend the maximum number of requests while reducing the costs to be paid.

The described version of the Hamiltonian path can be reduced to the decision version of the DARP-M. Let $I = (G, u^+, u^-, Y)$ be an instance of the Hamiltonian problem. An instance $J = (N, M, T, C, Z)$ for the DARP-M is created from I. First, a set M is created with only one vehicle k. The vehicle has source u'^+ and destination point u'^-, where u'^+ and u'^- correspond to vertices u^+ and u^- from I. It also has $e_k = 0$, $l_k = Y$, $q_k = 1$ and $d_{k^+} = d_{k^-} = 0$. A set N is created with $|V| - 2$ requests, one for each vertex $v \in V$, except u^+ and u^-. Each request i corresponding to a vertex $v \in V$, has $i^+ = i^- = v'$, $e_i = 0$, $l_i = Y$, $p_i = 1$, $s_i = 1$ and $d_{i^+} = d_{i^-} = 0$. The travel time $t^k_{u,v}$ between vertices u and v is equal to $w_{u,v}$ and the travel costs

are zero. Finally, $Z = |N|$, meaning that all requests must be served.

In order to prove the reduction, the following statement must be proved: there is a Hamiltonian path starting at u^+ and ending at u^-, in the instance I, with cost at most Y, if and only if there is a feasible solution for the instance J of the DARP-M with objective function value greater than or equal to Z. In order to prove it just observe that the same sequence of vertices used in the Hamiltonian path can be used to obtain the required solution for the DARP-M. On the other hand, a solution of the DARP-M induces a Hamiltonian path, since every request is attended and there is no repetition of vertices, because the vehicle capacity is equal to one. Also the vehicle arrives at its destination point by its latest arrival time Y.

2.4. Dynamic Version

Since we want to model a real situation of ride-sharing, we need to consider the dynamic version of the DARP-M.

We consider the existence of a graph $G = (U, E)$ representing a city map, where U is the set of vertices representing intersections, points of interest, etc, in the city, and E is the set of edges representing streets connecting these points. Requests and vehicles arrive on demand with source and destination points in U.

In the dynamic version, we call the set of requests by $\mathbb{N}$ and the set of vehicles by $\mathbb{M}$. The set $\mathbb{N}$ has requests that are waiting to be served. At the beginning, $\mathbb{N}$ is empty. At each instant of time, new requests may arrive and are added to $\mathbb{N}$. Also, requests can be removed, because they cannot be served anymore, since it is too late to satisfy their time window, or because they were already served.

In the same way, the set $\mathbb{M}$ is not static and it has all vehicles that are available at the current time. Each vehicle $k \in \mathbb{M}$ has a scheduled route R_k starting at k^+ and ending at k^-. The source point k^+ is the current position of the vehicle and e_k must be greater than or equal to the current time. R_k may already have points, different from k^+ and k^-, that correspond to the requests that are going to be served. Moreover, new requests can be added at any time, but the route must always be feasible, and once a request is added to R_k, it cannot be removed. One important point is that if there are passengers aboard the vehicle at the current time, then new requests can only be added after the destination point of the last delivered passenger aboard. This is necessary to prevent changes in the price charged from these passengers. This can be accomplished by setting k^+ to be the destination point of the last delivered passenger aboard, and e_k to be the time the vehicle will be at this point. Each vehicle k defines a travel time and a travel cost for each edge $(i, j) \in G$. But in a practical situation, the vehicles are divided in classes such that vehicles in the same class have the same values.

The objective function and all the constraints of the static version of DARP-M, described in Section 2.2, are still presented in the dynamic problem.

3. The Proposed Method

In order to deal with the dynamic problem, the day is divided in time periods of same length. In the beginning of a period, an instance for the static DARP-M is created considering the current sets $\mathbb{N}$, $\mathbb{M}$ and the graph G. This instance is solved by a heuristic, which must finish its execution before the end of the period. After the execution, $\mathbb{N}$ and $\mathbb{M}$ are updated based on the new solution. Our heuristics are described in Section 3.1.

To create an instance for the static DARP-M, the sets N and M and the matrices T and C need to be filled.

The set M has all vehicles in $\mathbb{M}$. But we change k^+, e_k and R_k, if a vehicle k has passengers inside at the moment when the instance is being created, as mentioned in the previous section. The source k^+ is changed to the first point u in R_k where the vehicle will be empty ($L_u^k = 0$). All points in R_k prior to u will be removed and considered to be served. The requests that have these points will also be removed from $\mathbb{N}$. The earliest departure time e_k will be changed to B_u^k. The route remains the same after u, but the next heuristic execution is allowed to modify it. Points can be added, removed, swapped and so on.

The set N has all requests in $\mathbb{N}$, which are the requests not served and the ones which were already matched to be served in the future.

In order to compute the matrices T^k and C^k for each vehicle k, we need to calculate the shortest path in the graph G between all points $w \in W$. Since G represents a city, it can have millions of vertices, but W is a small subset of it. There are exact algorithms in the literature that solve efficiently the many to many shortest path problem. In our experiments we used Contraction Hierarchies, see Geisberger et al. (2008), which is a very fast algorithm to compute shortest paths in road networks and it can be adapted to answer many to many queries as presented in Schultes (2008), using the same method presented in Knopp et al. (2007) for Highway Hierarchies.

3.1. GRASP Heuristics

We propose a heuristic to solve the static version of the DARP-M that is based on GRASP, see Feo & Resende (1995). Each iteration of a GRASP heuristic has two phases: in the first phase, a greedy randomized initial solution (Section 3.1.1) is computed. The greedy strategy leads to good solutions and randomization is used so that different initial solutions can be constructed. In the second phase, a local search is performed in order to improve the solution found in the first phase (Section 3.1.3). We also use the Path Relinking technique (Section 3.1.4), which explores new solutions in the path between two known solutions. The procedure ends after a certain number of iterations or a maximum allowed time.

3.1.1. Initial Solutions

We propose three different greedy methods, GA, GB and GC, that can be used to build an initial solution. Methods GA and GB use sequential insertions, but with a different greedy function. Method GC uses parallel insertions.

In the sequential method, one vehicle is processed at a time. They are selected randomly. When processing a vehicle k, we try to update its route by inserting new requests. A set P containing all requests $i \in N$, that do not belong to any route, is created. While P is not empty, the following procedure is repeated: for each request $i \in P$, a greedy function μ_i^k (described in Section 3.1.2) is calculated. This function measures how good is to insert i in R_k. After that, a Restricted Candidate List (RCL) is built, containing requests with the best values of μ_i^k. Let μ_{min}^k and μ_{max}^k be the smallest and the largest value of a greedy function and assume the lower the value the better it is. A request i is included in the RCL if $\mu_k^i \leq \mu_{min}^k + \alpha(\mu_{max}^k - \mu_{min}^k)$, where α is a [0, 1] parameter. A request $i \in$ RCL is selected randomly and an attempt to insert i in R_k is made (insertion methods are described in Section 3.1.5). Note that the greedy function does not know if the request can be inserted in R_k, so not always there will be successful attempts. Finally, the selected request i is removed from P.

The difference in the parallel method is the fact that no vehicle is fixed to be processed, so the set P contains pairs (k, i) of a vehicle k and a request i.

The parameter α can have a significant influence in the method. But finding the best value of α is not an easy task, since it may differ in each instance. In order to reduce the number of parameters to be tuned, we propose a reactive approach. The idea is to collect statistics about the previous iterations to choose a good α for the current one. First, the domain of α is discretized, having only 11 possible values: $\{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$. For each value, we keep a variable containing the average of the objective function values of solutions found using this α. The first 11 iterations are executed using a different α, one for each possible value. After that, in each iteration, a new α is selected randomly with probability proportional to the value kept in their variables. At the end of each GRASP iteration, we update the variable representing the chosen α with the best objective function value found during the iteration.

3.1.2. Greedy Functions

We have two different greedy functions. The first one is used in GB and GC and the second one in GA.

Let $\delta_{v,u}^k$ be the amount of time that the duration of route R_k is increased if point u is inserted right after point v in R_k.

If u is a source point then:

$$\delta_{v,u^+}^k = \begin{cases} t_{v,u^+}^k + t_{u^+,v+1}^k + d_{u^+} - t_{v,v+1}^k & \text{if } B_v^k + t_{v,u^+}^k + d_v \geq e_u \\ e_u - (B_v^k + d_v) + t_{u^+,v+1}^k + d_{u^+} - t_{v,v+1}^k & \text{if } B_v^k + t_{v,u^+}^k + d_v < e_u \end{cases}$$

Else:

$$\delta_{v,u^-}^k = \begin{cases} t_{v,u^-}^k + t_{u^-,v+1}^k + d_{u^-} - t_{v,v+1}^k & \text{if } B_v^k + t_{v,u^-}^k + d_{u^-} \leq l_u \\ \infty & \text{if } B_v^k + t_{v,u^-}^k + d_{u^-} > l_u \end{cases}$$

With that, we can calculate our first greedy function

$$\mu_i^k = min_{v \in R_k} \delta_{v,i^+}^k + min_{v \in R_k} \delta_{v,i^-}^k,$$

that represents the minimum amount of time that the duration of R_k is going to be increased if request i is inserted on it. However, note that the insertion points considered may be infeasible.

The second greedy function computes the amount of increase in the objective function if i is inserted in R_k. To calculate it, we try to insert $i+$ and i^- in R_k considering all possible places, and the best one is chosen. If there is no place to insert i^+ and i^- that makes the route feasible, $\mu_i^k = -\infty$.

Note that for the first greedy function, the smaller the value the better it is, but for the second, the best value is the largest one.

3.1.3. Local Search

This section describes the local search called L1 which is based on the classical swap-operator.

In each iteration of the local search, there is a current solution, which will have its neighborhood explored in order to obtain a better solution. It is a First Improvement method, so anytime a better solution is found, we replace the current solution by the better one and a new iteration begins. For each current solution, L1 iterates over all pairs $(i, j) \in N$, in a random order. For each pair (i, j), i and j are removed from their routes and attempts to insert i in the old route of j and j in the old route of i are made (insertion methods are described in Section 3.1.5). Note that this procedure is done even if one of them does not have a route, or if they are in the same route. The local search is performed until there is no improvement and the best solution found is returned.

3.1.4. Path Relinking

This section describes the Path Relinking method that is performed after the local search. We have two solutions s_1 and s_2 where s_1 is the initial solution and s_2 is the guiding one. The idea is to create a set of operations that when applied one by one over s_1, we obtain in each application a new solution that is more similar to s_2. After the application of all the operations over s_1, it becomes equal to s_2. In each step, a new solution is found, which can be better than s_1 and s_2. At the end of the procedure, the best solution found is returned.

Our specific procedure of path relinking works as follows: we have a current solution s, where in the beginning $s = s_1$, which we want to transform in s_2. Each step of the procedure consists in finding requests that: (1) are not served in s but are served in s_2; (2) are served in s but not in s_2, and (3) are served by different vehicles. An operation consists of: in case (1) include the request served in s_2 in the same vehicle in s; (2) remove the request from s; (3) change a request from one vehicle to the same vehicle it is served in s_2. Among all these operations, the one that has the best increment in the objective function value, after the changes, is chosen. The method ends when there is no operation that leads to a feasible solution.

Let Π be the elite set of solutions found during the execution of the heuristic and let γ be a parameter that indicates the maximum allowed size of this set. At the end of each GRASP iteration, we have a solution s', which is the best solution found in the iteration. Each solution s' is a candidate to enter in Π and it will enter if Π is not full or if s' is better than the worst elite solution. In the last case, we remove the one, among those which is not better than s', that has the greatest similarity with s'. A request $i \in N$ is similarly attended in two solutions if it is served by the same vehicle in both solutions or if it is not served in both solutions. The similarity between two solutions is defined as the number of requests $i \in N$ that are similarly attended.

We opted for the Backward Path Relinking, so the guiding solution s_2 will be the solution returned by the local search and the initial solution s_1 will be a solution in Π. The elite solution is chosen randomly with probability proportional to the similarity with the guiding solution, in a way that solutions with less similarity are preferred. We chose Backward Path Relinking because our method can be interrupted before we get to s_2 (Truncated Path Relinking), so it will spend more time near s_1 which is more probable to have better solutions, since s_1 is an elite solution.

After the path relinking, the local search is applied to the returned solution, since it is not guaranteed to be a local optimum.

3.1.5. Insertion Methods

This section describes our two insertion methods, called IA and IB.

Given a feasible route R_k of a vehicle k and a request i that was not served, the problem is to insert the source i^+ and destination point i^- in R_k, in a way that it remains feasible. The problem also consists of maximizing the new objective function value, which will be obtained after the insertion.

Method IA is an exhaustive method. For each pair of points (a, b) in R_k, where b is located after a and $b \neq k^-$, i^+ is inserted immediately after a and i^- is inserted immediately after b. After that, the service time and the number of passengers at each point of the route, the shared costs and the new objective function value are recalculated. In the end, the pair with the best result is chosen. Note that, pairs (a, a) are also considered, in this case, i^+ is inserted after a and i^- after i^+.

IB is a heuristic method. First, a set Δ^+ is created with the β points $j \in R_k$ (β is a parameter) that have the smallest delay δ^k_{j,i^+}, when i^+ is inserted immediately after j. For each pair (a, b), where $a \in \Delta^+$, $b \in R_k$, a is located before b and $b \neq k^-$, i^+ is inserted immediately after a and i^- is inserted immediately after b. The same procedure is done, but now considering point i^-. So, a set Δ^- is created with the β points $j \in R_k$ with smallest δ^k_{j,i^-}. For each pair (a, b), where $a \in R_k$, $b \in \Delta^-$, a is located before b and $b \neq k^-$, i^+ is inserted immediately after a and i^- is inserted immediately after b. Notice that Δ^+ contains presumably the closest β points to i^+ while Δ^- contains possible the closest β points to i^-. After these two procedures, the feasible route with the best objective function value is chosen.

Both methods report a failure in case where no feasible insertion was found.

4. Computational Results

This section presents the results and analyses of experiments involving our heuristics for the static DARP-M (Section 4.1) and the proposed method for the dynamic version (Section 4.2).

All experiments were made in an Intel Xeon X3430, 2.4Ghz, with 8Gb of memory, using Linux (Ubuntu). The source code was implemented in C++ and compiled with gcc 4.6.

We used $\lambda = 0.99$ to balance the second criteria of the objective function, so it will always be better to serve a request, even if there is no sharing.

4.1. Comparative Results between the GRASP Heuristics

This section shows the results of the experiments involving all heuristics described in Section 3.1. There are three methods to build an initial solution, GA, GB and GC, two insertion methods, IA and IB, a local search, L1 and a path relinking method, PR. Moreover, the heuristics have a parameter α, used in GA, GB and GC, a parameter β, used in IB, and a parameter γ, used in PR. It is also possible to use the reactive version instead of choosing a value for α. There are several possible combinations, each one resulting in a different heuristic. In order to obtain the best setup for a GRASP heuristic, we made a series of computational experiments.

Each possible heuristic was executed 15 times, with different seeds, for each instance. The average of the 15 results was taken and then used as the final result. The time limit is 150 seconds.

We used the data of the Open Street Map (OSM) [3] to create a graph of São Paulo, containing 828635 nodes and 1755786 edges, among those, 136700 are one-way. This graph was used to create 80 instances for the static DARP-M. In all instances, we used the fact that there are 32k taxis in São Paulo city and 78k requests per day [4].

Among the 80 instances, 40 have 500 requests while the others have 750. The number of available vehicles in each instance is proportional to the number of requests, obeying the 78/32 ratio, with 205 and 307 vehicles, respectively. Each request has an earliest departure time equal to 0. The latest arrival time depends on the travel time between the source and destination points, and the maximum delay allowed. There are instances with maximum delay of 30 and 45 minutes and others with maximum delay proportional to the travel time, with multipliers of 1.3 and 1.5. We denote the maximum delay parameters by $30a$ and $45a$ respectively, and the proportional delays by $1.3r$ and $1.5r$ respectively. For example, if some request i is such that the time to go from i^+ to i^- is 30 minutes and the maximum delay has a multiplier of 1.3, then $l_i = 39$, but if the maximum delay is 30 minutes, then $l_i = 60$. The distance between each pair of points is the distance of the shortest path in the graph, and the travel time is calculated considering that each vehicle has a velocity of 40km/h. The travel cost is one dollar per minute. Each request is only for one passenger and the maximum

[3] http://www.openstreetmap.org/

[4] http://www1.folha.uol.com.br/cotidiano/773125-taxi.shtml

allowed price is the cost of a private ride going from the source to the destination. Each vehicle has a maximum capacity of 4 passengers, the earliest departure time equal to 0 and does not have a latest arrival time, which means it is always available. The duration of the service in each point is zero.

To summarize, for each number of requests $n \in \{500, 750\}$ and for each maximum delay $d \in \{30a, 45a, 1.3r, 1.5r\}$, there are 10 different instances, each one with different source and destination points, defined randomly. Each instance is called sp-n-d-x, where x is the instance number (0 to 9).

All comparisons were made graphically, using a method known as Performance Profiles, see Dolan & Moré (2002), which was proved to be robust against outlier instances when using a large set of instances. The idea is to compare the different heuristics against themselves. Each heuristic is executed in all instances. For each instance, it is determined the best solution generated among all heuristics. Then, for each heuristic, it is computed the ratio between the value of the best solution and the value of the solution found by this heuristic. With these data we can determine, for example, the percentage of best solutions found by a specific heuristic, by counting the number of ratios equal to one. As another example, it is possible to determine the percentage of solutions found by a heuristic whose cost is at most two times worse than the best one, by determining the number of ratios less than or equal to two. With these data, we create a $2D$ graph with a line for each heuristic. Each point (τ, p) of a line is the fraction p of instances that have a ratio, in relation to the best, less than or equal to τ. For example, the point $(1.0, 0.4)$ means that 40% of the instances have the best results, while the point $(1.5, 0.9)$ means that 90% of the instances have a ratio at most 1.5.

A complete set of experiments comparing different setups of the heuristic were done in order to discover the best combination of methods to be used. We do not present these experiments here. Our best heuristic is GRASP-GAR-IA-L1-PR01, where GAR means the reactive version of the greedy method GA, IA stands for the insertion method and PR01 means that it uses path-relinking with elite set size equal to 1. We present here (see Figure 1) just the last comparison among the best heuristic using the GA initial (GRASP-GAR-IA-L1-PR01) method against the best heuristic with GB as initial method to construct solutions (GRASP-GBR-IA-L1-PR01). We can see from Figure 1, that the heuristic GRASP-GAR-IA-L1-PR01 found almost 90% of the best solutions.

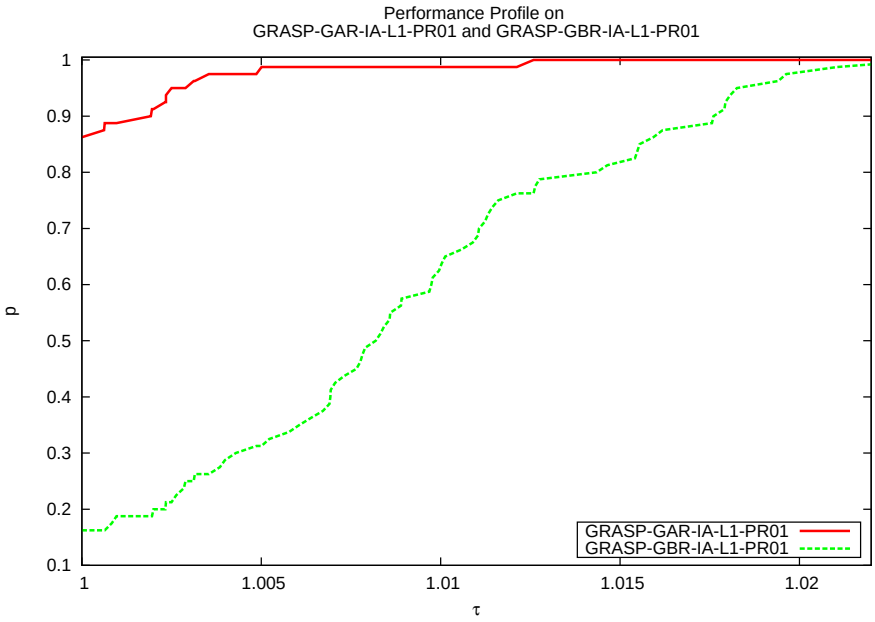

Figure 1: Comparative results between GRASP-GAR-IA-L1-PR01 and GRASP-GBR-IA-L1-PR01.

Our best heuristic uses the method GA to build initial solutions. The difference between GA and GB is small but both of them are much better than GC, which is the only parallel method. Instead of choosing a value for α, which is a parameter used to build the RCL, we chose to use the reactive version, since it has good results and adapts itself for each instance. The insertion method IA outperforms IB, so IA was the chosen method. The path relinking technique contributed a lot to improve the heuristic. The best value we found for the maximum size of the elite set is $\gamma = 1$. Summarizing, our best heuristic is GRASP-GAR-IA-L1-PR01.

An experiment was made to compare our best heuristic, GRASP-GAR-IA-L1-PR01, with the heuristic presented in our previous work in Santos & Xavier (2013). The results are presented in Figure 2, Table 1 and in Table 2. Figure 2 shows the performance profile of these heuristics. Table 1 shows the results of the 40 instances with 500 passengers' requests, while Table 2 shows the results of the instances with size 750. Each table has the objective function value, the number of served requests and the savings made by the passengers in comparison to a sole trip.

Our new heuristic clearly outperforms the previous one, reaching the best results in all instances and being almost two times better in some of them. The main difference between the two heuristics is the path relinking technique, presented only in the new heuristic. Our experiments showed that path relinking had a great impact in the performance. The other difference is the local search. Now we have a new method for the local search, which is an exhaustive method. The new local search only stops when a local optimum is found, while the previous local search has a maximum number of iterations. Besides that, the method to

build the initial solution is different and better, but our experiments showed that it is not responsible for the main gain.

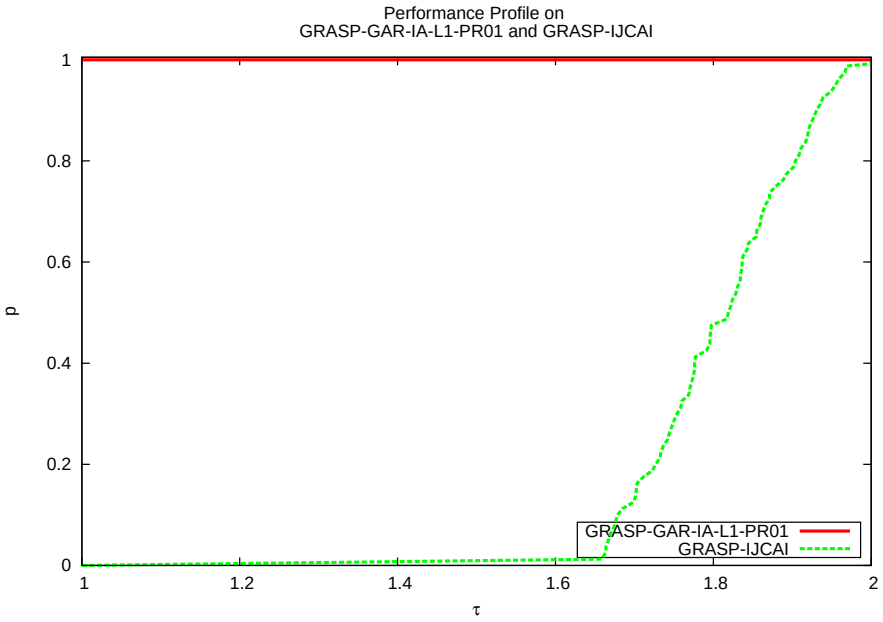

Figure 2: Comparative results between GRASP-GAR-IA-L1-PR01 and our heuristic of the previous work.

	GRASP-GAR-IA-L1-PR01			GRASP-IJCAI		
Instance	Obj	RM	Sav %	Obj	RM	Sav %
sp-500-1.3r-0	**100.2**	**338.3**	**27.8**	54.8	295.0	16.5
sp-500-1.3r-1	**104.2**	**337.7**	**29.3**	56.7	290.3	18.3
sp-500-1.3r-2	**98.3**	**336.1**	**28.5**	51.5	289.6	17.6
sp-500-1.3r-3	**93.3**	**331.4**	**26.9**	48.2	281.3	15.7
sp-500-1.3r-4	**104.1**	**340.3**	**29.6**	58.6	293.2	18.9
sp-500-1.3r-5	**105.6**	**336.5**	**30.6**	58.8	291.0	19.7
sp-500-1.3r-6	**93.6**	**331.9**	**26.8**	52.1	291.4	17.0
sp-500-1.3r-7	**102.3**	**335.7**	**29.2**	56.3	291.9	17.9
sp-500-1.3r-8	**101.4**	**338.8**	**28.2**	52.6	287.9	17.0
sp-500-1.3r-9	**104.5**	**340.5**	**30.2**	56.9	291.3	19.2
sp-500-1.5r-0	**141.3**	**395.5**	**33.7**	79.5	358.3	20.9
sp-500-1.5r-1	**144.8**	**393.9**	**35.7**	81.7	359.1	22.1
sp-500-1.5r-2	**140.4**	**402.3**	**34.3**	76.7	354.2	21.9
sp-500-1.5r-3	**134.8**	**392.6**	**33.2**	73.9	346.2	20.2
sp-500-1.5r-4	**145.4**	**393.4**	**35.8**	84.2	357.5	22.6
sp-500-1.5r-5	**141.1**	**391.7**	**35.7**	79.7	349.5	22.2
sp-500-1.5r-6	**135.0**	**390.7**	**33.2**	72.8	358.3	19.5
sp-500-1.5r-7	**146.7**	**397.2**	**35.6**	83.8	362.7	21.9
sp-500-1.5r-8	**144.5**	**399.0**	**33.9**	78.7	361.0	20.5
sp-500-1.5r-9	**145.5**	**398.5**	**36.1**	83.5	356.7	23.4
sp-500-30a-0	**114.7**	**364.3**	**25.5**	61.3	302.7	15.7
sp-500-30a-1	**127.1**	**369.3**	**28.4**	65.2	303.8	17.2
sp-500-30a-2	**119.9**	**368.5**	**27.2**	63.8	306.0	17.6
sp-500-30a-3	**114.1**	**363.9**	**25.5**	59.1	300.5	16.3
sp-500-30a-4	**120.0**	**359.3**	**28.9**	64.5	299.1	19.2
sp-500-30a-5	**127.4**	**370.9**	**30.6**	67.3	307.2	19.4
sp-500-30a-6	**116.1**	**363.5**	**25.4**	63.0	308.3	17.3
sp-500-30a-7	**113.2**	**352.1**	**25.5**	57.5	292.1	15.3
sp-500-30a-8	**119.8**	**358.0**	**28.1**	63.0	301.5	17.2
sp-500-30a-9	**120.0**	**367.2**	**28.4**	62.6	302.8	18.6
sp-500-45a-0	**152.3**	**411.0**	**31.6**	86.1	352.7	20.9
sp-500-45a-1	**160.4**	**408.9**	**34.0**	91.2	360.9	22.1
sp-500-45a-2	**151.0**	**408.9**	**31.6**	84.1	351.3	20.4
sp-500-45a-3	**147.7**	**398.3**	**32.1**	81.2	345.7	20.7
sp-500-45a-4	**157.7**	**410.1**	**34.3**	89.6	353.1	23.8
sp-500-45a-5	**161.9**	**413.9**	**35.1**	92.3	360.3	23.8
sp-500-45a-6	**147.5**	**401.1**	**31.1**	83.1	355.3	20.6
sp-500-45a-7	**155.7**	**402.0**	**33.0**	83.2	342.1	21.2
sp-500-45a-8	**150.7**	**399.5**	**32.6**	86.4	352.5	21.6
sp-500-45a-9	**157.4**	**410.4**	**34.2**	85.8	350.1	23.0

Table 1: Comparative results between GRASP-GAR-IA-L1-PR01 and the heuristic in Santos & Xavier (2013). For each instance, the objective function value (Obj), the number of requests matched (RM) and the savings in % made by the passengers (Sav), in comparison to a sole trip, are presented. The best results are in bold.

	GRASP-GAR-IA-L1-PR01			GRASP-IJCAI		
Instance	Obj	RM	Sav %	Obj	RM	Sav %
sp-750-1.3r-0	**161.0**	**505.5**	**30.6**	86.3	440.5	19.4
sp-750-1.3r-1	**156.6**	**505.1**	**30.9**	84.9	439.5	18.9
sp-750-1.3r-2	**167.0**	**508.5**	**30.6**	92.9	446.0	19.7
sp-750-1.3r-3	**152.2**	**496.4**	**29.8**	78.5	439.3	16.6
sp-750-1.3r-4	**158.9**	**499.5**	**30.1**	81.1	437.9	17.4
sp-750-1.3r-5	**154.9**	**491.5**	**30.5**	83.5	437.5	18.6
sp-750-1.3r-6	**155.3**	**496.0**	**29.5**	86.7	445.3	18.4
sp-750-1.3r-7	**159.6**	**508.7**	**29.7**	83.1	446.9	17.0
sp-750-1.3r-8	**168.0**	**507.8**	**31.4**	92.2	448.5	19.4
sp-750-1.3r-9	**152.3**	**494.1**	**29.9**	81.9	434.1	17.9
sp-750-1.5r-0	**215.6**	**574.7**	**36.6**	129.6	546.9	23.9
sp-750-1.5r-1	**206.6**	**568.6**	**35.6**	121.4	543.6	21.8
sp-750-1.5r-2	**217.7**	**566.3**	**36.2**	130.6	544.8	22.8
sp-750-1.5r-3	**205.2**	**569.1**	**35.1**	120.8	548.2	21.4
sp-750-1.5r-4	**211.7**	**565.5**	**35.7**	122.2	545.2	21.2
sp-750-1.5r-5	**207.3**	**559.2**	**35.8**	121.7	537.5	21.8
sp-750-1.5r-6	**207.7**	**553.7**	**35.7**	122.0	547.5	21.5
sp-750-1.5r-7	**212.1**	**578.1**	**35.2**	122.1	553.9	20.8
sp-750-1.5r-8	**220.3**	**580.1**	**36.7**	132.4	553.0	23.6
sp-750-1.5r-9	**207.0**	**568.4**	**34.9**	119.4	535.7	21.4
sp-750-30a-0	**172.0**	**522.3**	**27.4**	90.0	446.8	16.8
sp-750-30a-1	**172.9**	**528.5**	**28.4**	87.9	439.4	17.7
sp-750-30a-2	**179.6**	**528.4**	**28.7**	95.1	448.5	17.8
sp-750-30a-3	**171.2**	**525.1**	**27.7**	91.9	450.8	16.9
sp-750-30a-4	**168.3**	**517.7**	**27.2**	86.1	436.1	17.0
sp-750-30a-5	**174.4**	**520.1**	**28.2**	95.1	452.0	18.2
sp-750-30a-6	**176.6**	**524.3**	**28.9**	91.8	451.3	17.5
sp-750-30a-7	**174.6**	**530.7**	**27.0**	91.7	449.5	16.5
sp-750-30a-8	**176.4**	**514.9**	**28.9**	91.9	444.1	17.3
sp-750-30a-9	**170.5**	**524.8**	**27.6**	83.1	431.5	16.1
sp-750-45a-0	**217.4**	**570.9**	**33.2**	131.1	524.3	21.9
sp-750-45a-1	**218.1**	**577.2**	**34.1**	126.6	524.4	21.9
sp-750-45a-2	**223.0**	**577.1**	**33.8**	130.3	521.1	22.0
sp-750-45a-3	**214.9**	**570.7**	**33.1**	128.7	528.4	20.9
sp-750-45a-4	**219.7**	**575.1**	**33.7**	123.7	517.7	21.4
sp-750-45a-5	**219.1**	**572.3**	**34.0**	130.9	528.1	21.9
sp-750-45a-6	**213.3**	**571.0**	**33.4**	126.6	527.1	21.8
sp-750-45a-7	**218.1**	**574.8**	**33.1**	130.1	528.5	21.4
sp-750-45a-8	**218.9**	**564.4**	**34.3**	130.4	520.3	22.3
sp-750-45a-9	**213.9**	**574.6**	**33.0**	122.4	516.7	20.6

Table 2: Comparative results between GRASP-GAR-IA-L1-PR01 and the heuristic in Santos & Xavier (2013). For each instance, the objective function value (Obj), the number of requests matched (RM) and the savings in % made by the passengers (Sav), in comparison to a sole trip, are presented. The best results are in bold.

4.2. Computational Results for the Dynamic Problem

This section shows the results of our proposed method to solve the dynamic DARP-M, described in Section 3. In order to execute the method, we need to set two parameters: the heuristic used to solve a static snapshot of the problem, and the length of the period of time, used to collect requests and to run the heuristic. The heuristic used is GRASP-GAR-IA-L1-PR01, which is our best heuristic, as shown in Section

4.1. We tried three different lengths for the period: 150, 300 and 600 seconds.

We built 4 instances for the dynamic version of the problem, each one represents a 12 hour period of taxi activity in São Paulo. The graph of São Paulo, used to create the instances for the static problem, was also used as the graph G for all 4 instances. The dynamic set $\mathbb{M}$ of vehicles is the same for all instances. It starts with 1333 vehicles, and in every hour 1333 new vehicles arrive, each one having a random source and destination. Each vehicle is available for 4 hours. The dynamic set $\mathbb{N}$ of requests is also the same for all instances. Every minute 54 new requests arrive, with random source and destination. The difference between each instance is the maximum delay allowed, which is used to compute the latest arrival time for the requests. There are instances with maximum delay of 30 and 45 minutes and also with delays proportional to the travel time, with multipliers of 1.3 and 1.5. The travel cost, time window, maximum capacity, number of passengers per request, duration of the service are defined in the same way as in the static instances.

The use of an efficient method to compute shortest paths is crucial. In the beginning of each period, a distance table must be computed, which can be very large. We made a preliminary set of experiments that showed that the Dijkstra algorithm took, on average, 448.67 seconds to build a 1000×1000 distance table considering our graph, while the Contraction Hierarchy algorithm took only 0.01 seconds. In order to have a low query time, the Contraction Hierarchy algorithm needs to preprocess the whole graph, but it must be done just once, assuming that the graph does not change. Nonetheless the pre-processing phase took only 60 seconds. So we used the Contraction Hierarchy algorithm in our method to solve the problem.

The final result of each instance, and for each length of period of time is shown in Table 3.

	150 seconds				300 seconds				600 seconds			
Instance	Obj	RM	Sav %	SR	Obj	RM	Sav %	SR	OBJ	RM	Sav %	SR
big-1.3r	5426.4	**32513**	18.0%	13596	5537.2	32165	18.3%	**14230**	**5548.4**	30712	**18.8%**	14140
big-1.5r	9707.4	**35929**	27.8%	23320	**9869.7**	35791	**28.2%**	24490	9774.7	35034	**28.2%**	**24536**
big-30a	**7702.5**	**33494**	**18.4%**	20090	7661.4	33321	18.2%	**20573**	7601.9	33044	18.0%	20155
big-45a	11492.1	**35301**	27.7%	26391	**11540.2**	35165	**27.8%**	**27245**	11050.4	34941	26.9%	27123

Table 3: Final results of the 4 big instances that simulates São Paulo. For each instance and for each length of period, the table shows the objective function value (Obj), the number of requests matched (out of 38880) (RM), the savings made by the passengers (Sav) and the number of passengers that had a shared ride (SR). The best results are in bold.

There is a tradeoff when the length of the period is increased. The length of the period is the maximum allowed time the heuristic can execute, so when it increases, higher is the chance of getting better solutions for the static problem. However, with a large period, requests can be missed, since it can take an entire period to start considering some requests. When the length of the period was 150 seconds, it had the greatest number of requests served, in all instances, but it had the best objective function in only one of them. When the length of the period was 300 seconds it was obtained the best results, having the best objective value in 2 out of the 4 instances. We can conclude that using 300 seconds as the length of the period is a good

choice, because 300 seconds is enough for the heuristic to generate good solutions for static instances with up to 1500 requests. Also, it is not too long to have an influence in the time windows of the passengers.

Our results also show that the savings can be almost 30%, when passengers allow a larger maximum delay. This number does not indicate the real savings that passengers of São Paulo can made. However, we believe that the real savings can be even higher. In our instances, requests and vehicles are randomly spread throughout the city, but in a real situation there are some patterns of taxi mobility: for example moving from home places to workplaces areas and between workplaces areas, see Peng et al. (2012). These patterns can lead to requests that facilitate the ride-sharing.

5. Conclusions

In this work we defined a dynamic Dial-a-Ride problem denoted by DARP-M and proposed a method to solve it. The method needs an heuristic for the static version of the problem and an algorithm to solve the many to many shortest path problem. We developed GRASP heuristics and used a very efficient algorithm to compute shortest paths, based on Contraction Hierarchies. We also created instances, using real data, for the static and the dynamic problem.

The Dial-a-Ride Problem with Money as an Incentive is more general and more realistic than other problems presented in the literature, since it has all basic set of constraints and it also allows the passengers to choose the maximum value they are willing to pay for the ride. This new constraint is important because passengers need to have a benefit in order to accept a ride-sharing service. Because of the new constraint, shared costs must be computed while evaluating solutions, which make the problem harder.

DARP-M can be incorporated inside the server side of any framework that provides ride-sharing. Our proposed method can effectively solve it, even in large scale, since our experiments were made with a real number of taxis and passenger's requests.

Our best GRASP heuristic had better results than our previous heuristic in Santos & Xavier (2013), mainly because the path relinking, which is a very powerful technique.

As future research topics new heuristics could be developed and tested against ours which is based on GRASP. Heuristics based on Large Neighborhood Search obtained good results for the Pickup and Delivery Problem, see Pisinger & Ropke (2010), and maybe reach good results when adapted for the DARP-M. Another interesting problem is to develop efficient algorithms to determine the best place to insert a new request (i^+, i^-) in a given route considering the restrictions of DARP-M. In our heuristic we used a brute force approach since we basically try all possible pairs of locations where the request can be placed. Besides that, efficient algorithms to recalculate the route parameters, and feasibility check could be developed as the work in Masson et al. (2013) for the Pick-up and Delivery problem. Finally another type of cost sharing rule maybe implemented in the algorithm. Our cost sharing rule divides the cost of each part of the route evenly among the passengers in the car. In Agatz et al. (2011), the costs of the joint trip are allocated in a manner that is proportional to the distances of the separate trips. It could also be investigated the design

of truthful mechanisms in order for both passengers and car owners reveal the truthful private amount of money they are willing to pay or receive in shared trips.

Acknowledgements.

This research was partly supported by CNPq and FAPESP.

References

Agatz, N., Erera, A., Savelsbergh, M., & Wang, X. (2012). Optimization for dynamic ride-sharing: A review. *European Journal of Operational Research*, *223*, 295–303.

Agatz, N. A., Erera, A. L., Savelsbergh, M. W., & Wang, X. (2011). Dynamic ride-sharing: A simulation study in metro atlanta.(report). *Transportation Research Part B: Methodological*, *45*, 1450(15).

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

Cordeau, J.-F., & Laporte, G. (2007). The dial-a-ride problem: models and algorithms. *Annals OR*, *153*, 29–46.

Dolan, E. D., & Moré, J. J. (2002). Benchmarking optimization software with performance profiles. *Mathematical Programming*, *91*, 201–213.

Feo, T. A., & Resende, M. G. C. (1995). Greedy randomized adaptive search procedures. *Journal of Global Optimization*, *6*, 109–133.

Geisberger, R., Sanders, P., Schultes, D., & Delling, D. (2008). Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In C. McGeoch (Ed.), *Experimental Algorithms* (pp. 319–333). volume 5038 of *Lecture Notes in Computer Science*.

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

Horn, M. E. (2002). Fleet scheduling and dispatching for demand-responsive passenger services. *Transportation Research Part C: Emerging Technologies*, *10*, 35 – 63.

Knopp, S., Sanders, P., Schultes, D., Schulz, F., & Wagner, D. (2007). Computing many-to-many shortest paths using highway hierarchies. In *Proceedings of the Nine Workshop on Algorithm Engineering and Experiments, ALENEX 2007, New Orleans, Louisiana, USA, January 6, 2007* (pp. 36–45). volume 7.

Lalos, P., Korres, A., Datsikas, C., Tombras, G., & Peppas, K. (2009). A framework for dynamic car and taxi pools with the use of positioning systems. In *Computation World: Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns, 2009. COMPUTATIONWORLD '09.* (pp. 385 – 391).

Ma, S., Zheng, Y., & Wolfson, O. (2013). T-share: A large-scale dynamic taxi ridesharing service. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)* (pp. 410–421).

Masson, R., Lehuédé, F., & Péton, O. (2013). Efficient feasibility testing for request insertion in the pickup and delivery problem with transfers. *Operations Research Letters*, *41*, 211–215.

Parragh, S., Doerner, K., & Hartl, R. (2008a). A survey on pickup and delivery problems. *Journal für Betriebswirtschaft*, *58*, 21–51.

Parragh, S., Doerner, K., & Hartl, R. (2008b). A survey on pickup and delivery problems part 2. *Journal für Betriebswirtschaft*, *58*, 81–117.

Peng, C., Jin, X., Wong, K.-C., Shi, M., & Li, P. (2012). Collective human mobility pattern from taxi trips in urban area. *PLoS ONE*, *7*, e34487.

Pisinger, D., & Ropke, S. (2010). Large neighborhood search. In M. Gendreau, & J.-Y. Potvin (Eds.), *Handbook of Metaheuristics* (pp. 399–419). volume 146 of *International Series in Operations Research and Management Science*.

Santos, D., & Xavier, E. (2013). Dynamic taxi and ridesharing: A framework and heuristics for the optimization problem. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence* IJCAI 2013 (pp. 2885–2891).

Schultes, D. (2008). Route planning in road networks. In *Ausgezeichnete Informatikdissertationen* (pp. 271–280).

Sims, R., Schaeffer, R. et al. (2014). *Climate Change 2014: Mitigation of Climate Change*.

Tao, C.-C. (2007). Dynamic taxi-sharing service using intelligent transportation system technologies. In *International Conference on Wireless Communications, Networking and Mobile Computing, 2007. WiCom 2007.* (pp. 3209 –3212).