Dynamic Taxi and Ridesharing: A Framework and Heuristics for the Optimization Problem *

Douglas O. Santos and **Eduardo C. Xavier**
Institute of Computing, University of Campinas, Campinas, Brazil
douglasantos00@gmail.com
eduardo@ic.unicamp.br

Abstract

In this paper we study a dynamic problem of ridesharing and taxi sharing with time windows. We consider a scenario where people needing a taxi or interested in getting a ride use a phone app to designate their source and destination points in a city, as well others restrictions (such as maximum allowable time to be at the destination). On the other hand, we have taxis and people interested in giving a ride, with their current positions and also some constraints (vehicle capacity, destination, maximum time to destination). We want to maximize the number of shared trips: in the case of taxis, people going to close locations can share the costs of the trip, and in case of rides, the driver and passengers can share costs as well. This problem is dynamic since new calls for taxis or calls for rides arrive on demand. This gives rise to an optimization problem which we prove to be NP-Hard. We then propose heuristics to deal with it. We focus on the taxi sharing problem, but we show that our model is easily extendable to model the ridesharing situation or even a situation where there are both taxis and car owners. In addition, we present a framework that consists basically of a client application and a server. The last one processes all incoming information in order to match vehicles to passengers requests. The entire system can be used by taxi companies and riders in a way to reduce the traffic in the cities and to reduce the emission of greenhouse gases.

1 Introduction

In this work we propose a framework that can be used by taxi companies to provide taxi sharing among their users. The same framework can be used by users that are interested in sharing their trips, and in this case, the system can be used with payment as an incentive to people with car to accept rides in their car. The framework consists basically of a client application and a server that processes all incoming information in order to match routes that can reduce costs of taxis users, by doing taxi sharing and splitting the trip costs, and by matching users interested in selling trips with users wanting to buy a cheap trip. In the case of the taxi sharing system, a person uses the cell app (or a regular phone call to a taxi company) specifying its current location, destination, and others informations such as a maximum delay that he is available to accept (the delay is in comparison with a sole taxi trip). If a user gets a shared trip, the system must guarantee that it is cheaper than the trip where he goes alone, and that he arrives at his destination within the maximum delay specified. In the case of ridesharing, car owners and riders specify their origins and destinations. Car owners also indicate a maximum delay they can accept. The riders input how much they are willing to pay for the trip. Based on the car gas price and consumption per kilometer, the system computes matchings between car owners and riders, such that on each shared trip, it is advantageous for the driver to give the ride based on the amount of money to be received and the maximum delay constraint.

The system is easy to use. For a taxi user, there is no difference from the actual system. Users have incentives to use the system since they are going to save money, in case of taxi users and riders, and possible earn some money, in case of car owners.

According to the 2005 IPCC report [Metz *et al.*, 2005], the transportation sector is responsible for about 16% of CO_2 emissions being the second largest producer, behind the Energy sector, but emissions from the transportation sector are growing faster than the energy sector. Hence, is mandatory to optimize the use of fossil fuels and our framework can be used as a small part of the effort. Interestingly, to maximize the number of shared trips gives rise to an interesting optimization problem which we prove to be NP-Hard.

In this paper we will focus principally on the optimization problem that we call dynamic taxi sharing with time windows problem (TSTWP). It is to assign taxis vehicles to passengers requests, which are spread over different locations in a given city. Several constraints must be considered: the vehicle capacity , the cost of a shared ride must not exceed the cost of a private ride, each passenger and taxi has time window, so the problem also consists of assign a time for each pickup and delivery event, within these time windows. We want to maximize the number of served requests and minimize the total cost incurred to passengers, which is the same as maximize the sharing. Although we focus on the taxi sharing problem,

*This work was supported by CNPq.

we show that our model is easily extendable to model the ridesharing situation or even a situation where there are both taxis and car owners who want to share their vehicles to reduce costs.

We show that the TSTWP problem is NP-hard and present a method to solve the dynamic problem using a GRASP heuristic to solve static snapshots of the problem. We used the proposed heuristics to solve instances using real world datasets provided by [Herbawi and Weber, 2012], which were extracted from a travel and activity survey for northeastern Illinois conducted by Chicago Metropolitan Agency for Planning. We also tested our heuristic with instances that simulate a metropolis, like Sao Paulo.

There are several studies about ridesharing in the literature, but we are interested in the ones related to the optimization problem known as Dial a Ride Problem (DARP), [Cordeau and Laporte, 2007]. To the best of our knowledge, there is no previous work on the dynamic taxi sharing with time windows problem. A similar problem was investigated in [Herbawi and Weber, 2012]. They presented a genetic and an insertion heuristic to solve a dynamic ride-matching problem with time windows. The main difference is that we consider shared costs, having a constraint that does not allow people to pay more on a shared trip than they would pay on a sole trip, and also, our objective function includes minimizing the costs to be payed by the passengers. This constraint is important in a real life situation. In [Agatz *et al.*, 2011], another kind of dynamic ridesharing 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.

Frameworks for taxi sharing and ridesharing were presented in [Tao, 2007] and in [Lalos *et al.*, 2009]. In these previous works, just framework models were proposed without emphasis on how to compute route matching between users. They used simple greedy heuristics and do not consider complicated constraints. Also the money incentive is not taken into account, since they do no try to minimize the costs. But interesting safety restrictions are presented such as, to use the cell phone GPS to check if a car is doing a precomputed route. In our framework we use social networks as a safety incentive to ride-sharing. Users can specify that they will share a ride only with friends of friends, for example.

In Section 2 we briefly describe our proposed framework for the problem. In section 3 we give a formal description of the optimization problem that needs to be solved in a server side according to the proposed framework. In section 4 we proof that the TSTWP is NP-hard. Section 5 describes our proposed method to solve this optimization problem and in Section 6 we present experimental results of it. ~~Section 7 draws some conclusions~~.

2 The Framework

In this section we describe a system that, on one side, is going to be used by taxi companies and car owners interested in offering a ride, and on the other side is going to have taxi's users and people interested in a ride (we call these last ones by hitchhiker, although they usually pay for the ride).

The system is very simple and consists of two components: (1) A cell app used by taxi passengers to specify source and destination, time to be served and maximum time to be delivered. In case of hitchhikers, they also need to input how much they would be willing to pay for the ride. The app is also used by car owners that specify, source, destination, leaving time and maximum allowable time to be at destination. (2) A server that receives all information on demand and tries to match taxis to multiple taxis requests, and hitchhikers to car owners.

The hard part is to construct good solutions that satisfy several requirements: **Time:** No one can arrive at its destination later than its maximum allowable time, and no one is served before its service time. **Costs:** A taxi passenger on a shared trip can not spend more money than on a sole trip. A car owner interested in offering rides, must receive at least as much he spend doing an alternative route, instead of his shortest path route. This can be easily computed by considering how much kilometers the driver diverted from his original route, the gas price, and the car consumption per kilometer. This can be done by the server which then informs the driver if it is advantageous to give the ride. **Safety:** The system used by hitchhikers and car owners requires a previous registration. Users can specify that they only ride with friends of friends for example, using data from social networks. Also, others safeguards can be used such as the track of the route using GPS data as proposed in [Lalos *et al.*, 2009].

The system can also use virtual payments for taxi drivers as well car owners. In this last case a hitchhiker must buy first some online credit that is later used to pay for the rides. The system then intermediate the payment between car owners and passengers.

With this description we have a good idea of how the system works, and the incentives for people using it: taxi users may have cheaper trips if the system finds routes that can be shared. Otherwise, the taxi user will have a normal sole trip. For car owners there is an incentive to reduce his costs of the trip, by being paid to deliver someone near to his original route. The most challenging part of the system is how to compute good matchings. We describe this problem formally in the next section.

3 Formal Problem Definition

In this section we give a formal description of the TSTWP problem. We also show that the formulation of the problem can also be used to deal with ridesharing in general. At any moment we will have a set of users wanting a taxi trip (or a ride) and a set of taxis (or car owners) that can serve requests. We want to match riders to taxis (or car owners) in order to maximize the number of shared trips.

In the TSTWP we are given a graph $G = (V, E)$, which represents the map of the city, a set $N = \{1, ..., n\}$, representing n taxis requests, and a set $K = \{2n, ..., 2n + ta\}$, representing ta taxis vehicles. The set of vertices V, represents all the locations of the given city, and the set E of edges represents the streets that connect these locations. Each request $i \in N$ has a source point i and a destination point $i+n$,

which are vertices of G. It also has an earliest departure time e_i from point i, a latest arrival time l_i to point $i+n$ and the number of people p_i that will use the service at that point. Each taxi vehicle $k \in K$ has a source point k, which is a vertex of G, the vehicle capacity ca_k, the start time e_k and the end time l_k, that represents the period in which the taxi will be available. We also know the travel time t_{ij} and the travel cost c_{ij} between every pair of vertices i, j.

On a shared trip, the cost of each part of the trip is shared among all the passengers who were in the vehicle on that part.

We define a taxi route as a sequence of pickup and delivery points. We always use the shortest path to go from a point to its consecutive point in a route. The first point of the route is always a taxi source point.

The problem is to find a route for each taxi and to assign a service time for every point of the route, subject to the following constraints:

1. Each served passenger must be picked up at the specified source point, after its earliest departure time and must be delivered at the specified destination point, before its latest arrival time. Also, each taxi vehicle must only serve after its start time and before its end time.
2. Each vehicle cannot load more passengers than its capacity.
3. The total cost to be paid by a passenger on a shared trip must be less than the cost of a private trip.
4. Every passenger of a same request must travel together, in the same vehicle.

We also want to optimize an objective function, which consists of the following conflicting criteria:

1. Maximize the number of served requests.
2. Minimize the cost payed by the served passengers.

We have an upper bound for the second objective, by computing the cost of each trip as a private trip. This way in order to minimize the cost of served passengers, shared trips must be used. Thats why this second objective cost is equivalent to maximize the number of shared trips.

In order to optimize these two possible conflicting objectives we propose the following objective function: Let x_i^k be a binary variable which indicates if request i was served by taxi k. And, let C_i^k be the cost payed by the passengers of the request i to the taxi k. We want to maximize: $\sum_{k \in K} \sum_{i \in N} x_i^k - \alpha \frac{C_i^k}{c_{i,i+n}}$. Notice that $c_{i,i+n}$ represents the cost of a private trip for request i and since $C_i^k \leq c_{i,i+n}$ the fraction $\frac{C_i^k}{c_{i,i+n}}$ is always smaller than or equal to 1. The smaller the fraction, the larger is the economy of the users of request i. We have α as a constant in $[0, 1]$ which is used to weight more or less the second criteria.

The proposed formulation of the problem can also be extended to model a ridesharing situation or even a situation where there are taxis and car owners who want to share their vehicles to reduce trip costs. If the vehicle is not a taxi, we only need to add a destination point, the remaining problem remains the same.

4 Proof of NP-Hardness

In this section we show that the TSTWP is NP-Hard by reducing the metric hamiltonian path problem to it. An instance of the metric hamiltonian path problem consists of a complete graph $G = (V, E)$ with cost w_{ij} for each edge (i, j), satisfying the triangle inequality, and a vertex u where the hamiltonian path must start. The metric hamiltonian problem is to find a hamiltonian path in G starting at a given vertex u with minimum cost. The decision version of the problem is to verify if there is a hamiltonian path starting at u with cost less than or equal to T. The decision version of the TSTWP is to verify if there is a feasible solution with objective function greater than or equal to Z.

Let $I = (G, u, T)$ be an instance of the hamiltonian path problem. An instance J for the taxi sharing problem will be created from I. The graph that represents the city will be the same graph G from I. The travel time t_{ij} of each edge (i, j) will be w_{ij}, and the travel cost c_{ij} will be zero. There will be only one taxi k in the set K, with source point u, $e_k = 0$, $l_k = T$ and $ca_k = 1$. J will have n requests, where $n = |V|$. For each vertex $v \in V(G)$ there will be a request i with source point and destination point v, and also, $e_i = 0$, $l_i = T$, $p_i = 1$. Finally, $Z = n$, meaning that we want to serve all requests. We prove this is a correct reduction below.

Theorem 1. *There is a hamiltonian path from u, in I, with cost less than or equal to T, if and only if, there is a feasible solution for the dynamic taxi sharing problem with time windows, in J, with objective function greater than or equal to n.*

Proof. $\Rightarrow$ Let P be a hamiltonian path starting in u. This path can be used as the route of the taxi k. At every vertex the taxi will pickup and delivery the passenger. Because the cost of P is less than or equal to T, the total travel time of the taxi will not exceed l_k. Moreover, all vertices will be visited, because P is a hamiltonian path, so all requests will be served.

$\Leftarrow$ Let R be the route of the taxi k in a feasible solution that serves all n requests. We have that R visits every vertex of G, and the sum of the costs of the edges in R is less than or equal to T, because $l_k = T$. So R can be viewed as a path in G that visits every vertex and has cost not greater than T, but R can have repeated vertices. We can eliminate this repetition using shortcuts. If b is a vertex between a and c and b has already been visited, we can modify the path by adding a shortcut from a to c. Shortcuts will never increase the cost, because G satisfies the triangle inequality. So $w_{ac} \leq w_{ab} + w_{bc}$. At the end of this procedure we will have a hamiltonian path in I with cost at most T.

□

5 The proposed method

The TSTWP is an online problem, because new requests and vehicles can arrive at any moment. So, we divide the day into a set of short time periods. In each period we solve a static version of the problem using the GRASP heuristic (described later in this section), taking into account all known requests and vehicles. The requests that were not served are added in the input of the next execution. Also, the taxis routes formed

on the previous execution can be updated on the new execution, but we have to take care because some events may have already occurred.

The time period must not be too long, as some passengers' time window would become unfeasible. And, of course, it must not be shorter than the execution time of the GRASP heuristic. In our experiments the execution time of the heuristic seemed to be short enough to not lose any request, and also, we can set a time limit, in extreme cases.

The GRASP heuristic: We proposed a greedy randomized adaptive search procedure (GRASP) [Resende, 1995] to solve the static version of the TSTWP. It has two phases: In the first phase we have to compute a greedy randomized initial solution. 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. The GRASP metaheuristic repeats these two phases for a maximum number of iterations (or a maximum time).

Solution Representation: A feasible solution contains a feasible route for each taxi. We define a route as a sequence of points (pickup or delivery) in the order that they will be visited by the taxi. A feasible route is a route that satisfies all the problem constraints.

Each point i in the route of a taxi k has a service time T_i^k, that represents the time immediately after the taxi served the point i. The service time is computed by considering the time of the trip from the taxi starting point until point i in the given route. Remember that for any two points (i, j) in the route we have a travel time t_{ij}. Each point i also has a load L_i^k, representing the number of people in the taxi after the service at point i. Moreover, each request i that was served by the taxi k has a total cost C_i^k to be paid.

Computing Initial Solutions: First of all, we initialize the route of each taxi. We add the source point k of each taxi as the first point of its route and set T_k^k as the maximum between e_k and the time at the moment. We set $L_k^k = 0$, because the taxi driver is not counted as a passenger.

Note that some taxis may already have a route, because of a previous execution of the heuristic. In this case, besides of initialize their route, we also select the first point in the route with a service time greater than the time at the moment, and add, in the new route, this point and all points that come after it in the old route. This way, we can improve the route formed in a previous execution.

After this initialization, the following procedure is performed: First, a random taxi k is chosen from K. Then, for all requests $i \in N$ that were not served yet, we calculate a greedy function (described in the next section) μ_i^k. After that, a request i is chosen randomly with probability proportional to μ_i^k. We try to insert the source and destination points of the request i in the route of the taxi k (the insertion method is described later). If the insertion is feasible, we remove this request from N, and we update the greedy function for all remaining unserved requests in N. If the insertion is unfeasible we do not try to insert this request anymore in the route of this taxi. Then, another request is chosen in the same manner. After we tried to insert all requests, we remove k from K and the procedure is repeated, until $K = \emptyset$.

The feasibility of the route is checked in the insertion method, so the initial solution is always feasible.

Greedy Function: When considering to insert a request i in a feasible route, we compute a greedy function to evaluate how well would request i fit in the route.

We define the delay time $d_{j,i}^k$ of a point i as the amount of time that would be increased if point i had been inserted after point j, in the route of taxi k. So, for a source point i we have $d_{j,i}^k = max\{t_{ji}, e_i - T_j^k\}$. For a destination point $i + n$ we have $d_{j,i+n}^k = t_{j,i+n}$ if $t_{j,i+n} + T_j^k \leq l_i$, and $d_{j,i+n}^k = \infty$ otherwise, meaning that the solution would become unfeasible.

For a request i and a taxi k we define the greedy function $\mu_i^k = min_{j \in R(k)} d_{j,i}^k + min_{j \in R(k)} d_{j,i+n}^k$, where $R(k)$ is the set containing all current points in the route of taxi k.

Insertion Method: Now we present the insertion method that is used to create initial solutions.

Given a feasible route $R(k)$ of a taxi k, we want to insert the new points i and $i + n$, in a way, that i is situated before $i + n$, and the route must remain feasible.

First of all, we create a set S with the first β points j of $R(k)$ with the minimum delay time $d_{j,i}^k$ in relation with the source point i. These β points are the most favorable places to insert the source point i, but we can not ensure that the optimum point is among them.

For each pair of points (a, b), where $a \in S$ and $b \in R(k)$ and a comes before b in $R(k)$, we try to insert i immediately after a and $i + n$ immediately after b. If the insertion is feasible, we mark (a, b) as a feasible pair associated with the new value of the objective function, since the insertions may cause a change in this value. Note that the bigger β, more points will be considered, increasing the chance of an optimum insertion. But it also increases the execution time, so β is a constant which determines if the insertion method will be an exhaustive search or an heuristic.

We do the same procedure with the destination point: create a set S with the first β points of $R(k)$ with the minimum delay time, but this time, in relation with the destination point $i + n$. Then we consider every pair of points (a, b) where $a \in R(k)$, $b \in S$ and a comes before b in $R(k)$. For each such pair we insert $i + n$ immediately after b and insert i immediately after a and check the feasibility of the route. If the route remains feasible, we mark (a, b) as feasible.

After that, a pair (a, b) is randomly selected, among all feasible pairs, with probability proportional to the value of the objective function associated with it. The insertion is then performed considering this selected pair of points.

Now, we describe how the insertion of the points i and $i+n$ is performed in $R(k)$ considering the pair (a, b).

Note that the values of T_j^k and L_j^k do not change for all $j \leq a$. The points that come after a will be inserted again in the same order, including the points i and $i + n$. To insert a source point j immediately after a point $j - 1$, we have that $T_j^k = max\{T_{j-1}^k + t_{j-1,j}, e_j\}$ and $L_j^k = L_{j-1}^k + p_j$. We need to verify if $L_j^k \leq ca^k$ and if $T_j^k \leq l_k$. To insert a destination point j immediately after a point $j - 1$, we have $T_j^k = T_{j-1}^k + t_{j-1,j}$ and $L_j^k = L_{j-1}^k - p_j$. Also, it is necessary to verify if $T_j^k \leq l_j$ and if $T_j^k \leq l_k$. Finally, C_i^k must be

recalculated for all requests i served by the taxi k. Therewith, we can verify if the cost constraint of the problem is satisfied for all requests served by k, and can calculate the new objective function value. Let $a_1, a_2, ..., a_x$ be the points visited in order by the passengers of request i, where $a_1 = i$ and $a_x = i + n$. We have that $C_i^k = \sum_{j=1}^{x-1} \frac{c_{(a_j, a_{j+1})} p_i}{L_{a_j}^k}$.

Local Search: The local search method starts setting as the current solution an initial feasible solution, and explore new feasible solutions that are in its neighborhood. Solutions in the neighborhood are obtained by performing some modifications in the current solution. We propose three types of operations to obtain new solutions:

- Removal of some request from some route and attempt to insert, in this route, some request that was not served.
- Permutation of two requests from different routes.
- Permutation of two consecutive points of the same route.

In the first operation, we randomly select a served request i and an unserved request j. Let k be the taxi that attends i. We remove i from the route of k and make an attempt to insert j in this route. The insertion method here is quite different from the insertion method used when creating initial solutions. Instead of choosing the first β points to consider an insertion, we try to insert the source and destination points of j considering all pair of points in the route. We then choose the feasible insertion with the best improvement in the objective function value. The removal method of some request i is the same as the reinsertion of all points of the route, in the same order, except i and $i + n$, which will be removed.

In the second operation, we randomly select two served requests i served by taxi k_i and j served by taxi k_j. First, we remove i from k_i and j from k_j, then we make an attempt to insert the request i in the route of k_j and j in the route of k_i. The insertion and removal method are the same used in the first operation.

In the last operation, we first select randomly some taxi k. After that, we randomly select a point i of this route, that is not the last point nor the first point. Then, we change the order of the point i with its successor. In order to do this, we insert again all the points of the route in the same order, except the two points, which will be permuted.

In our local search procedure, at each iteration, we perform each of the three operations defined above, obtaining three new solutions. If the best of them is better than the current one, then the method sets this solution as the new current one. This procedure is repeated until there is no improvement in the current solution after a certain number of consecutive iterations, or until some maximum number of iterations is achieved.

6 Experimental Results

In this section we provide experimental results using the proposed heuristic and three variations of it. The proposed heuristic is denoted by G-IA-L, where G means that we use a greedy function, IA means that we use the insertion method A (the one described in Section 5) and L means that local search is performed. The modified heuristics are denoted by G-IB-L, NoG-IA-L and G-IA-NoL. The insertion method B is a modification of the insertion method A. Instead of choosing the first β points to consider an insertion, we try to insert the source and destination points of some request considering all pair of points in the route. We then select a feasible insertion randomly with probability proportional to the new objective function associated with each insertion. The term NoG means that we do not use the greedy function, in this case, the requests are selected randomly. And NoL means that we do not perform local search. This way we can verify the effectiveness of the proposed greedy function, the insertion method and the use of local search.

	Objective Function			
Instances	G-IA-L	G-IB-L	G-IA-NoL	NoG-IA-L
inst-698-1-30	**14.0**	11.6	13.6	12.5
inst-698-5-30	**35.1**	29.4	34.1	31.1
inst-698-10-30	**49.2**	41.3	48.7	43.5
inst-698-20-30	**64.8**	56.4	64.6	57.9
inst-698-30-30	**71.3**	64.0	70.6	65.0
inst-698-50-30	**72.2**	64.0	71.4	65.6
inst-698-1-120	**18.6**	16.9	12.8	17.0
inst-698-5-120	**39.0**	33.4	31.3	33.9
inst-698-10-120	**51.4**	42.2	45.7	42.6
inst-698-20-120	**67.1**	56.0	63.2	56.0
inst-698-30-120	**71.8**	61.7	68.1	62.5
inst-698-50-120	68.5	57.1	**68.8**	59.1
inst-744-1-30	**6.9**	5.9	6.3	6.1
inst-744-5-30	**16.3**	13.5	15.6	13.6
inst-744-10-30	25.0	21.4	**25.1**	19.8
inst-744-20-30	**39.4**	34.9	39.0	30.8
inst-744-30-30	**49.9**	45.7	49.5	40.9
inst-744-50-30	62.2	57.7	**62.4**	52.3
inst-744-1-120	**10.5**	9.5	6.5	9.7
inst-744-5-120	**18.9**	16.9	13.9	16.8
inst-744-10-120	**26.3**	22.2	22.2	21.9
inst-744-20-120	**39.3**	31.9	35.5	31.3
inst-744-30-120	**49.7**	42.1	46.8	38.4
inst-744-50-120	**57.9**	50.7	56.5	49.2

Table 1: The objective function value of the four heuristics for the 24 instances. The best result of each instance is in bold.

The experiments were done on an Intel Xeon X3430, 2.4Ghz, with 4 cores and 8Gb of memory, using the Linux (Ubuntu) operation system. We used the programming language C++ and the compiler g++ 4.4.3 to implement the heuristics.

In all experiments we set $\alpha = 0.99$ and $\beta = 5$. Instead of setting a maximum number of iterations for the heuristics, we set a time limit of 5 minutes. For each iteration, the heuristics executed at most 250 iterations of the local search, but it is interrupted if there is no improvement in 50 consecutive iterations. For each instance we executed each heuristic 30 times, and we took the average of the results.

We used 24 instances provided by [Herbawi and Weber, 2012], which were extracted from a travel survey for northeastern Illinois conducted by CMAP. There were two different travel patterns, one with 698 requests and the other with 744 requests. All destination and source points were given in latitude and longitude coordinates, so we used the Haversin formula [Sinnott, 1984] to measure the distance between two pair of points. The travel time between two pair of points was calculated considering a vehicle speed of 60km/h and

the travel cost was set as ten times the travel time. The earliest departure time was between 10:00 and 16:40 for all requests of the first pattern and between 00:30 and 23:30 for all requests of the second one. The latest arrival time of each request was set as the earliest departure time, plus the travel time between the source and the destination, plus a maximum delay. For each pattern, 12 instances were created varying the maximum delay and the number of taxis. Half of this 12 instances had maximum delay of 30 minutes while the other half had 120 minutes of maximum delay. And for each of the 6, we had different numbers of taxis, 1, 5, 10, 20, 30 and 50. All vehicles had a capacity of 4 people and all requests were just for one person. The source of each taxi was generated randomly in the limits of the requests' points. We call each instance as *inst-n-ta-md*, where n is the number of requests, md the maximum delay and ta the number of taxis.

We also created 4 instances, simulating the city of Sao Paulo. Sao Paulo has $1528km^2$ with 78 thousands of taxis requests per day, giving an average of 54 requests per minute [1]. We also know that a typical square of this city has side of 100m. To represent this, for each instance, we created a 500x300 grid where each block of the grid had a side of 100m. We put 540 taxis requests, representing a period of 10 minutes, and 200 taxis vehicles spread randomly over the grid. We set the maximum delay as 30 minutes and all requests had the same earliest departure time. The distance between two points was measured using the Manhattan's distance, the travel time, the travel cost and the latest arrival time were calculated as described above. We call each instance as *grid-540-200-30-x*, where x is a number between 0 and 4.

Table 1 shows the results comparing the four heuristics (results in bold are the best ones). As we can see, the proposed heuristic gives better results in almost every instance. The insertion method A seems better than the insertion method B, because it is faster so we can do more iterations and also it is more greedy and less random since we select just the β best points. The method B can select bad feasible insertions even though the probability is low. The local search makes big difference in some instances but not in all of them. The main reason is that without local search the heuristic gets faster so it can execute more iterations and also, the local search is kind of random, so sometimes it will not make good improvements. Our greedy function seems to increase significantly the results because without it, although the heuristic gets faster and can do more iterations, the results are not as good. We made tests with others modifications of the heuristic but these ones were the most relevant.

Table 2 shows the results of our proposed method. The economy is calculated as the percentage of the cost saved by the requests served per the cost they would pay if traveling in a sole taxi. As we can see, the economy varies from around 10% through 40% with an average of 18.58%. The proposed method seems to be an effective way to compute shared routes bringing significative economy to users on a shared trip.

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

Instance	OF	RS	Eco	Ite-Time (s)
inst-698-1-30	14.0	61.3	29.0%	1.9
inst-698-5-30	35.1	247.9	18.9%	3.2
inst-698-10-30	49.2	416.2	15.1%	3.0
inst-698-20-30	64.8	611.8	13.5%	2.9
inst-698-30-30	71.3	684.4	12.9%	2.8
inst-698-50-30	72.2	697.8	12.8%	2.7
inst-698-1-120	18.6	64.6	24.0%	3.8
inst-698-5-120	39.0	240.0	17.6%	5.0
inst-698-10-120	51.4	415.7	14.8%	4.8
inst-698-20-120	67.1	628.6	12.8%	4.3
inst-698-30-120	71.8	693.5	12.2%	3.9
inst-698-50-120	68.5	696.9	12.0%	3.4
inst-744-1-30	6.9	30.3	24.0%	0.3
inst-744-5-30	16.3	123.4	15.9%	1.1
inst-744-10-30	25.0	230.0	14.1%	1.7
inst-744-20-30	39.4	416.9	11.6%	2.5
inst-744-30-30	49.9	564.6	10.6%	2.8
inst-744-50-30	62.2	726.5	10.4%	3.0
inst-744-1-120	10.5	31.6	21.8%	0.6
inst-744-5-120	18.9	123.1	14.1%	1.8
inst-744-10-120	26.3	231.4	11.9%	2.6
inst-744-20-120	39.3	434.1	9.8%	3.7
inst-744-30-120	49.7	601.4	9.3%	4.1
inst-744-50-120	57.9	741.9	8.8%	4.1
grid-540-200-30-0	223.2	539.8	40.7%	0.9
grid-540-200-30-1	223.0	539.6	40.7%	0.9
grid-540-200-30-2	223.1	539.7	40.6%	0.9
grid-540-200-30-3	222.2	539.7	40.6%	0.9

Table 2: Results of the proposed heuristic for all instance. OF = objective function, RS = requests served, Eco = The economy made by the passengers served, Ite-Time = Execution time of each iteration.

7 Concluding remarks

Taxi and ridesharing are used nowadays, but not with a systematic approach. We propose a system that can help people finding shared rides, which reduce emission of greenhouse gases, and has the advantage of money saving. Our framework is simple to use. It will provide the best way to go to the destination modeling a dynamic scenario, so people can share a ride even if the vehicle had already left its source. Although the optimization problem has complicated constraints, which make it NP-Hard, our method gives good solutions and can be used in practice. In our simulations, passengers on a shared trip had an economy of 18.58% on average. As future work, experimentations with graphs representing a real map of the city can be done. In order to do this, we need to compute shortest paths in a fast way. For this we could use Highway Hierarchies [Knopp *et al.*, 2007] as it is one of the fastest method to compute distance tables for road networks.

References

[Agatz *et al.*, 2011] Niels A.H Agatz, Alan L Erera, Martin W.P Savelsbergh, and Xing Wang. Dynamic ride-sharing: A simulation study in metro atlanta.(report). *Transportation Research Part B: Methodological*, 45(9):1450(15), 2011.

[Cordeau and Laporte, 2007] Jean-Francois Cordeau and Gilbert Laporte. The dial-a-ride problem: models and algorithms. *Annals OR*, 153(1):29–46, 2007.

[Herbawi and Weber, 2012] Wesam Mohamed Herbawi and Michael Weber. A genetic and insertion heuristic algo-

rithm for solving the dynamic ridematching problem with time windows. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference*, GECCO '12, pages 385–392, New York, NY, USA, 2012. ACM.

[Knopp *et al.*, 2007] Sebastian Knopp, Dorothea Wagner, and et al. Computing many-to-many shortest paths using highway hierarchies, 2007.

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

[Metz *et al.*, 2005] Bert Metz, Ogunlade Davidson, Heleen de Coninck, Manuela Loos, and Leo Meyer. *IPCC 2005 Report*. Cambridge University Press, 2005.

[Resende, 1995] Mauricio G. C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.

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

[Tao, 2007] Chi-Chung Tao. Dynamic taxi-sharing service using intelligent transportation system technologies. In *Wireless Communications, Networking and Mobile Computing, 2007. WiCom 2007. International Conference on*, pages 3209 –3212, sept. 2007.