10^{th} *International Conference on MOdeling, Optimization and SIMlation - MOSIM14 - November 5-7-2014- Nancy - France "Toward circular Economy"*

INSERTION HEURISTIC FOR A DYNAMIC DIAL-A-RIDE PROBLEM USING GEOGRAPHICAL MAPS

Sacha Varone* **Vytenis Janilionis**

University of Applied Sciences Western Switzerland
Haute École de Gestion de Genève
1022 Carouge - Switzerland
sacha.varone@hotmail.com

Independant Consultant
Kaunas - Lithuania
vytjani@gmail.com

ABSTRACT: *We present an insertion heuristic for a dynamic Dial-a-Ride Problem, applied to a real world application on taxi sharing. The application relies on city-sized geographical maps and location coordinates for taxis and customers. A customer sends a ride request to a taxi call center, which assigns to it in (quasi) real-time the most appropriate taxi, with respect to capacity and time constraints: maximum waiting time before pick up and maximum drive duration for each passenger's request. Passengers may share a taxi along some part of their journey. Our methodology relies on the computation of four shortest paths, followed by an insertion algorithm with a validity check. Each new request is so tentatively inserted into a taxi route. Experiments on real data shows the viability of our methods.*

KEYWORDS: *Dynamic DARP, Taxi sharing, Geographical maps.*

1 INTRODUCTION

The Dial-a-Ride Problem (DARP) is a type of a vehicle routing problem where multiple customers specify pick-up and drop-off requests, and vehicle route and schedules have to be defined. Customers specify their pick-up and drop-off locations usually associated with time window constraints that represent their desired pick-up and/or drop-off times. The usual objective of a DARP is to find for each vehicle a set of routes that can accommodate all customer requests, while minimizing customers' inconvenience with respect to transportation constraints.

In this paper we focus on a taxi company which wants to optimize its daily running costs by implementing a new business model allowing customers to share taxis. The idea is the following: while a customer is being taken to his destination in a taxi, a new customer ride request may arrive and it may be convenient to adjust the route of this taxi in order to include the new customer. Each customer specifies his pick up and drop off locations, and the most appropriate taxi has to be assigned to fulfil the ride request while respecting capacity constraints and assuming that multiple customers may share one taxi along some part of their journey. Additionally, to take customer convenience into account, specific time constraints have to be fulfilled: maximum waiting time before pick up and maximum drive duration for each customer.

Since customer requests arrive randomly during the day have to be fulfilled in real time, our problem is a Dynamic Dial-a-Ride Problem (DDARP). In literature such problems are not associated with the difficulty of finding the shortest paths needed to include each new customer (Cordeau and Laporte 2007, Berbeglia, Cordeau and Laporte 2010). Instead it is assumed that all shortest (quickest) paths between all possible locations can be pre-calculated and a resulting fixed distance (duration) matrix can be used by DDARP solving algorithms every time a new solution needs to be found. However, such approach has practical limitations: real road networks are dynamic and the traversal time for any given road segment may change significantly during the day (i.e. rush hours). As such circumstances may result in reasonably high solution quality losses, we find it insufficient to use the usual fixed distance (duration) matrix approach. Therefore we develop a heuristic based on Dijkstra (Dijkstra 1959) and A* (Floyd 1962) shortest paths computations and on insertion (Coslovich, Pesenti and Ukovich 2006, Häme 2011) algorithms. By using the DDARP constraints to cut down the solution space of the corresponding shortest paths problems (SPP), our heuristic is able to find solutions in (quasi) real time for city size problems even while dealing with live route searches.

The main existing algorithms and heuristics for SPP were reviewed in (Fu, Sun and Rilett 2006) while

*This research was funding thanks to the swiss CTI grant 15229-1 PFES-ES received by Sacha Varone

the most important research papers for DARP were summarized in (Cordeau and Laporte 2007) and (Berbeglia et al. 2010). Since, to our knowledge, there is no other research combining the two before mentioned problems, we give a brief overview of the methods and findings in both areas which gave foundations to our developed heuristic.

Dijkstra Algorithm (Dijkstra 1959) is a label setting algorithm used to solve one-to-one and one-to all SPPs. The most popular area limiting heuristic is the A* search algorithm (Floyd 1962). The main difference is that the latter uses an evaluation function instead of just labels when selecting nodes to examine. In addition to the value of the labels, the evaluation function includes an estimate of the minimum weight needed to reach the target node from each scanned node. This results in selecting nodes that are more likely to be on the shortest path from source to target and the search area can therefore be reduced significantly if the estimate function is properly chosen. A* algorithm is guaranteed to find the optimal solution if the estimate function never overestimates the actual weight to the destination node.

DDARP algorithms' performance has been investigated in (Hyytiå, Håme, Penttinen and Sulonen 2010) and route selection criteria for solving a large scale dynamic pickup and delivery problem. They showed that trying to assign a new customer to each vehicle one by one (i.e. immediate allocation) results in a reasonably small loss in the solution quality compared to an optimal allocation, where the existing customers may be reassigned to different vehicles. Additionally, they investigated the performance of an insertion algorithm, where the original customer sequence is preserved in each vehicle and the pick-up and drop-off points of the new customer are inserted in this sequence optimally. They showed that the solution quality of the insertion algorithm decreases with the increase in the number of customers per vehicle compared to an optimal complete enumeration algorithm. However, as these algorithms are asymptotically identical, the solution quality gap between them is almost negligible if there are few existing customers in vehicles. This gap was shown to decrease with the number of vehicles available.

(Coslovich et al. 2006) proposed a two-phase insertion technique for new customer queries. The first phase is designed to run off-line, while a vehicle is in movement, and finds a feasible neighbourhood for the current route of the vehicle. These results are then used for the second phase which is performed online when a new customer query arrives. This phase attempts to insert the pick-up and drop-off points of the new customer into routes belonging to the precalculated feasible neighbourhood. It was shown that such approach is able to produce real-time solutions that are close to an optimal static solution in quality.

(Häme 2011) studied an adaptive insertion algorithm for DARP with narrow time windows that could also be used for the dynamic case. They exploited the fact that having narrow time windows makes most of the new potential routes infeasible when trying to insert a new customer into an existing route. Using this, along with a parameter to limit the number of feasible routes considered, their adaptive insertion algorithm reduces the solution space greatly and speeds up the insertion procedure. Moreover, the algorithm is adaptive in a way that the parameter can adjust itself dynamically and expand the solution space, if no feasible solutions are found at first.

The remainder of the paper is organized as follows. Section 2 describes the problem of predicting the location of a moving vehicle, whose solution enables to accurately estimate the positions of vehicles. Section 3 describes the method that we use to solve the DDARP. In Section 4 we give an insight on practical applications of our heuristic by presenting results of a simulation experiment on real road maps. Finally, Section 5 summarizes our achievements, outlines limitations and provides recommendations for future work.

2 PREDICTION

Driver devices send their geographical coordinates generally at regular time intervals. However, technical problems such as loss of signal while passing through a tunnel, battery's device expiration, application turned off, etc. may prevent the sending of this information. Precise location of current driver's position has therefore to be estimated. Formally, the problem of predicting trajectories on a road network can be defined as follows, based on Eisner *et al.*(Eisner, Funke, Herbst, Spillner and Storandt 2011): let $G = (V, A, w)$ be a (road) network with a set V of vertices, a set A of arcs, and a function w on the arc set that gives a measure associated to an arc, usually a duration or a length. Given a path $P = v_{t_0}, v_{t_1}, \ldots, v_{t_i}$ which gives object's positions at times $t_0 < \ldots < t_i$, what could be the position of this object at times $t_{i+1} < \ldots < t_{i+k}$? Based on historical data, one may estimate the probability of reaching some target points from the last sent GPS positions, as done in (Liu, Jou and Lee 2010, Kim, Won, Kim, Shin, Lee and Kim 2007). In our application we do not have such historical data. We use the following scheme to solve this problem: either a driver has not defined a target position, meaning that he has no customer, or the taxi already has a destination.

In the former case, since there is generally no other information than its previous GPS position, the only reliable position's estimation consists in observing its

environment. For example, if the driver's last positions are along a highway, or along a one-way street, the next best possible position's estimation is toward the next junction or the next motorway exit. If the difference between the current time and the last GPS position is low, knowing the speed limitation of the road and with an assumption on the taxi's speed, it is possible to specify its approximate position.

In the latter case, when the taxi's route and its stops are known, the following assumption is made: the taxi follow a set of shortest paths along the stops. It is therefore possible to estimate its position, with an assumption on the taxi's speed on different types of roads. The method consists in running a shortest path algorithm to find the couples (position, time) along the (shortest) route.

3 RIDE INSERTION HEURISTIC

The following problem has to be solved: given a ride request from a source point s to a target destination t, which taxis can service this request, such that maximum waiting time before pick-up, and maximum time-to-target are respected for all passengers?

Our approach consists in first finding a set of shortest/quickest path toward the source point s. In this first step, all taxis able to pick up the new customer at the source point s within a maximum waiting time, are discovered. Next, three other sets of shortest/quickest paths are computed, which might be used as components of new routes for the taxis previously discovered. Last, the aforementioned new routes are constructed for those taxis by an insertion procedure, with an admissibility check.

The four sets of shortest paths are described as follows

1. a pick up: from the positions of the taxis to the location of the request
2. a pick up to destination: from the origin of the request to its destination
3. a destination to drop off: from the request destination to the drop off of current passengers in the taxi considered
4. a drop off to destination: from the drop off of current passengers in the taxi considered to the request destination

Quickest paths resulting from those four steps are denoted as, respectively, $P_{.\rightsquigarrow s}, P_{s\rightsquigarrow .}, P_{.\rightsquigarrow t}, P_{t\rightsquigarrow .}$. Those are sets of quickest paths.

Those quickest paths are then combined to form a new route for a taxi. This route is checked for its admissibility: maximum waiting time before pick-up, as well as maximum drive request duration for each passenger.

We note a quickest path from a to b as $a \rightsquigarrow b$, whereas its duration is noted $\delta(a, b)$.

3.1 Pick-up

The first problem to be solved consists in the selection of taxi candidates able to fulfill a ride request within a maximum waiting time, *maxwait*, before pick up.

One way to get such a taxi selection list could be to apply n times a shortest path algorithm, n being the number of taxis in the fleet, from their position to the customer's position. This means that a shortest path algorithm has to be applied each time a new request arrives. Knowing that the frequency of new requests in big cities may easily reach several dozens per seconds, such a methodology is not efficient.

In our approach, the taxi selection list is based on a many-to-one shortest paths, computed reversly from the pick-up position of the customer. Our shortest path algorithm is based on the well-known Dijkstra algorithm (Dijkstra 1959) for several reasons: first, the problem is restricted to a city-size problem since we consider a taxi company operating in a city. Second, using the true road network instead of a compact form as it is the case for highway hierarchies (Sanders and Schultes 2005) or contraction hierarchies (Geisberger 2008) algorithms, allows us to use real-time information such as temporary road restrictions, traffic jams, temporal speed limitations, ...

The adaptation of the Dijkstra algorithm is made as follows:

- All arcs are reversed: all arcs (i, j) in the underlying road network $G(V, A, w)$ become arcs (j, i). That is to say that, started from the location o of the request, each path π_{od} discovered during the algorithm represent a path from d to o.
- A limit *maxwait* is introduced as the maximum allowed waiting time before pickup. This limit is used as a stopping criterion: if the node currently considered in the Dijkstra algorithm has a duration label greater than *maxwait*, then there is no path from this node to the request location with a duration less than or equal to *maxwait*.

In the following pseudo-algorithm, we adopt the formulation available in (Cormen, Stein, Rivest and Leiserson 2001).

3.2 Pick up to destination

A taxi able to pick-up the new customer c_k may not have time to first drive to the destination of this new

Algorithm 1 ReverseBoundedDijkstra($G, s, maxwait$)

Require: Graph $G = (V, E, w >= 0)$, source s, length $maxwait$
Ensure: The single-source s shortest-paths problem on a weighted, directed graph G, restricted to an upper bound length $maxwait$.

```
for all v ∈ V do
    v.d = ∞
    v.π = nil
end for
s.d = 0
S = ∅
Q = V
while Q ≠ ∅ do
    u = extract-min(Q)
    if u.d > maxwait then
        STOP
    end if
    S = S ∪ {u}
    for all v such that (v, u) ∈ A do
        if v.d > u.d + w(u, v) then
            v.d = u.d + w(u, v)
            v.π = u
        end if
    end for
end while
```

customer and then drive to the drop off locations of its other customers $c_1, c_2, \ldots, c_{k-1}$. In this case, it might be efficient, after the pick-up of the new customer c_k, to first drive to the drop off locations of some already-in customers $c_1, c_2, \ldots, c_{k-1}$, before driving to the destination t.

We use a modified A* algorithm (Hart, Nilsson and Raphael 1968) from the pick-up location s of the new request k, toward its destination t. The A* algorithm stops after the destination has been reached. Therefore we modify it so that detours can be done. For that purpose, let's define λ as an increase coefficient on the duration of a quickest path $\delta(s,t)$ from the pick-up location s to the destination t. Once the "pure" A* algorithm has reached the destination t, hence $\delta(s,t)$ is known, then it is not stopped but continue until all reached nodes have a label duration greater than or equal to $\lambda\delta(s,t)$. In our case, we set $\lambda = 1.2$, which means that all customers allow trip duration at most 20% greater than that of a quickest path.

From this algorithm the greatest duration until destination D_k^{dest} for customer c_k is computed as

$$D_k^{dest} = maxwait + \lambda\delta(s,t)$$

We set the latest arrival time as

$$t_{\max k} = t_0 + D_k^{dest}$$

where t_0 is the time when the request has been sent by customer c_k. Note that this maximum time $t_{\max k}$ remains constant until the drop off of customer k.

3.3 Destination to drop off

Once the taxi has reached the destination t of the new customer c_k, it might happen that other customers are still waiting for their drop off. In this case, shortest paths have to be computed from the destination t to the next drop off locations. This is done with a Bounded Dijkstra algorithm, in which we limit the expansion until a maximal duration $D_k^{dest-drop}$ has been reached.

$$D_k^{dest-drop} = \max_{i=1,\ldots,k-1}(t_{\max i} - (t_0 + \delta(s,t)))$$

This limit corresponds to the situation in which a taxi is at time t_0 at the pick-up location s, drives to the destination t, and then drives to the other drop off locations.

3.4 Drop off to destination

Suppose that a taxi has picked up the new customer c_k, but then drives to a drop off location of some of its other customers. Therefore, shortest paths have to be computed from the drop off locations to the destination t.

This shortest path computation is done with a Reverse Bounded A* algorithm. The bound $D_k^{drop-dest}$ is set to

$$D_k^{drop-dest} = \lambda\delta(s,p)$$

We use a similar modification of the A* algorithm as done in Section 3.1.

3.5 Ride insertion

So far, four sets of shortest paths have been computed. A schematic representation of the situation is shown in Figure 1. The new request is represented with a green node as the pick-up location, and its destination with a red node. Other nodes represent either the current location of the taxis, or the drop off locations of their customers. In Figure 1 there are two taxis with their routes: the first one contains four nodes, and the second one contains two nodes. The pick up shortest paths are represented in the upper left, pick-up to destination in the upper right, destination to drop off in the bottom left, and drop off to destination in the bottom right.

By performing two forward and two reverse shortest path searches, our algorithm finds all paths that construct a *possibly feasible* solution for the DDARP problem. Not all routes constructed from those four sets of paths are feasible solutions, but all feasible

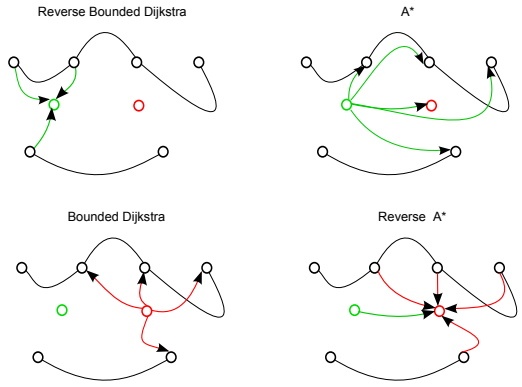

Figure 1 – Sets of shortest paths computation

routes can be constructed from them. An admissibility check is therefore performed on the maximum travel time for each passenger, to ensure the feasibility of the solution.

The idea of this ride insertion heuristic is the following: start from a taxi path, from its current location x_0, to its drop off locations $x_1, \ldots, x_{k-1}$. The considered path is therefore $x_0, x_1, \ldots, x_{k-1}$. From this path, try to insert the new pick up location s, and the new drop off location t. We do not change the relative ordering of the drop off locations $x_1, \ldots, x_{k-1}$, so that the number of solutions is kept reasonable : k possibilities for the insertion of the pick up location s, and at most k possibilities for the destination t, which has to be inserted after s on the path. The number of such paths which include s and t is $\frac{k(k+1)}{2}$.

Each solution path uses some shortest (or quickest) sub-paths. The construction of the new path for a taxi is given by Algorithms 2 and 3:

In Algorithm 2, we consider a stop as either a pick up position or a drop off position. Starting from a route P associated to a taxi, a new point x is tried to be inserted after the position i in the route P. Since this new route could be inadmissible, a check has to be performed to test if the quickest path from i to x and then from x to $i+1$ is possible, i.e. respects the maximum waiting time limit *maxwait*. If this is not the case, then the algorithm returns null as this solution is not admissible, else, an update of the estimated time to arrival is done with t'. This allows to check for the maximum arrival time $t_{\max i}$ at each drop off location. If each estimated time to arrival is below its maximum arrival time, then the new route P' is admissible and the algorithm returns P'. Its complexity is in $O(k)$, where k is the number of stops of the current driving path P.

Algorithm 3 tries to insert the new pick up location s and the new drop off location t inside the current

Algorithm 2 LocalInsertion

Require: A path $P = v_0, v_1, \ldots, v_{k-1}$
$t_i, i = 0 \ldots k-1$ estimated arrival times at vertex v_i
$t_{\max i}, i = 0 \ldots k-1$ maximal time to destination v_i
$i \in \{0, \ldots, k-1\}$ insertion position
x new stop (either s or t).
Ensure: A new path P' with new estimated arrival times t'
if $v_i \rightsquigarrow x \notin P_{\cdot \rightsquigarrow x}$ OR $x \rightsquigarrow v_{i+1} \notin P_{x \rightsquigarrow \cdot}$ **then**
return null
end if
$P' =$ Insert x after v_i in P
for all $j = 0 \ldots i$ **do**
$t'_j = t_j$
end for
$t'_x = t_i + \delta(v_i, x)$
$t'_{i+1} = t'_x + \delta(x, v_{i+1})$
for all $j = i+2 \ldots k-1$ **do**
$t'_j = t'_{j-1} + (t_j - t_{j-1})$
end for
if $t'_x > t_{\max x}$ **then**
return null
end if
for all $j = i+1 \ldots k-1$ **do**
if $t'_j > t_{\max j}$ **then**
return null
end if
end for
return P', t'

route P of a taxi. It uses Algorithm 2 as a subrouting. Not all solutions are tested, since the relative order of the stops are maintained during the insertion attempts. If the insertions of s and t result in a feasible solution, then this new solution may be the best possible solution and therefore an update on the latter is done via Algorithm 4. Its complexity is in $O(k^3)$ since there is a main loop, with an inner loop and the call to Algorithm 2, where k is the number of stops of the current driving path P.

Algorithm 4 first checks if it is the first time that a feasible solution P'' is submitted. If it is the case, then P'' is de facto the best solution found so far. Else, the value of the objective function is compared between the current best solution Pbest and the new solution P''. Its complexity is in $O(1)$.

3.6 Objective function

The definition of the objective function f depends on the goal of the managers: the quality of service might be improved in several aspects

- The waiting time before pick up could be mini-

Algorithm 3 Insertion of a new customer

Require: A path $P = v_0, v_1, \ldots, v_{k-1}$
s, t new pick up and drop off positions).
Ensure: Best insertion or null
nbsolution = 0
for all $i = 0 \ldots k-1$ **do**
P' =LocalInsertion(P, i, s)
if P' is not null **then**
for all $j = s, i+1 \ldots k-1$ **do**
P'' =LocalInsertion(P', j, t)
if P'' is not null **then**
nbsolution = nbsolution +1
UpdateBest(Pbest,P'',nbsolution)
else
break
end if
end for
end if
end for
return Pbest

Algorithm 4 UpdateBest

Require: Path Pbest, P''
Number of solutions *nbsol*
Ensure: Best path
if $nbsol = 1$ **then**
Pbest = P''
else
if P'' is "better" than Pbest **then**
Pbest = P''
end if
end if
return Pbest

mized.

$$\min f(P, s, t) = \min t_s$$

- The mean time to destination may be minimized, which is similar to the maximization of the margin. This might also be adapted, for example by the summation of the square differences, so that no passenger stay too long in the taxi, and thus reduces extreme service time to destination.

$$\max f(P, s, t) = \max \sum (t_{\max i} - t_i)$$

- The last drop off time might be minimized, so that trip duration is kept small for the driver. The idea is to take the makespan of the drive as the objective function.

$$\min f(P, s, t) = \min t_{last}$$

- The detour length for the taxi might be minimized so that fuel consumption would be reduced. This has been our choice.

$$\begin{aligned} \min f(P, s, t) &= \min \delta(i, s) + \delta(s, i+1) \\ &\quad +\delta(j, t) + \delta(t, j+1) \end{aligned}$$

Remark: some $\delta(\cdot, \cdot)$ might be 0

4 PRACTICAL APPLICATION

Real routing applications need geographical maps, so that routing algorithms can be applied on. There exist several different map sources such as google maps[1], which is one of the most known and used maps, the Microsoft Bing maps[2], nokia maps[3] or also openstreetmap[4] (OSM). OSM is a a collaborative open source project whose goal is to create a free editable map of the world. This has been our choice to run our application. Download of parts of this world map is available from Geofabrik[5] for example.

We use OsmSharp[6], described as "an open-source mapping tool designed to work with OpenStreetMap-data." for data processing. It uses the C# programming language, developed by Microsoft. We use its OSM data processing as well as its routing library to test our algorithms.

As there was no taxi sharing services from the company with which we are working, we tested our algorithm with simulations. We have adopted the methodology used by Erbawi and Weber (Herbawi and Weber 2012), that we applied to the real data set from the company. From the log file of a taxi fleet company, that contains the Origin/Destination and timestamp of the drives, the data is partitioned into 2/3 as requests and 1/3 as drives. We then solve the ridematching problem in dynamic ridesharing, i.e. to match the requests to the available drives, considering time-windows.

For illustration purpose only, we present the results from a randomly selected few hours of a particular day in a small swiss city (real data): 138 requests and 69 drives. In the following tables the exponent m means that ride-sharing is allowed, that is to say that a taxi might serve several customers at the same time, whereas no exponent means that at most one customer is served at a given time.

w^u = maximum waiting time *maxwait* [min]
nbs = number of satistied requests
nbd = number of potential drivers
did = mean distance for drivers [km]
dud = mean duration for drivers [min]
dir = mean distance for riders [km]
dur = mean duration for riders [min]
w = mean waiting time for riders before pickup [min]

Table 1 illustrates that the number of satisfied requests in a ride-sharing environment is greater than in a non ride-

[1] https://maps.google.com/
[2] http://www.bing.com/maps/
[3] http://m.here.com
[4] http://www.openstreetmap.org
[5] http://download.geofabrik.de/
[6] http://www.osmsharp.com/

Table 1 – Global

λ	w^u	nbs^m	nbs	nbd^m	nbd
1.8	30	106	88	65	67
1.8	20	94	75	63	64
1.8	10	64	50	49	50
1.5	30	85	76	62	64
1.5	20	74	63	58	58
1.5	10	48	42	41	42
1.2	30	35	33	29	31
1.2	20	28	26	23	25
1.2	10	18	18	17	18

sharing environment, and needs, of course, less drives to fulfil those requests.

Table 2 – Riders

λ	w^u	w^m	w	dir^m	dir	dur^m	dur
1.8	30	15.8	17.2	14.5	11.1	18.9	13.6
1.8	20	11.3	11.2	13.8	10.5	17.8	13.1
1.8	10	6.6	6.7	13.4	11.2	16.8	13.4
1.5	30	15.2	15.8	14.3	12.5	16.3	13.8
1.5	20	10.8	11.0	13.5	12.1	15.7	13.5
1.5	10	6.4	6.3	11.2	10.5	13.8	12.6
1.2	30	14.5	14.6	13.6	14	13.5	13.2
1.2	20	10.5	10.7	13.7	14.2	13.2	12.9
1.2	10	5.9	5.8	13.4	13.3	12.5	12.5

Table 3 – Drivers

λ	w^u	did^m	did	dud^m	dud
1.8	30	69.0	68.2	69.2	67.7
1.8	20	71.7	70.3	69.2	66.4
1.8	10	65.3	64.3	61.5	59.3
1.5	30	65.1	66.3	62.5	62.3
1.5	20	66.0	66.0	60.8	59.5
1.5	10	62.5	62.5	55.7	55.0
1.2	30	60.7	60.8	50.7	50.7
1.2	20	59.3	59.3	49.7	49.7
1.2	10	59.2	59.2	49	49.1

Table 2 and 3 illustrate that riders' mean waiting time decrease in a ride-sharing environment since more vehicles might be able to pick them up. On the other side, mean durations and mean lengths of the drives generally increase, but might also in some cases decrease.

In order to get robust results, we suggest a second set of simulations to be done in the following way: generate $n = 100$ taxi routes simultaneously on a city sized area. The number of passengers in those taxis are distributed uniformly between 0 and 4. The demand of a new ride, i.e. a new pick up location and a new drop off location is based on the Poisson distribution $\mathcal{P}(\lambda)$ where $\lambda = 20$ is the average demand per hour. The mean quickest ride duration is $\bar{\delta} = 10$ minutes and the average time a customer spends in the system is $\tilde{\delta} = 20$ minutes. The mean service time $\mu = \frac{1}{\delta} = 0.05$ is the mean duration between the demand and service completion. The experimental conditions is chosen in such a way that for a taxi company without taxi sharing, which is usually the case, the arrival rate λ exceeds the system's total capacity $n\tilde{\mu} = 5$. Then perform the test on a time horizon of 1 hour.

Results from simulations show that our approach is viable since it only requires 1-2 seconds to obtain a solution and increased the number of served customer's requests.

5 CONCLUSION

We have presented a heuristic method that solves a dynamic Dial-A-Ride Problem, that occurs in the case of a taxi sharing application. Although an optimal solution is not guaranteed, our method benefits from very short running time, since it has been designed for a real time application. This allows a taxi company to serve more customers with the same capacity, at the price of a completely disruptive business system, compared to what is available now.

We fix two parameters *maxwait* and λ as constant. It is also possible to make them dependent on the customer, which would reflect the sensibility of each customer to the time. The question to know if such a possibility is appropriate or not has to be answered through a marketing research: would the customer always choose the quickest trip or would he prefer to stay more time in the taxi, so that the price is decreased?

Extension to this work may be done considering real-time traffic information. In that case, it might be useful to reject some demands (!) in order to keep the system running as smooth as possible. Another extension would be booking in advance, so that this static part of the problem could be solved optimally. The key point would be to define the threshold value between exact methods and heuristic methods.

Some limitations occur if applying the same heuristic without adaptations to bus sharing application, in which buses act as taxis. In this case, the number of passengers might increase up to 50 or more, which results in an increase in computational time that could not be acceptable for a real-time application.

References

Berbeglia, G., Cordeau, J.-F. and Laporte, G., 2010. Dynamic pickup and delivery problems, *European Journal of Operational Research* **202**(1): 8–15. A lire absolutment, tres souvent cite.

Cordeau, J.-F. and Laporte, G., 2007. The dial-a-ride problem: models and algorithms, *Annals OR* **153**(1): 29–46.

Cormen, T. H., Stein, C., Rivest, R. L. and Leiserson, C. E., 2001. *Introduction to Algorithms*, 2nd edn, McGraw-Hill Higher Education.

Coslovich, L., Pesenti, R. and Ukovich, W., 2006. A two-phase insertion technique of unexpected customers for a dynamic dial-a-ride problem, *European Journal of Operational Research* **175**(3): 1605–1615.

Dijkstra, E. W., 1959. A note on two problems in connexion with graphs., *Numerische Mathematik* **1**: 269–271.

Eisner, J., Funke, S., Herbst, A., Spillner, A. and Storandt, S., 2011. Algorithms for matching and predicting trajectories, *in* M. Müller-Hannemann and R. F. F. Werneck (eds), *ALENEX*, SIAM, pp. 84–95.

Floyd, R. W., 1962. Algorithm 97: Shortest path, *Commun. ACM* **5**(6): 345.

Fu, L., Sun, D. and Rilett, L., 2006. Heuristic shortest path algorithms for transportation applications: State of the art, *Computers & Operations Research* **33**(11): 3324 – 3343. Part Special Issue: Operations Research and Data Mining.

Geisberger, R., 2008. *Contraction hierarchies: Faster and simpler hierarchical routing in road networks*, Master's thesis, Institut für Theoretische Informatik, Universität Karlsruhe.

Häme, L., 2011. An adaptive insertion algorithm for the single-vehicle dial-a-ride problem with narrow time windows, *European Journal of Operational Research* **209**(1): 11 – 22.

Hart, P. E., Nilsson, N. J. and Raphael, B., 1968. A formal basis for the heuristic determination of minimum cost paths, *IEEE Transactions on Systems, Science, and Cybernetics* **SSC-4**(2): 100–107.

Herbawi, W. M. and Weber, M., 2012. A genetic and insertion heuristic algorithm for solving the dynamic ridematching problem with time windows, *Proceedings of the Fourteenth International Conference on Genetic and Evolutionary Computation Conference*, GECCO '12, ACM, New York, NY, USA, pp. 385–392.

Hyytiå, E., Håme, L., Penttinen, A. and Sulonen, R., 2010. Simulation of a large scale dynamic pickup and delivery problem, *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*, SIMUTools '10, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), ICST, Brussels, Belgium, Belgium, pp. 77:1–77:10.

Kim, S.-W., Won, J.-I., Kim, J.-D., Shin, M., Lee, J. and Kim, H., 2007. Path prediction of moving objects on road networks through analyzing past trajectories, *in* B. Apolloni, R. Howlett and L. Jain (eds), *Knowledge-Based Intelligent Information and Engineering Systems*, Vol. 4692 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 379–389.

Liu, C.-L., Jou, E. and Lee, C.-H., 2010. Analysis and prediction of trajectories using bayesian network, *Natural Computation (ICNC), 2010 Sixth International Conference on*, Vol. 7, pp. 3808–3812.

Sanders, P. and Schultes, D., 2005. Highway hierarchies hasten exact shortest path queries, *13th European Symposium on Algorithms*, Vol. 3669 of *LNCS*, Springer, pp. 568–579.