A Genetic and Insertion Heuristic algorithm for solving the dynamic ridematching problem with time windows

Wesam Herbawi and Michael Weber
Institute of Media Informatics
University of Ulm
Ulm, Germany
firstname.lastname@uni-ulm.de

ABSTRACT

In this paper, we address the dynamic ridematching problem with time windows in dynamic ridesharing. The dynamic ridesharing is a special type of ridesharing where the participants form ridesharing on short notice. The ridematching problem is to assign riders to drivers and to define the ordering and timing of the riders' pickup and delivery. Because not all information is known in advance, the problem is dynamic. This is an optimization problem where we optimize a multicriteria objective function. We consider minimizing the total travel distance and time of the drivers and the total travel time of the riders and maximizing the number of the transported riders.

We propose a genetic and insertion heuristic algorithm for solving the addressed problem. In the first stage, the algorithm works as a genetic algorithm while in the second stage it works as an insertion heuristic that modifies the solution of the genetic algorithm to do ridematching in real-time. In addition, we provide datasets for the ridematching problem, derived from realistic data, to test the algorithm. Experimentation results indicate that the algorithm can successfully solve the problem by providing answers in real-time and it can be easily tuned between response time and solution quality.

Categories and Subject Descriptors

I.2.8 [**Problem Solving, Control Methods, and Search**]: Scheduling, Heuristic methods; G 1.6 [**optimization**]: Combinatorial optimization

General Terms

Algorithms

Keywords

Ridesharing, ridematching, genetic algorithms, heuristics

GECCO'12, July 7-11, 2012, Philadelphia, Pennsylvania, USA.

1. INTRODUCTION

The shared use of vehicle by its driver and one or more passengers (riders) is called ridesharing. Recently, there is an increasing interest in ridesharing as response to the environmental pollution and climate change, traffic congestion and oil cost [4]. The increasing ubiquity of mobile handheld devices also made the idea of ridesharing to be more appealing. It takes the idea of ridesharing from static to dynamic ridesharing where a rideshare can be formulated on very short notice [1].

In this study, we are interested in the ridematching problem with time windows (RMPTW) in dynamic ridesharing. The problem consists of a set of drivers' offers (representing a set of vehicles) and riders' requests. Each participant (driver or rider) specifies a source and a destination location for her/his trip. In addition, each participant specifies an earliest departure time and a latest arrival time which we use to define a time window at each location as we show later. Drivers define maximum travel time and distance for their trips and they are willing to detour to serve some riders. Riders define maximum travel time for their trips. The RMPTW is to assign riders to drivers and determine the timing and order of the pickup and delivery of riders in real-time. Because information about the drivers' offers and riders' requests is available only after their arrival to the system, the problem is dynamic.

The RMPTW is an optimization problem and it is subject to multicriteria optimization. In this study we consider the following criteria to optimize (We use the word vehicle instead of driver):

c_1. The total distance of vehicles' trips (to be minimized)

c_2. The total time of vehicles' trips (to be minimized)

c_3. The total time of the trips of the matched riders' requests (to be minimized)

c_4. The number of matched (served) riders' requests (to be maximized)

Note that the trip's time might include some waiting time (A vehicle with possibly some riders is waiting to pick up a rider when time allows). Criterion c_4 is conflicting with the other three criteria. Also criteria c_1 and c_2 could be conflicting with criterion c_3 as vehicles might tend to take riders for longer time to reduce their own time and distance.

The RMPTW could be seen as a pickup and delivery problem with time windows (PDPTW) which is a hard optimization problem. In this study we propose an algorithm alter-

nating between a genetic algorithm and an insertion heuristic to solve static snapshots of the problem. In addition we provide real world datasets for the dynamic RMPTW extracted from a travel and activity survey for northeastern Illinois conducted by Chicago Metropolitan Agency for Planning. The datasets are used to test the proposed algorithm and could be used as benchmarks.

To the best of our knowledge, the only work that considers a similar ridematching problem is the one by Agatz et al. [2]. However, in their problem definition, a driver can make only one pickup and one delivery. While this assumption makes the problem easier to solve, it prevents the driver from serving some riders even if they are on his exact way. The problem is represented using a maximum-weight bipartite matching model and the optimization software CPLEX is used to solve it. In this work, we consider the possibility of making more than one pickup and delivery where we could simply put a constraint on the number of such operations.

The rest of the paper is organized as follows. In Section 2, we formally describe the problem and provide the mathematical model. The proposed algorithm is explained in Section 3 followed by the experimentation and results in Section 4. Finally we outline the conclusions of the paper.

2. FORMAL PROBLEM DEFINITION AND THE MATHEMATICAL MODEL

The ride matching problem with time windows is similar to the PDPTW especially the dial-a-ride problem (DAR). Therefore the formulated mathematical model is very similar to the DAR problem model in [6] which in turn is an extension to the pickup and delivery problem model in [9] and [3]. The main difference between the RMPTW problem and the DAR problem is that, in RMPTW the vehicles are not free to move everywhere\when and limited by the vehicles' sources and destinations (with possible detour flexibility) and time windows. We adapt the model of DAR problem for the RMPTW and modify the definition of the objective function and add additional constraints on it (Constraints 12, 13 and 14 in the model).

The problem consists of a set $R = \{1, 2, ..., n\}$ of n riders' requests and a set $V = \{2n + 1, 2n + 2, ..., 2n + v\}$ of v vehicles representing drivers' offers, $R \cap V = \varnothing$. For each rider's request $i \in R$, we have a pickup point i, a delivery point $i+n$, demand dm_i defining the numbers of persons to be delivered from point i to point $i + n$, earliest departure time ED_i from point i, latest arrival time LA_i to point $i+n$ and constants AT_i and BT_i to define the rider's maximum travel time MTT_i. Let $t_{i,j}$ be the direct travel time between points i and j. We compute MTT_i as done in [5]: $MTT_i = AT_i + BT_i \cdot t_{i,i+n}$. For each pickup point i and delivery point $i+n$, we compute a time window as follows: $[a_i, b_i] = [ED_i, LA_i - t_{i,i+n}]$ denoting the earliest and latest pickup time and $[a_{i+n}, b_{i+n}] = [ED_i + t_{i,i+n}, LA_i]$ denoting the earliest and latest delivery time. Figure 1 shows the calculation of the time windows.

For each vehicle $k \in V$ we have a source point k, destination point $k+v$, a maximum capacity C^k, earliest departure time ED_k from point k, latest arrival time LA_k to point $k+v$, and constants AT_k, BT_k, AD_k and BD_k to define the vehicle's maximum travel time MTT_k and distance MTD_k. The calculation of the time windows for the points k and $k+v$ ($[a_k, b_k]$ and $[a_{k+v}, b_{k+v}]$) that denote the earliest and latest departure time and the earliest and latest arrival time of vehicle k respectively is the same as for riders' points. The calculation of the maximum vehicle's travel time MTT_k is the same as for riders. Let $d_{i,j}$ denote the direct travel distance between points i and j, then the maximum vehicle's travel distance is $MTD_k = AD_k + BD_k \cdot d_{k,k+v}$.

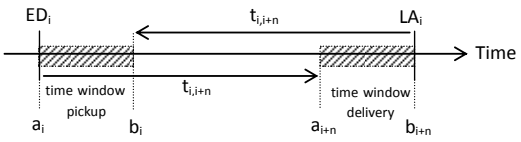


Figure 1: Time window calculation at pickup point i and delivery point $i + n$ for request i

Let $P = \{1, ..., n\}$ denote the set of pickup points and $D = \{n+1, ...2n\}$ denote the set of delivery points. The set of all pickup and delivery points is $N = P \cup D$ and the set of all points including the vehicles' sources and destinations is $A = N \cup \{k, k + v\}\ \forall k \in V$. Let $l_i = dm_i$ and $l_{i+n} = -dm_i$ represent the load change at points i and $i + n$ respectively. The load after servicing point i is L_i^k. Let T_i^k denote the service starting time at point i by vehicle k. It denotes the pickup and delivery times for riders and the departure and arrival times for vehicles (drivers). The service time at point $i \in N$ (time for pickup or delivery) is s_i and $\forall i \in A \setminus N$, $s_i = 0$. A binary decision variable $x_{i,j}^k$ is set to 1 if vehicle k services point i and travels directly to point j to service it. it is set to 0 otherwise. The multicriteria objective function that minimizes the total distance and time of the vehicles' trips and the total time of the trips of the matched riders' requests and maximizes the number of matched riders' requests r is provided in Equation 1. The weights α, β, γ and δ define the relative importance of the different components.

$$min\ \alpha \sum_{k \in V} \sum_{i,j \in A} x_{i,j}^k d_{i,j} + \beta \sum_{k \in V} (T_{k+v}^k - T_k^k) + \gamma \sum_{k \in V} \sum_{i \in P} (T_{i+n}^k - s_i - T_i^k) + \delta(n - r) \quad (1)$$

subject to:

$$\sum_{k \in V} \sum_{j \in P \cup \{k+v\}} x_{k,j}^k = v \quad (2)$$

$$\sum_{k \in V} \sum_{i \in D \cup \{k\}} x_{i,k+v}^k = v \quad (3)$$

$$\sum_{k \in V} \sum_{j \in A} x_{i,j}^k \leq 1, \forall i \in N \quad (4)$$

$$\sum_{j \in N} x_{i,j}^k - \sum_{j \in N} x_{j,i+n}^k = 0, \forall k \in V, i \in P \quad (5)$$

$$x_{i,j}^k (T_i^k + s_i + t_{i,j} - T_j^k) \leq 0, \forall k \in V, i, j \in A \quad (6)$$

$$a_i \leq T_i^k \leq b_i, \forall k \in V, i \in A \quad (7)$$

$$T_i^k + s_i + t_{i,i+n} \leq T_{i+n}^k, \forall k \in V, i \in P \quad (8)$$

$$x^k_{i,j}(L^k_i + l_j - L^k_j) = 0, \forall k \in V, i, j \in A \quad (9)$$

$$l_i \leq L^k_i \leq C^k, \forall k \in V, i \in P \quad (10)$$

$$L^k_k = L^k_{k+v} = 0, \forall k \in V \quad (11)$$

$$\sum_{i,j \in A} x^k_{i,j} d_{i,j} \leq MTD_k, \forall k \in V \quad (12)$$

$$(T^k_{k+v} - T^k_k) \leq MTT_k, \forall k \in V \quad (13)$$

$$(T^k_{i+n} - s_i - T^k_i) \leq MTT_i, \forall i \in P \quad (14)$$

$$x^k_{i,j} \in \{0, 1\}, \forall i, j \in A \quad (15)$$

Constraints (2) and (3) ensure that the number of vehicles leaving their sources is equal to the number of vehicles arriving to their destinations. Constraint (4) ensures that a rider's request is matched at most once and constraint (5) ensures that both the pickup and delivery of the rider are performed by the same vehicle. Constraint (6) ensures that the minimal travel time between two consecutive points is not violated and constraint (7) ensures that the time windows are not violated. Constraint (8) is a precedence constraint where a pickup point has to be visited before its corresponding delivery point. Constraint (9) ensures that if vehicle k has a load L^k_i after servicing point i, it must have a load of $l_j + L^k_i$ after servicing point j and therefore it ensures that the number of loaded riders at a pickup point should be equal to the number of unloaded riders at its corresponding delivery point.

Constraint (10) ensures that the vehicles' capacities are not exceeded and the load of the vehicle is at least l_i after servicing a pickup point i. Constraint (11) ensures that vehicles depart without having riders onboard and arrive to their destinations without having riders onboard (Note that riders and vehicles might share the same geographic sources and destinations, however we consider them to be distinct points). Constraints (12) and (13) ensure that the maximum distance and time, respectively, for each vehicle's trip is within a defined value. Constraint (14) ensures that the time of the trip of each rider is within his defined value.

3. THE PROPOSED ALGORITHM

To solve the dynamic RMPTW, we divide the day into a set of time periods. Each period starts with executing a genetic algorithm to solve a static version of the problem given all known requests and offers. After the execution of the genetic algorithm we utilize an insertion heuristic very similar to the one proposed by Jaw et al. [5] to give an online answer, by updating the solution produced by the genetic algorithm, in real-time for each newly received request/offer until the end of the period. All non-matched requests by the genetic algorithm and the insertion heuristic are stored for future matching and to be as an input for the genetic algorithm in the next time period if not already matched. The genetic algorithm as a metaheuristic is supposed to produce better quality matches and to better optimize the objective function as compared to the insertion heuristic that is subject to local optima. However, the insertion heuristic is more suitable for the dynamic RMPTW as it can give answers in real-time.

The shorter the time period, the more often the genetic algorithm is executed resulting in better results on the cost of longer execution time. On the other hand, if the time period is long, then we put more pressure on the insertion heuristic resulting in a more responsive algorithm with probably lower quality solutions.

A^+	5^+	5^-	4^+	7^+	4^-	7^-	A^-		
B^+	1^+	3^+	3^-	1^-	B^-				
C^+	8^+	2^+	8^-	2^-	C^-				
D^+	6^+	6^-	D^-						
E^+	10^+	10^-	9^+	11^+	9^-	11^-	12^+	12^-	E^-

Figure 2: Solution representation. Five Vehicles (A-E) and twelve matched riders' requests (1-12). + - denote the pickup and delivery points of riders' requests and the vehicles' sources and destinations

3.1 The Genetic Algorithm

In this section, we describe the different components of the proposed genetic algorithm. The proposed algorithm is a generational genetic algorithm gGA that follows the general skeleton of gGAs (with binary tournament selection).

3.1.1 Solution Representation

A solution is represented as a schedule of v routes, one route for each vehicle. Each route starts with the source point of the vehicle followed by a list of pickup and delivery points and ends with the destination point of the vehicle. The pickup and delivery points are inserted satisfying the constraints defined in the model. A solution may not match all riders' requests and therefore we keep a list of all non-matched requests. Figure 2 shows an example solution representation matching 5 vehicles (A - E) and 12 riders' requests (1 - 12). To speed up the genetic algorithm, we keep for each vehicle a list of matchable rider's requests taking time and distance feasibility into consideration.

3.1.2 Solution Initialization

An initial solution is created as follows. We randomly select a route to initialize among the v routes. The first two points to be inserted in the route are the source and destination points of the vehicle. Initially the two points are scheduled at their earliest time a_i (to take advantage of the maximum possible push forward as below). Then we continuously select a random pair of pickup and delivery points from the list of matchable rider's requests of the vehicle of the selected route and try to insert them in the route. The process repeats for all routes.

During the initialization process, we adopt the Solomon's insertion method [8]. Let $p = \{p_1, p_2, ..., p_u\}$ be a partially created route for vehicle k. The service time at p_i is $T^k_{p_i} = max\{a_{p_i}, T^k_{p_{i-1}} + s_{p_{i-1}} + t_{p_{i-1},p_i}\}$. If the vehicle arrives too early at point p_i, then it should wait before servicing p_i and the waiting time at p_i is $w_{p_i} = max\{0, a_{p_i} - T^k_{p_{i-1}} - s_{p_{i-1}} - t_{p_{i-1},p_i}\}$.

Now we discuss the necessary conditions for time feasibility when inserting a new point between the points p_{i-1} and p_i, $1 < i \leq u$. Let $newT^k_{p_i}$ denote the new service time by vehicle k for point p_i after inserting the new point before p_i. Assuming that the triangular inequality holds both on time and distance between the points, then the insertion of the new point will result in a time push forward (PF) in the route at p_i such that $PF_{p_i} = newT^k_{p_i} - T^k_{p_i} \geq 0$. It will also result in a time push forward at the points after p_i such that $PF_{p_{j+1}} = max\{0, PF_{p_j} - w_{p_{j+1}}\}$, $i \leq j < u$. After the time push forward, the time window and/or the maximum travel time constraints of some point p_j, $i \leq j \leq u$, might be violated. We sequentially check these two time feasibility constraints until we reach a point p_j with $PF_j = 0$ or its time window or maximum travel time constraint is violated or in the worst case until $j = u$.

In addition to time feasibility, we check the driver's maximum distance constraint after inserting the new point p. The distance should be less than MTD_k. For the precedence constraint, we always insert the pickup point before it corresponding delivery point and we search for a feasible insertion position for the delivery point in the route to the right of its pickup point.

3.1.3 Crossover Operator

We have defined a single point crossover operator. Given two parent solutions, we randomly select a crossover position (route index) and we make crossover by exchanging the routes among the parents starting from the crossover position upwards as in Figure 3. Note that the resulting children do not violate any of the defined constraints except constraints (4) and (5) (a riders' request could be matched twice with different vehicles). Example violation points are shaded in the child solutions. In addition to the constraints violation, some pairs of pickup and delivery points that exist in the parents might disappear in the children because of the crossover (e.g. points 10^+ and 10^- in child 1).

After the crossover operation, we apply a repair operation for the children. The repair operation removes all the pickup and delivery points in the routes after the crossover position whenever they exist in any route before the crossover position. Besides the repair operation, we also try to insert the pickup and delivery points of the riders' requests that are not matched in the resulting children after each crossover in a process similar to the one during solutions initialization.

3.1.4 Mutation Operators

We have defined five different mutation operators. All of them are on the route level (gene level). A mutation operation that violates any of the constraints is rejected.

The first mutation operator is the push backward. In this operator, we randomly select a point p_i from the route and define the push backward (PB_i) value as a random percentage of the time difference between the service time of the selected point and its earliest service time: $PB_i = random(T^k_{p_i} - a_{p_i})$. We also push backward all the points after p_i in the route: $PB_{p_{j+1}} = min\{PB_j, T^k_{p_{j+1}} - a_{p_{j+1}}\}$, $i \leq j < u$. We stop the push backward operation when we reach a point j with $PB_j = 0$ or when $j = u - 1$

The second mutation operator is the push forward (very similar to initialization push forward). In this operator, we randomly select a point p_i from the route and define the push forward (PF_i) value as a random percentage of the time difference between the latest service time of the selected point and its service time: $PB_i = random(b_{p_i} - T^k_{p_i})$. We also push forward all the points after p_i in the route: $PF_{p_{j+1}} = max\{0, PF_{p_j} - w_{p_{j+1}}\}$, $i \leq j < u$. We stop the push forward operation when we reach a point j with $PF_j = 0$ or when $j = u - 1$.

The third mutation operator is the remove-insert operator. We randomly select a pair of pickup and delivery points to delete from the route (and mark their request as non-matched). After the delete operation, we perform a push backward operation in the positions of the deleted points. By the end of the delete operations, we try to insert as much pairs of pickup and delivery points as possible from the points of the non-matched requests.

The fourth mutation operator is the transfer mutation. In this operator, we randomly select a pair of pickup and delivery points from the route and try to insert them in another route. The last mutation operator is the swap mutation operator. In this operator, we swap a randomly selected point p_i in the route and its neighbor point p_{i+1}.

We noticed that the push forward and push backward mutation operators are the most important among the mutation operators. However, still the other three mutation operators participate in the quality of the solution.

3.2 The Insertion Heuristic

Given a solution from the genetic algorithm, the insertion heuristic answers each newly received offer or request by modifying the solution when possible. If a driver offer is received, then a route is added to the solution and the heuristic tries to match as much as possible from the non-matched riders' requests. If a rider's request is received, then the heuristic tries to insert it in one of the routes. Finding a match for a given request involves finding all possible insertions for the pickup and delivery points of the request in all routes and selecting the best insertion. The best insertion is defined as the one that adds the minimum value to the heuristic objective function. The heuristic objective function is the weighted sum of the total distance and time of vehicles' trips and the total time of the trips of the matched riders' requests. We do not consider the number of matched riders in the heuristic's objective function because each insertion possibility adds one rider.

To find a feasible insertion, the same insertion method that is used in solution initialization is used here. By the time of receiving a new request x, parts of the solution might have been executed (points already visited). Therefore we apply an additional constraint when finding an insertion position for the pickup point of the new request. The constraint states that the scheduled time of the point just before the candidate insertion position i of the new pickup in the route p of vehicle k should be greater than the arrival time inT_x of the new request x, $T^k_{p_{i-1}} > inT_x$.

4. EXPERIMENTATION AND RESULTS

In this section, we describe the proposed real world datasets for ridematching and the experimentation methodology and discuss the experimentation results.

4.1 Experimentation data

To test the behavior of the proposed algorithm for solving the dynamic RMPTW, we have utilized a travel and activity survey for northeastern Illinois conducted by Chicago

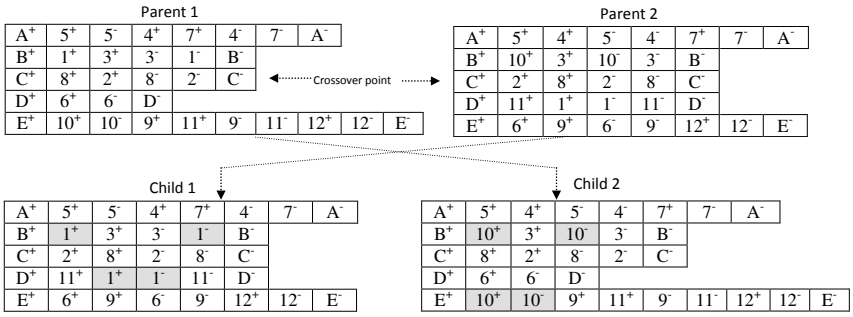


Figure 3: Single point crossover operator. Shaded points violate constraints (4) and (5)

Table 1: Problem instances. DMD = divers with maximum distance, TW= time window, DTD and DTT are the total direct travel distance and time respectively of the vehicles (drivers) without ridesharing.

Instance	No. drivers	No. riders	DMD	Earliest departure	Latest departure	TW ± min	DTD (km)	DTT (min)
RM744_R15	268	476	no	10:00	16:40	15	2621.8	2737
RM744_L15	268	476	yes	10:00	16:40	15	4468.2	4582
RM744_R60	268	476	No	10:00	16:40	60	2621.8	2737
RM744_L60	268	476	yes	10:00	16:40	60	4468.2	4582
RM698_R15	250	448	No	00:30	23:30	15	2451.9	2565
RM698_L15	250	448	yes	00:30	23:30	15	4602.6	4717
RM698_R60	250	448	No	00:30	23:30	60	2451.9	2565
RM698_L60	250	448	yes	00:30	23:30	60	4602.6	4717

Metropolitan Agency for Planning CMAP [1]. Data collection took place between January 2007 and February 2008 and covered a total of 10,552 households participant. A trip in the study is defined as a pair of source and destination positions (using latitude and longitude) with departure and arrival timestamps.

Our datasets are mainly based on two travel patterns. The first consists of a total of 698 trips (RM698) among them 469 trips with participants travel with their own vehicles and the rest rely on other forms of transportation. The second consists of a total of 744 trips (RM744) among them 533 trips with participants travel with their own vehicles and the rest rely on other forms of transportation. We select from the trips with participants traveling with their own vehicles what constitutes 36% of the total trips in each pattern to be as drivers and the rest as riders. The drivers are either selected randomly or those with the maximum travel distance. A time window is engineered for each trip. We consider a vehicle speed of 60km/h.

The distance between two points is measured using the Haversin formula [7] and the travel time between them is the ceil of the computed travel time (always integer). Table 1 shows the eight problem instances used in this study. For each one of the eight instances, we study the behavior of the algorithm in case of the percent of known riders' requests (KRR) by the time of executing the genetic algorithm is 20%,60% and 100% . We consider that all drivers' offers are known in advance by the time of executing the genetic algorithm.

4.2 Experiment settings

The parameters' settings (defined in Sec. 2) of the problem instances are $AT_i = 0$ and $BT_i = 1.3$ for all riders' requests and vehicles (drivers' offers), $C^k = 5$, $AD_i = 0$ and $BD_i = 1.3$ for all vehicles and $s_i = 0$ and $dm_i = 1$ for all riders' requests. The criteria weights are: $\alpha = 0.7$ and $\beta = \gamma = \delta = 0.1$.

[1]http://www.cmap.illinois.gov/travel-tracker-survey.

Table 2: Wilcoxon rank-sum test comparing different configurations G_i with 30 independent runs for each configuration. ▲ = row element is better than column element regarding the value of the objective function and the opposite for ▽.

crossover	mutation		G_1	G_2	G_3	G_4	G_5	G_6	G_7	G_8	G_9	G_{10}	G_{11}
1.0	1.0	G_0	–	–	▲	▲	▲	▲	–	▲	▲	▲	▲
1.0	0.7	G_1		–	▲	▲	▲	▲	▲	▲	▲	▲	▲
1.0	0.4	G_2			▲	▲	▲	▲	▲	▲	▲	▲	▲
1.0	0.1	G_3				▽	▽	–	▽	▽	▲	▽	–
0.7	1.0	G_4					▲	▲	–	–	▲	–	▲
0.4	1.0	G_5						▲	▽	▽	▲	▽	▲
0.1	1.0	G_6							▽	▽	▲	▽	–
0.7	0.7	G_7								–	▲	–	▲
0.7	0.3	G_8									▲	–	▲
0.7	0.1	G_9										▽	▽
0.4	0.7	G_{10}											▲
0.1	0.7	G_{11}											

The parameters' settings of the genetic algorithm are: population size = 100, number of generations = 100, crossover probability = 1.0 and mutation probability =0.4. The selection of the probabilities of the crossover and mutation is based on a study result summarized in Table 2. The algorithm is executed 30 independent times for each problem instance and percentage of KRR.

Experiments are performed on a computer with 4G RAM and 2.4GHz AMD 64bit dual core processor running Windows XP x64. JRE 1.6.0_22 is the runtime environment

4.3 Results discussion

Table 3 shows the results of the 30 independent runs of the algorithm for each instance and percentage of KRR (mean and standard deviation). The results include the value of the objective function and each of its components, the runtime of the genetic algorithm and the runtime per request of the insertion heuristic. To avoid scaling problems, the values of the different criteria are normalized before computing the

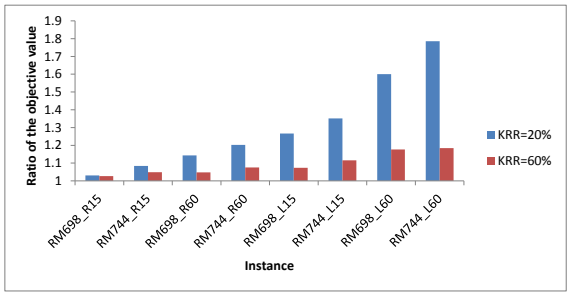


Figure 4: The ratio of the value of the objective function (OF) under the percentages 20% and 60% of the KRR to its value under percentage 100%.

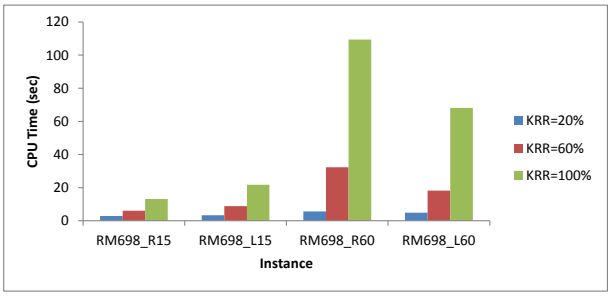


Figure 5: The average execution time of the genetic algorithm for instances RM698 under different percentages of KRR

value of the objective function. Note that the direct travel time and distance and the maximum allowed travel time and distance can provide us with upper and lower bounds for the total travel time and distance. The number of the riders' requests gives us an upper bound for the number of matched riders' requests.

We make use of figures to further discuss our findings. Figure 4 shows for each instance the ratio of the value of the objective function under percentages 20% and 60% of the KRR to its value under percentage 100%. In other words, it shows how worse the value of the objective function if we rely on the insertion heuristic than if we totally rely on the genetic algorithm.

We see that by decreasing the percentage of the known riders' requests at the time of executing the genetic algorithm we get wore final values of the objective function (larger ratio). Also we notice that the ratio is larger for instances with $DMD = yes$. This is because for instances with $DMD = yes$, the vehicles travel for longer distances and are supposed to serve larger number of riders requests. Therefore, there is larger room for optimization considering the ordering and timing of the riders' points where the insertion heuristic gets stuck in local optima and the genetic algorithm performs better. Within each group with $DMD = yes$ or $DMD = no$, instances with time windows of $\pm$ 60 minutes showed larger ratio than instances with time windows of $\pm$ 15 minutes. Increasing the time windows increases the search space by increasing the matchable riders' requests list of each vehicle. Also we notice that for each two instances with the same DMR and the same time window, we notice that the ratio is larger for the instances $RM744$.

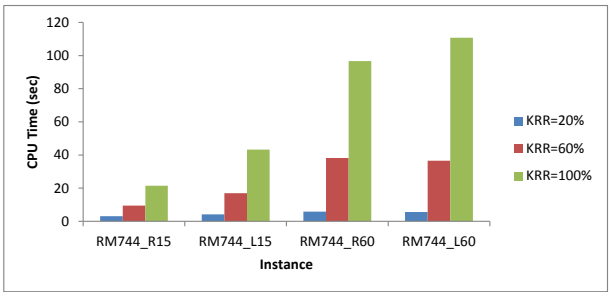


Figure 6: The average execution time of the genetic algorithm for instances RM744 under different percentages of KRR

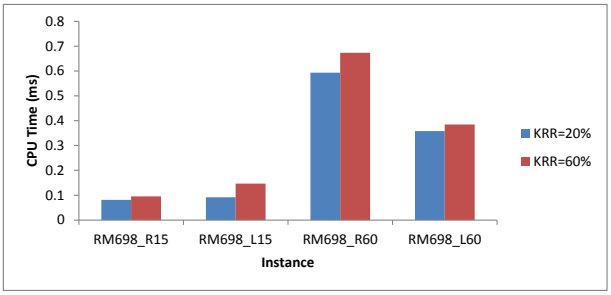


Figure 7: The average execution time of the insertion heuristic per request for instances RM698 under different percentages of KRR

This is because for instances $RM744$ we have more trips and the trips' departures are closer to each other as compared to instances $RM698$. This means that there is larger search space and riders' requests can be served with larger number of vehicles.

Figure 5 shows the runtime of the genetic algorithm for the instances RM698 under different percentages of the KRR. Obviously the runtime of the genetic algorithm increases with the increase of the percentage of the KRR at the time of its execution. Also we notice that the length of the time window plays a major rule in the runtime of the genetic algorithm. Longer time windows increase the size of the list of the matchable riders' requests of each vehicle which is used often after each crossover and remove-insert mutation to try to match additional non-matched requests. This leads

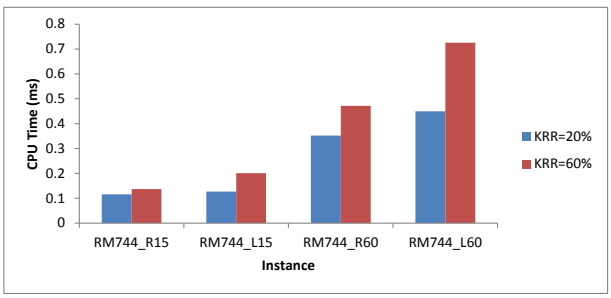
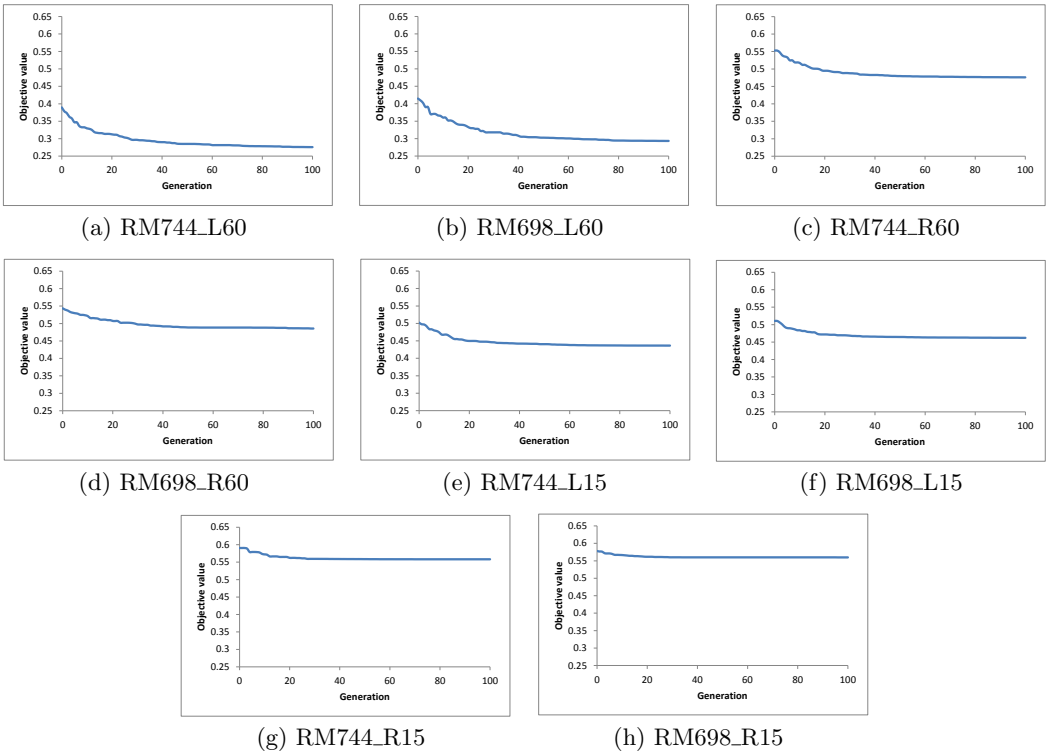


Figure 8: The average execution time of the insertion heuristic per request for instances RM744 under different percentages of KRR

Table 3: Results summary. %KRR= percentage of the known riders' requests, OF= objective function, No. MRR= No. of matched riders' requests, GRT= genetic alg. run time, HRT = heuristic run time per request.

Instance	%KRR	OF	No. MRR	vehicles' distance	vehicles' time	riders' time	GRT (sec)	HRT (ms)
RM744_R15	20	$0.605_{\sigma=0.0050}$	$105.333_{\sigma=4.671}$	$2786.127_{\sigma=8.524}$	$2932.833_{\sigma=10.453}$	$834.233_{\sigma=38.518}$	$3.15_{\sigma=0.17}$	$0.116_{\sigma=0.024}$
	60	$0.585_{\sigma=0.0030}$	$119.367_{\sigma=2.168}$	$2786.254_{\sigma=4.507}$	$2947.667_{\sigma=4.643}$	$901.267_{\sigma=9.889}$	$9.443_{\sigma=0.251}$	$0.137_{\sigma=0.039}$
	100	$0.558_{\sigma=0.0010}$	$138.7_{\sigma=0.69}$	$2784.907_{\sigma=2.259}$	$2953.8_{\sigma=2.701}$	$952.967_{\sigma=6.374}$	$21.509_{\sigma=0.431}$	$0.0_{\sigma=0.0}$
RM744_L15	20	$0.585_{\sigma=0.0020}$	$115.8_{\sigma=2.072}$	$4736.434_{\sigma=6.795}$	$4902.6_{\sigma=8.048}$	$1026.8_{\sigma=6.6}$	$4.188_{\sigma=0.073}$	$0.127_{\sigma=0.012}$
	60	$0.483_{\sigma=0.0050}$	$199.2_{\sigma=4.261}$	$4849.71_{\sigma=15.703}$	$5072.8_{\sigma=15.01}$	$1194.167_{\sigma=12.662}$	$17.004_{\sigma=0.705}$	$0.201_{\sigma=0.041}$
	100	$0.433_{\sigma=0.0030}$	$234.9_{\sigma=2.856}$	$4872.177_{\sigma=15.818}$	$5104.2_{\sigma=16.138}$	$1298.133_{\sigma=17.043}$	$43.301_{\sigma=1.628}$	$0.0_{\sigma=0.0}$
RM744_R60	20	$0.576_{\sigma=0.0050}$	$134.567_{\sigma=4.295}$	$2820.904_{\sigma=9.317}$	$2985.433_{\sigma=11.635}$	$1019.567_{\sigma=24.415}$	$5.892_{\sigma=0.472}$	$0.352_{\sigma=0.023}$
	60	$0.515_{\sigma=0.0040}$	$182.667_{\sigma=2.675}$	$2861.524_{\sigma=8.328}$	$3050.867_{\sigma=8.755}$	$1203.467_{\sigma=23.581}$	$38.229_{\sigma=1.82}$	$0.472_{\sigma=0.048}$
	100	$0.479_{\sigma=0.0040}$	$207.933_{\sigma=2.502}$	$2863.787_{\sigma=11.109}$	$3062.367_{\sigma=10.88}$	$1270.1_{\sigma=24.701}$	$96.662_{\sigma=4.273}$	$0.0_{\sigma=0.0}$
RM744_L60	20	$0.484_{\sigma=0.0070}$	$197.2_{\sigma=6.172}$	$4794.537_{\sigma=19.037}$	$4997.067_{\sigma=21.695}$	$1384.2_{\sigma=27.165}$	$5.641_{\sigma=0.321}$	$0.45_{\sigma=0.029}$
	60	$0.321_{\sigma=0.0060}$	$326.5_{\sigma=4.395}$	$4990.478_{\sigma=13.664}$	$5278.2_{\sigma=13.531}$	$1660.9_{\sigma=27.795}$	$36.614_{\sigma=2.223}$	$0.726_{\sigma=0.043}$
	100	$0.271_{\sigma=0.0050}$	$363.933_{\sigma=3.316}$	$5010.591_{\sigma=13.59}$	$5316.367_{\sigma=14.538}$	$1802.667_{\sigma=33.336}$	$110.767_{\sigma=8.226}$	$0.0_{\sigma=0.0}$
RM698_R15	20	$0.575_{\sigma=0.0030}$	$96.8_{\sigma=2.821}$	$2543.046_{\sigma=7.241}$	$2670.867_{\sigma=8.838}$	$557.8_{\sigma=10.028}$	$2.84_{\sigma=0.023}$	$0.081_{\sigma=0.01}$
	60	$0.573_{\sigma=0.0030}$	$110.533_{\sigma=3.096}$	$2593.739_{\sigma=8.929}$	$2730.133_{\sigma=9.976}$	$671.8_{\sigma=13.088}$	$6.072_{\sigma=0.156}$	$0.095_{\sigma=0.034}$
	100	$0.558_{\sigma=0.0010}$	$124.4_{\sigma=1.083}$	$2605.949_{\sigma=5.883}$	$2750.433_{\sigma=7.149}$	$685.467_{\sigma=8.913}$	$13.079_{\sigma=0.943}$	$0.0_{\sigma=0.0}$
RM698_L15	20	$0.584_{\sigma=0.0080}$	$106.6_{\sigma=6.859}$	$4796.58_{\sigma=21.38}$	$4947.433_{\sigma=26.087}$	$1050.233_{\sigma=17.701}$	$3.323_{\sigma=0.057}$	$0.092_{\sigma=0.018}$
	60	$0.495_{\sigma=0.0030}$	$174.3_{\sigma=1.754}$	$4973.511_{\sigma=12.49}$	$5170.5_{\sigma=13.455}$	$1198.4_{\sigma=13.827}$	$8.795_{\sigma=0.361}$	$0.147_{\sigma=0.018}$
	100	$0.461_{\sigma=0.0030}$	$198.267_{\sigma=1.459}$	$4983.294_{\sigma=11.52}$	$5195.0_{\sigma=12.565}$	$1264.9_{\sigma=10.888}$	$21.718_{\sigma=1.657}$	$0.0_{\sigma=0.0}$
RM698_R60	20	$0.551_{\sigma=0.0060}$	$135.633_{\sigma=4.963}$	$2651.593_{\sigma=14.225}$	$2799.333_{\sigma=17.034}$	$786.4_{\sigma=46.253}$	$5.657_{\sigma=0.23}$	$0.593_{\sigma=0.021}$
	60	$0.505_{\sigma=0.0030}$	$170.6_{\sigma=2.715}$	$2677.996_{\sigma=8.112}$	$2841.667_{\sigma=8.42}$	$1054.267_{\sigma=37.865}$	$32.277_{\sigma=2.622}$	$0.673_{\sigma=0.048}$
	100	$0.482_{\sigma=0.0030}$	$187.867_{\sigma=2.68}$	$2679.89_{\sigma=13.82}$	$2853.967_{\sigma=15.096}$	$1014.733_{\sigma=41.151}$	$109.378_{\sigma=10.636}$	$0.0_{\sigma=0.0}$
RM698_L60	20	$0.472_{\sigma=0.0090}$	$194.067_{\sigma=6.011}$	$4937.461_{\sigma=15.07}$	$5130.933_{\sigma=17.16}$	$1448.0_{\sigma=36.303}$	$4.824_{\sigma=0.13}$	$0.358_{\sigma=0.017}$
	60	$0.347_{\sigma=0.0060}$	$291.667_{\sigma=3.037}$	$5140.778_{\sigma=14.842}$	$5398.133_{\sigma=17.627}$	$1681.267_{\sigma=27.566}$	$18.235_{\sigma=0.983}$	$0.385_{\sigma=0.04}$
	100	$0.295_{\sigma=0.0070}$	$326.067_{\sigma=3.855}$	$5158.488_{\sigma=21.662}$	$5438.733_{\sigma=22.044}$	$1798.033_{\sigma=38.815}$	$68.063_{\sigma=3.992}$	$0.0_{\sigma=0.0}$



(a) RM744_L60 (b) RM698_L60 (c) RM744_R60

(d) RM698_R60 (e) RM744_L15 (f) RM698_L15

(g) RM744_R15 (h) RM698_R15

Figure 9: The best fitness at different generations of the genetic algorithm for the instances with KRR=100%

to increase the runtime of the genetic algorithm. Figure 6 shows a similar runtime pattern of the genetic algorithm on the instances RM744.

Figures 7 and 8 show the run time of the insertion heuristic per rider's request on the instances RM698 and RM744 under different percentages of KRR respectively. The runtime increases for instances with higher percentages of KRR as the solution of the genetic algorithm contains routes with more served requests. This results in larger number of possible insertions which needs to be considered by the heuristic.

It takes the insertion heuristic fractions of millisecond to answer a request on all datasets and percentages of KRR which is a real-time answer and in the worst case it takes the genetic algorithm 110.767 seconds. The runtime of the genetic algorithm is justifiable as we run it only once per time period and we rely on the insertion heuristic for the rest of the period. The results in Table 3 and Figure 4 indicate that the genetic algorithm helps to better optimize the objective function.

Figure 9 shows the optimization of the objective function by the genetic algorithm under different generations for each instance with all riders' requests being known (KRR = 100%). The most optimization is for instances with $DMD = yes$ and for time windows ± 60 as in Figures 9(a) and 9(b). Figure 9(h) shows the least optimization for the instance RM698_R15 with time window ± 15 and $DMD = no$ where the algorithm converges after near about 20 generations.

By dividing the entry of the vehicles' distance in table 3 on the DTD entry of the corresponding instance in table 1 we get the percentage of the travel distance (with serving riders) relative to the distance that would have been traveled without serving riders. The same applies for the travel time. For example, the DTD = 2451.9km for the instance RM698_R15 and the vehicles' distance = 2543.046km at KRR =20% for the same instance, so the total vehicle's travel distance that includes serving in average 96.8 riders is 1.037% of the DTD.

To see how the trips of the vehicles look like, we have created a GPX file for some of the vehicles' trips and used a GPX visualizer to visualize it on Google maps. The visualization result is provided in Figure 10.

5. CONCLUSIONS AND OUTLOOK

In this work, we have proposed a genetic and insertion heuristic algorithm for solving the dynamic ridematching problem with time windows. In addition, we have introduced eight datasets extracted from real world data for testing the behavior of the proposed algorithm. We conclude that the proposed algorithm can successfully solve the dynamic ridematching problem by providing answers to ridesharing requests in real-time. The algorithm is flexible and can be easily tuned to balance between the solution's quality and the responsiveness of the algorithm.

There are many interesting ideas to extend this work. First, we might move to a more realistic scenario. This implies adding more constraints such as limiting the number of pickup and delivery operations for each driver, putting a maximum detour distance per rider's request and considering realistic travel times and routes using Google APIs. An interesting direction is to extend the problem definition to solve the ridematching problem with time windows in dynamic multihop ridesharing where riders can share rides with multiple drivers.

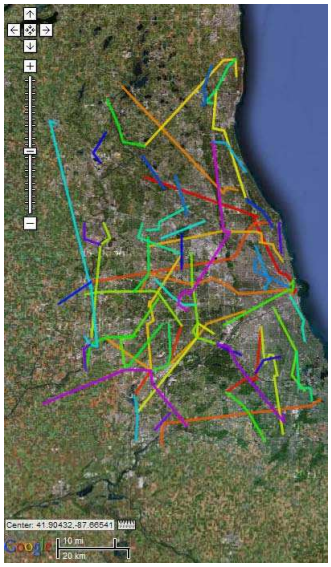


Figure 10: A color coded visualization of some vehicles' trips for instance RM744_L60 with KRR = 100%. The used GPX visualization service is: http://www.gpsvisualizer.com.

For the reproducibility of the results, the instances and the source code will be available on the authors' website.

6. REFERENCES

[1] Agatz, N., Erera, A., Savelsbergh, M., and Wang, X. Sustainable passenger transportation: Dynamic ride-sharing. Tech. rep., Erasmus Research Inst. of Management (ERIM), Erasmus Uni., Rotterdam, 2010.

[2] Agatz, N. A., Erera, A. L., Savelsbergh, M. W., and Wang, X. Dynamic ride-sharing: A simulation study in metro atlanta. *Transportation Research Part B: Methodological 45*, 9 (2011), 1450 – 1464.

[3] Baugh, J., Kakivaya, G., and Stone, J. Intractability of the dial-a-ride problem and a multiobjective solution using simulated annealing. *Engineering Optimization 30*, 2 (1998), 91–123.

[4] Chan, N., and Shaheen, S. Ridesharing in north america: Past, present, and future. *Transportation Research Board Annual Meeting* (2011).

[5] Jaw, J., Odoni, A., Psaraftis, H., and Wilson, N. A heuristic algorithm for the multi-vehicle advance request dial-a-ride problem with time windows. *Transportation Research Part B: Methodological 20*, 3 (1986), 243 – 257.

[6] Jorgensen, R. *Dial-a-Ride*. PhD thesis, Technical University of Denmark, 2002.

[7] Sinnott, R. W. Virtues of the haversine. *Sky and Telescope, Vol.68:2, P.158, 1984 62*, 2 (1984), 158.

[8] Solomon, M. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research 35* (1987), 254–265.

[9] Toth, P., and Vigo, D., Eds. *The vehicle routing problem*. Siam, 2002.