Real-time ride-sharing substitution service in multi-modal public transport using buckets

Kamel Aissat and Sacha Varone

[1]University of Lorraine - LORIA, Nancy, France. kamel.aissat@loria.fr
[2]University of Applied Sciences and Arts Western Switzerland (HES-SO), HEG Genève, Switzerland. sacha.varone@hesge.ch

Abstract We consider a mix transportation problem, which allows to combine a multi-modal public and a ride-sharing transports, in a dynamic environment. The main idea of our approach consists in labelling interesting nodes of a geographical map with information about either riders or drivers, in so-called buckets. Based on the information contained in these buckets, we compute admissible ride-sharing possibilities. To restrict the needed amount of memory, among the different stops along a public transportation path, we only consider the transshipment nodes, where travellers have to make a change between two modes. Each of those stops are potential pick-up or drop-off stops for ride-sharing. We consider a drivers' maximal waiting time, as well as the maximal driving detour time depending on the actual drive. Each new drive activates a search for new ride-sharing of existing riders. Each new ride activates another process which searches for potential drivers. Among all admissible ride-sharing possibilities, only those which best improve the earliest arrival time are selected. We provide numerical results using real road network of the Lorraine region (FR) and real data provided by a local company. Our numerical experiment shows a running time of a few seconds, suitable for a new real-time transportation application.

Keywords: Multi-modal, Ride-sharing, Public transportation, Real-time

1 Introduction

Mobility within a region can be done using different transportation modes, either public or private, which might be a function of the preferences or of the possibilities of the commuter. We propose an algorithmic approach to the real-time multi-modal earliest-arrival problem (EAP) in urban transit network, using public transportation and ride-sharing, so that the trip duration is minimized. We start from using public transportation, included walking sub-paths and then, considering (quasi) real-time ride-sharing opportunities, try to transfer sub-paths from public modality to ride-sharing modality. For that purpose, we consider riders, who provide requests to go from one location to another one, and drivers who offer ride-sharing. Riders' goal is to find a quickest trip in a dynamic context that effectively combines the use of several modes of transportation.

The search for a quickest or shortest path has been well studied in the literature, and has now very efficient algorithms to solve it, in the order of a millisecond for a continental trip: the authors in [2] survey recent advances in algorithms for route planning. Computation of shortest paths are key components in route planning, but not sufficient to deal with multi-modal problems, since they involve multi-criteria paths, transition or waiting times, etc. which is usually not taken into account in shortest paths on pure road networks. For example, the authors in [1] include several features in their objective function, so that they focus on the modal change node, and propose a two-step algorithm for computing multi-modal routes. An exact algorithm considering arrival and departure time-windows has been proposed in [5]. Multi-criteria search by computing the Pareto set is done in [4].

In public transportation problems, solving earliest-arrival problems (EAP) knowing the departure and arrival station, departure time and timetable information is reviewed in [6,7]. Timetabling information and EAP solving is nowadays often available on-line for users. Real-time journey using ride-sharing opportunities faces the commuting point problem: it has to be decided where the pick-up and drop-off locations have to be located. [3] define this as the 2 synchronization points shortest path problem (2SPSPP). The time complexity of their approach prevents its use in real-time ride-sharing. In our approach, we first find a quickest path using public transportation and consider its different transit stops. We then only allow pick-up and drop-off around those stops to reduce the search space. We then compute a potential drivers' list and further check for admissible drivers. Finally, if a ride-sharing is possible, we update the routes for both rider and driver.

2 Problem description

The problem to be solved is presented as follows: a user u wishes to go from an origin point O_u to a destination point D_u. He might use either public transportation, ride-sharing or a combination of both. All public transport timetabling are assumed to be known or at least accessible easily. In France, one might use the Navitia API [1]. The main idea of our approach consists in storing for each driver and for each rider information about his travel if passing trough a pick-up or drop-off stop. Each new ride arriving in the system launches a request for a public transportation path to determine the different stops. Each of those stops are considered as potential stops for ride-sharing. A search for drivers around those stops is then run, so that ride-sharing would improve the earliest arrival time to destination, under time-constraints: driver's waiting time, and detour time depending on the actual drive. Another process is run for each new drive, so that information about potential pick-up or drop-off stops are stored.
The network is represented as a directed graph model $G(V, E)$, in which V represents a set of nodes and E the set of arcs. Nodes model intersections and arcs depict street segments. A non negative weight is associated to each arc:

[1] `http://www.navitia.io`

its cost. A path that minimizes the sum of its cost is called a quickest/shortest path. We define a *stop* to be the location for which a transit or road node exists. Stops correspond to bus stops, subway stations, parking, etc. In this multi-modal context, G is modeled as the union of multiple networks, representing different travel modes (bike, car, public transport, ...). Our model uses public transport network and road network. TABLE 1 lists the notations used throughout this paper.

Table 1. A LIST OF NOTATIONS

Notation	Definition
$s \rightsquigarrow e$	Driving quickest path between s and e.
$\delta(s,e)$	Duration of a quickest path between s and e.
$d(s,e)$	Distance as the crow flies between s and e.
$\hat{\delta}(s,e)$	Estimated smallest duration from s to e, such that $\hat{\delta}(s,e) = \frac{d(s,e)}{v_{\max}}$, where $v_{\max}$ is the maximal speed.
λ_k	Detour coefficient, $\lambda_k \geq 1$.
$\tau(x)_{ab}$	Time required for moving from modality a to b at x.
$t_a^u(x,m)$	Arrival time at x using the m transport modality for user u.
$t_d^u(x,m)$	Departure time at x using the m transport modality for user u.
$w_{\max}^k$	Maximum waiting time for driver k at pick-up point.
L_x	List of potential drivers at x as pick-up stop.

We will further note as p the public transportation modality, and as c the car modality. The ride-sharing process is the following: a user (rider/driver) generates a request through his smartphone. The request is sent to a central server. Potential drivers/riders are then contacted. Those might accept or reject the request. If several drivers/riders have accepted the request, then the rider/driver might choose which one he prefers.

A driver k is characterized by his position O_k at current time t_0^k and his destination D_k. A driving route from its origin to its destination through a stop x_i has to respect a maximum waiting time at x_i, and a maximum detour time, the latter being a multiple of a shortest duration between the driver's origin and destination. An admissible ride-sharing route for the driver and the rider is defined as follows:

Definition 1. (Admissible ride-sharing)
We say that a path $x_i \rightsquigarrow x_j$ is an admissible ride-sharing for a driver k and a rider u if and only if a maximal driver's waiting time $w_{\max}^k$ at a pick-up location (1), a maximum driver's detour (2) and a rider's latest arrival time (3) constraints are satisfied, i.e,

$$t_a^u(x_i,p) + \tau(x_i)_{pc} - t_a^k(x_i,c) \leq w_{\max}^k \qquad \textit{waiting} \qquad (1)$$

$$\delta(O_k,x_i) + \max\left\{t_a^u(x_i,p) + \tau(x_i)_{pc} - t_a^k(x_i,c), 0\right\} + \delta(x_i,x_j) + \delta(x_j,D_k) \leq \lambda_k \delta(O_k,D_k) \qquad \textit{detour} \qquad (2)$$

$$\max\left\{t_a^u(x_i,p) + \tau(x_i)_{pc}, t_a^k(x_i,c)\right\} + \delta(x_i,x_j) \leq t_a^u(x_j,p) \qquad \textit{arrival} \qquad (3)$$

Constraint (1) is composed of the arrival time $t_a^u(x_i,p)$ at x_i by the traveller using public transportation plus the transshipment time $t_a^k(x_i,c)$ to walk from the stop towards the meeting point with the driver, and the arrival time $t_a^k(x_i,c)$

at x_i for the driver. If their difference is positive, then the driver have to wait. Else, the rider has to wait. Constraint (2) is the sum of the duration $\delta(O_k, x_i)$ for a driver k to reach stop x_i, plus a potential waiting time, plus the ride duration $\delta(x_i, x_j)$, plus the duration $\delta(x_j, D_k)$ to reach its final destination. Constraint (3) expresses that the total duration for the rider to reach stop x_j using a ride-sharing mode between x_i and x_j should not exceed its arrival time at x_j using only public transportation.
Each constraint (1), (2) and (3) takes into account $\tau(x_i)_{pc}$ the time required for moving from the public transportation modality to the ride-sharing modality.

For each new offer $O_k D_k$ in the system, related information about time, driver k are stored at each potential stop where driver k might meet a rider, while respecting its *detour* constraint.

3 Methodology

In a dynamic environment, announcements of riders and drivers are arriving continuously at arbitrary times. Hence, matches between riders and drivers have to be computed many times during the day which implies that future requests are unknown during each planning cycle. So, the decision to wait or not for other requests depends on the choice of the user. The underlying application assumes that GPS coordinates of drivers can be known, or at least estimated, continuously. We hereby describe the process for each new driver and each new rider entering the system, and how we remove outdated information. We use a double list of so-called buckets which contain information about riders and drivers: the system maintains for each potential pick-up or drop-off node, information about which driver and/or which rider is/are potential ride-sharing members. In order to reduce the amount of memory, considered riders' journey are only those belonging to a public transportation mode.

Definition 2. ***(Drivers bucket** $B_d(v)$ $v \in V$)*
The drivers bucket $B_d(v)$ *associated with a node v is the set of triplets, composed of a driver k, his earliest arrival time at v and his latest departure time from v, such that the drivers might pick-up riders at node v without violating the detour constraint (2).*

$$B_d(v) := \{(k, t_a^k(v,c), t_d^k(v,c)) | \textit{ constraints } (2) \textit{ is satisfied } \}. \tag{4}$$

A driver k is contained in $B_d(v)$ if and only if the duration of his trip via v does not exceed a maximal detour: $\delta(O_k, v) + \delta(v, D_k) \leq \lambda_k \delta(O_k, D_k)$. The entries of this driver k is added in the drivers bucket $B_d(v)$ by descending order according to his latest departure time $t_d^k(v,c) = t_o^k + \lambda_k \delta(O_k, D_k) - \delta(v, D_k)$ (the latest time is the first in the heap $B_d(v)$).

Definition 3. ***(Riders bucket** $B_r(v)$ $v \in V$)*
The riders bucket $B_r(v)$ *associated with a node v is the set of triplets, composed of a rider u, his arrival time at v and his departure time from v, using public transportation.*

$$B_r(v) := \{(u, t_a^u(v,p), t_d^u(v,p))\}. \tag{5}$$

In other words, $B_r(v)$ contains the list of riders and their pick-up time windows. Entries in $B_r(v)$ are sorted by ascending order according to arrival times.

We also maintain the OD paths with the time related information for each driver and each rider until she leaves the system.

3.1 Adding a driver's offer

Each new driver entering into the system generates new ride-sharing opportunities. Let's denote by k the driver, O_k his origin and D_k his destination. We compute the integration of those new ride-sharing offers by determining all possible pick-up/drop-off locations. For this purpose we use a modified bidirectional $A^\star$ algorithm (BA). The bidirectional search algorithm works with two simultaneous searches of the shortest path, one from the source node O_k and the second reversely from the target node D_k simultaneously, until "the two search frontiers meet". As in $A^\star$, BA search is guided by a heuristic function that estimates the remaining duration to the target (in the forward search) and another from the source (in the backward search). We maintain two priority queues, one, as in simple $A^\star$, from the source O_k, denoted by $\mathcal{Q}_1$, and another one from the target D_k, denoted by $\mathcal{Q}_2$. The latter is a forward search in the reversed graph G^{-1}, in which each arc (u, v) of G is replaced by (v, u) in G^{-1}.

Thus, for a given node v, the cost function used in the direct search ($\mathcal{Q}_1$) is $\delta(O_k, v) + \hat{\delta}(v, D_k)$, whereas in the reverse search ($\mathcal{Q}_2$), we use the cost function $\hat{\delta}(O_k, v) + \delta(v, D_k)$. (see figure 1)
At each iteration, we select the node with the smallest overall tentative cost, either from the source O_k or to the target D_k. In order to do this, a simple comparison between the minima of the two priority queues $\mathcal{Q}_1$ and $\mathcal{Q}_2$ is sufficient.

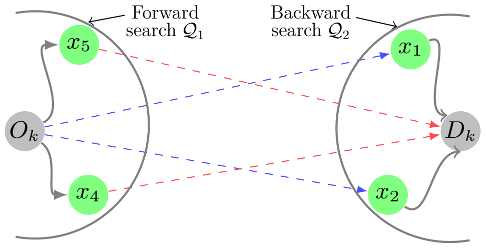
Forward search $\mathcal{Q}_1$
Backward search $\mathcal{Q}_2$
x_5
x_1
O_k
D_k
x_4
x_2

Figure 1. Situation of BA algorithm before finding the duration $\delta(O_k, D_k)$. The two searches are guided by estimates of the remaining durations (dashed red lines for the first search $\mathcal{Q}_1$ and dashed blue lines for the second search $\mathcal{Q}_2$). Thus, only the left half-disc around O_k is visited for the first search $\mathcal{Q}_1$, and the right half-disc around D_k for the second search $\mathcal{Q}_2$. The solid lines correspond to the exact durations determined after each labelling from either $\mathcal{Q}_1$ or $\mathcal{Q}_2$.

Once we select a node v in one queue that has already been selected in the other queue, we get the first tentative cost of a shortest path from O_k to D_k. Its cost is the cost of the path found by the forward search from O_k to v, plus the cost of the path found by the backward search from v to D_k.

Even when the two parts of the tentative solution of the forward and the backward meet each other, the concatenated solution path is not necessarily optimal. To guarantee optimality, we must continue until the tentative cost of the current minima of the queues is greater than the current tentative shortest path cost (which then corresponds to the cost of the shortest path). Once the shortest path O_k to D_k is found, then $\lambda_k \delta(O_k, D_k)$ is known. At this level, the search doesn't stop but continue until the cost of the current node v is greater than $\lambda_k \delta(O_k, D_k)$. (see figure 2)

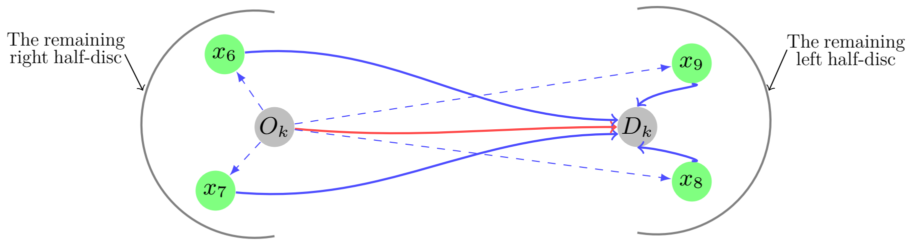

Figure 2. Duration $\delta(O_k, D_k)$ is found by the BA algorithm. The search space still expands until all nodes with label smaller than $\lambda_k \delta(O_k, D_k)$ are labelled. At this point, the remaining nodes to be labelled are situated in the right half-disc around the target O_k or in the left half-disc around D_k. The red line is the duration $\delta(O_k, D_k)$. Only nodes that satisfy the detour constraint are kept.

In our case, we set $\lambda_k = 1.2$, which means that all customers allow trip duration at most 20% greater than that of a quickest path. For each node v settled by the two queues $\mathcal{Q}_1$ and $\mathcal{Q}_2$, the *detour* constraint (2) is verified. If valid, then the new element is inserted into the *bucket* $B_d(v)$ by decreasing order according to their latest departure times. Then the riders bucket $B_r(v)$ is scanned in order to find the best rider with whom the driver will share his trip. Algorithm 1 is run for each node settled by the two queues $\mathcal{Q}_1$ and $\mathcal{Q}_2$. Rider $u^\star$ and its associated pick-up $x_{i^\star}$ and drop-off $x_{j^\star}$ are those that generate the greatest time-savings for the rider. The different steps for adding a carpool offer are defined in Algorithm 2.

Algorithm 1 Scan the riders bucket $B_r(v)$

```
Require: Offer $k$, $B_r(v)$.
Ensure: Best rider $u^\star$, Best drop-off stop $x_{j^\star}$.
        $\sigma^\star$ : Best time-savings generated for rider $u^\star$ having $v$ as pick-up stop.
1: Initialization: $u^\star \leftarrow -1$, $\sigma^\star \leftarrow 0$, $x_{j^\star} \leftarrow -1$
2: for all $u$ in $B_r(v)$ do
3:   for each stop $x_j$ in path of rider $u$ situated after $v$ do
4:     $\sigma = t_a^u(x_j, p) - (\max\{t_a^k(v, c), t_a^u(v, p) + \tau(v)_{pc}\} + \delta(v, x_j))$
5:     if $k \in B_d(x_j)$ and $(v, x_j)$ forms an admissible ride-sharing path for driver $k$
       and rider $u$ and $\sigma > \sigma^\star$ then
6:       $x_{j^\star} \leftarrow x_j$, $u^\star \leftarrow u$, $\sigma^\star \leftarrow \sigma$
7:     end if
8:   end for
9: end for
```

Algorithm 2 Adding carpool offer k

```
Require: Offer $k$, detour coefficient $\lambda_k$.
Ensure: Best rider $u^\star$, Best pick-up $x_{i^\star}$, Best drop-off $x_{j^\star}$, Update drivers' list bucket
 1: Initialization : $\mathcal{Q}_1.Insert(O_k)$ and $\mathcal{Q}_2.Insert(D_k)$
                     $Meeting\_locations\_list \leftarrow \emptyset$, $Search$ = true, $\delta(O_k, D_k) \leftarrow \infty$
 2: while $Search == True$ do
 3:   $Current\_node \leftarrow \min(\mathcal{Q}_1.GetFirst(), \mathcal{Q}_2.GetFirst())$
 4:   Mark $Current\_node$ as visited and remove it from appropriate queue
 5:   if $Current\_node$ is marked as visited by both $\mathcal{Q}_1$ and $\mathcal{Q}_2$ then
 6:     $Meeting\_locations\_list \leftarrow Meeting\_locations\_list \cup \{Current\_node\}$
 7:     if $\delta(O_k, Current\_node) + \delta(Current\_node, D_k) < \delta(O_k, D_k)$ then
 8:       Update $\delta(O_k, D_k) \leftarrow \delta(O_k, Current\_node) + \delta(Current\_node, D_k)$
 9:     end if
10:     if $\min\big(\delta(O_k, Current\_node), \delta(Current\_node, D_k)\big) > \lambda_k \delta(O_k, D_k)$ then
11:       $Search$ = False
12:     end if
13:   end if
14:   Update the neighbors' cost of the $Current\_node$ and $insert$ them in appropriate
      queue according to cost function $\big(\delta(O_k, Current\_node) + \hat{\delta}(Current\_node, D_k)$
      for $\mathcal{Q}_1$ and $\hat{\delta}(O_k, Current\_node) + \delta(Current\_node, D_k)$ for $\mathcal{Q}_2\big)$.
15: end while
16: for all $v$ in $Meeting\_locations\_list$ do
17:   if $\delta(O_k, v) + \delta(v, D_k) \le \lambda_k \delta(O_k, D_k)$ then
18:     $t_a^k(v,c) = t_o^k + \delta(O_k, v)$, $t_d^k(v,c) = t_o^k + \lambda_k \delta(O_k, D_k) - \delta(v, D_k)$
19:     $B_d(v) := B_d(v) \cup \{(k, t_a^k(v,c), t_d^k(v,c))\}$
        {insert operation in descending order according to $t_d^k(v,c)$}
20:     Scan the riders bucket $B_r(v)$ using Algorithm 1
21:     Update best rider $u^\star$, best pick-up $x_{i^\star}$ and best drop-off $x_{j^\star}$
22:   end if
23: end for
```

3.2 Removing outdated information in buckets

Each time a node v is visited, the bucket is updated so that all outdated information are removed. Since a bucket is sorted in decreasing time, the removal of its outdated elements can be done in linear time. The buckets are reinitialized each 24 hours in our application.

3.3 Adding a rider's request

Each new ride entering the system corresponds to an origin-destination O_uD_u path, with an initial time t_0. The first action is to provide a public transportation path. A second action is to determine if there are some ride-sharing opportunities which decrease the arrival time to destination. The different steps are described below:

Step 1 The first step gets the multi-modal public transportation path $P = O_u, x_1, \ldots, x_{nbs}, D_u$ starting at time t_{O_u}, using a public transportation API. For each stop x_i in P, we associate arrival time $t_a^u(x_i, p)$ and departure time $t_d^u(x_i, p)$, as illustrated by Figure 3.

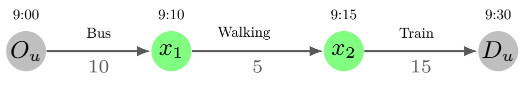

Figure 3. A rider u start travelling at time $t_u = 9{:}00$ from his origin O_u, which is also the first stop x_0, to his destination D_u. Path P is composed of two transshipment stops $\{x_1, x_2\}$.

Then possible driving substitution sub-paths along P are computed. Durations of these driving paths will then be used to test if there is an improvement for the rider by switching from a public transportation to a ride-sharing. Only driving paths which decrease the earliest arrival time to stops x_j or D_u are kept. A pick-up time-window on each stop x_i is determined by the arrival time and departure time at x_i, as illustrated by Figure 4.

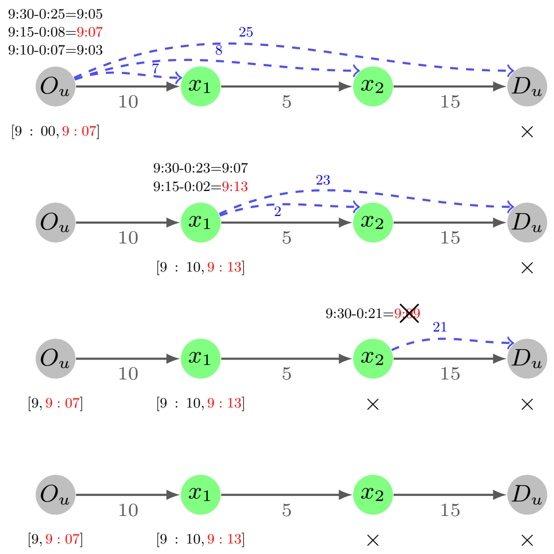

Figure 4. Determination of pick-up time-window for each potential stop. The red numbers above the nodes are the calculation of the latest departure times, so that arrival time at x_j is the same as the arrival time using public transportation. The interval below a node is composed of the earliest and the latest departure time. There is no time-window associated with x_2, since it is not possible to decrease any arrival time from this stop using ride-sharing.

Step 2 The goal of this step is to find a set of drivers able to pick-up and drop-off the rider at some stops, within their associated time-windows. Two constraints for the benefit of drivers are considered: a maximum waiting time and a maximum detour. This phase also includes the determination of the drivers' quickest paths towards each stop, computed reversely from each pick-up stop. The information contained in the drivers bucket $B_d(v)$ allows to restrict the computation of the quickest paths. Thus, for each stop x_i a list of potential drivers L_{x_i} is created.

Step 3 This step searches for ride-sharing opportunities and select the best one. For each possible substitution path $x_i \rightsquigarrow x_j$, determined from the set of drivers in $L_{x_i} \cap L_{x_j}$, its *admissibility* is verified. Among all admissible ride-sharing $x_i \rightsquigarrow x_j$, only those allowing maximal savings are chosen. Algorithm 3 explains this process, which is illustrated by Figure 5. The same operation is repeated for each $x_i \in P$, and the best pick-up stop $x_{i^\star}$, the best drop-off stop $x_{j^\star}$ as well as the driver $k^\star$ that generates the most positive time-savings are selected. Algorithm 4 describes it.

Algorithm 3 Scan the potential drivers' list L_{x_i}

```
Require: Demand $u$, potential drivers at $x_i$ i.e. $L_{x_i}$.
Ensure: $k^\star$ : The best driver having $x_i$ as pick-up stop and $x_j$ as drop-off stop
        $\sigma^\star$ : Time-savings generated by sub-path $(x_i, x_j)$.
 1: Initialization : $\sigma^\star \leftarrow 0$, $k^\star \leftarrow -1$
 2: for all $k$ in $L_{x_i}$ do
 3:   for each $x_j \in P$, such that $x_j$ is situated after $x_i$ do
 4:     if $k \in B_d(x_j)$ and $(x_i, x_j)$ form an admissible ride-sharing path for the driver
        $k$ and the rider $u$ then
 5:       if $\sigma^\star \geq t_a^u(x_j, p) - \big(\max\{t_a^u(x_i, p) + \tau(x_i)_{pc}, t_a^k(x_i, c)\} + \delta(x_i, x_j)\big)$ then
 6:         $\sigma^\star \leftarrow t_a^u(x_j, p) - \big(\max\{t_a^u(x_i, p) + \tau(x_i)_{pc}, t_a^k(x_i, c)\} + \delta(x_i, x_j)\big)$
 7:         Update the best driver $k^\star$
 8:       end if
 9:     end if
10:   end for
11: end for
```

Algorithm 4 Best driver in all drivers buckets

```
Require: $L_{x_i}$, Demand $u$.
Ensure: Best driver $k^\star$, Best pick-up stop $x_{i^\star}$, Best drop-off stop $x_{j^\star}$.
 1: Initialization: $k^\star \leftarrow -1$, $x_{i^\star} \leftarrow -1$, $x_{j^\star} \leftarrow -1$
 2: for all $x_i$ in $P$ do
 3:   Find the driver $k$ with greater time-savings having $x_i$ as pick-up stop, using
      Algorithm 3
 4:   Update the best driver $k^\star$,Update the best pick-up stop $x_{i^\star}$ and drop-off stop $x_{j^\star}$
 5: end for
```

Step 4 This last step is an updating process that occurs each time a ride-sharing $(x_{i^\star}, x_{j^\star})$ with positive time-savings has been found. The rider's route has to be updated to take into account his new arrival time $t_1 = \max\{t_a^{k^\star}(x_{i^\star}, c), t_a^u(x_{i^\star}, p) + \tau(x_{i^\star})_{pc}\} + \delta(x_{i^\star}, x_{j^\star})$ at $x_{j^\star}$. The update also occurs on the driver's side: its route becomes $O_{k^\star} \rightsquigarrow x_{i^\star} \rightsquigarrow x_{j^\star} \rightsquigarrow D_{k^\star}$, where $O_{k^\star}$ is the current location of the driver. Then *Step 1* is run again to take into account the stop $x_{j^\star}$ as the new starting location at time t_1, as well as updating the rider's bucket.

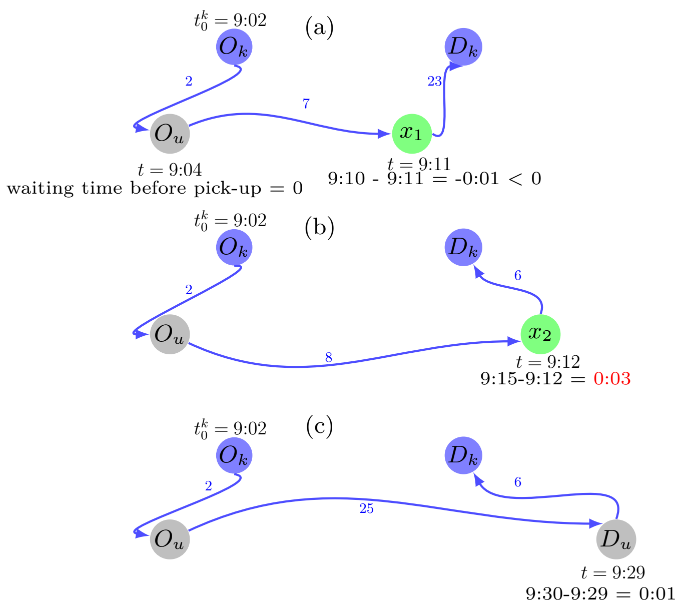

Figure 5. Substitution sub-path for potential driver k contained in the list L_{O_u}. Driver $k \in L_{O_u}$ is contained in $\{L_{x_1}, L_{x_2}, L_{D_u}\}$. Ride-sharing (O_u, x_1) increases the earliest arrival time at x_1; therefore it is cancelled. Between ride-sharing (O_u, D_u) and (O_u, x_2), the last one better improves the arrival time at x_2; it is therefore the best substitution sub-path.

4 Numerical results

Computational experiments have been conducted to evaluate the effectiveness of the proposed approach in terms of arrival time. Our practical application is tested on real data and real road network. Our methodology switches between car and public-transportation networks. Stop projections from public transportation to road network and computing shortest paths using car is done using OpenStreetMap[2]. Public transportation timetabling in Lorraine region is provided by the *navitia* API. The proposed methods were implemented in C# Visual Studio 2010. The experiments were done on an Intel Xeon E5620 2.4 Ghz processor, with 8 GB RAM memory.

Offers-demands data Real data is provided by Covivo company[3]. These data concern employees of Lorraine region travelling between their homes and their work places. The real data instance is composed of 1513 participants. Among them, 756 are willing to participate in ridesharing service only as drivers. The rest of participants (i.e. 757) are willing to use other modes in order to reach their destinations. Thus, the data instance is composed of 756 offers and 757

[2] http://www.openstreetmap.org
[3] http://www.covivo.fr

demands. The smallest and the greatest trip's distances are 2 km and 130 km, respectively. The set of offers and demands are filtered such that we can never find a driver's offer and a rider's demand which have either the same starting or ending locations. This way will also allow to test the flexibility of our approach compared to the traditional ride-sharing. The early departure time and the latest departure time for each trip are fixed at 7:00 a.m. and at 8:00 a.m, respectively. The detour time of the driver is fixed to at most 20% of his initial trip duration.

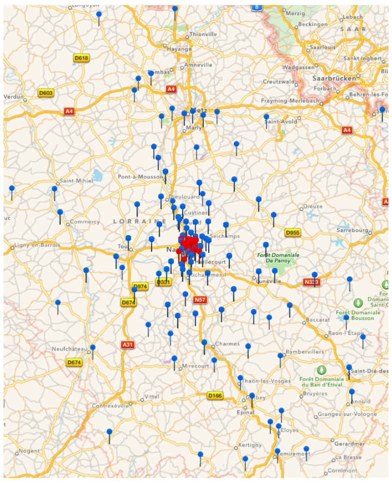

Figure 6. Map visualization of ride-sharing offers. Red and blue points are the geocoded home and work locations, respectively.

Computational results The company which provided the data is still in test process and asked for an embargo of 18 months before the publication of numerical results. Some insight about the efficiency of our approach can nevertheless be given, but the lack of available benchmarks does not allow a direct comparison with previous research. First, this shows that a true real-time application is possible since the whole process running on a personal computer requires only a few seconds. Second, using the alternative hypothesis that the difference between the arrival time at destination using only public transportation and the arrival time at destination partially using ride-sharing, we can show that there is a statistically significant gain for the rider.

5 Conclusion

The problem described in this paper is a first step in designing new dynamic multi-modal transportation process. Multi modalities usually includes pedestrian, cycling, private car or public bus or train transportations. Our approach allows to also include ride-sharing without need changing the process of public-transport. As far as we know, it is the first attempt to solve a mix public-transport and ride-sharing problem in real time under matching constraints.

We have used a simple objective function, which, although appropriate for certain types of users, might be redefined in several terms: the total monetary cost for the rider, the deviation from the origin-destination trip for the driver, the number of transshipment stops, etc. Our approach can integrate those features as a weighted sum in the objective function.

Our approach is sensitive to the number of information to be stored. We have contained the necessary amount of needed memory by means of three features: a restriction of pick-up and drop-off location along a path, a limited detour constraint, and a time-window for pick-up and drop-off. Nevertheless, our practical implementation and tests on real data instances show the viability of our approach for a real-time application.

Acknowledgments. This research was partially funded thanks to the swiss CTI grant 15229-1 PFES-ES received by Sacha Varone.

References

1. Ambrosino, D., Sciomachen, A.: An algorithmic framework for computing shortest routes in urban multimodal networks with different criteria. Procedia - Social and Behavioral Sciences 108(0), 139 – 152 (2014), operational Research for Development, Sustainability and Local Economies
2. Bast, H., Delling, D., Goldberg, A., Müller-Hannemann, M., Pajor, T., Sanders, P., Wagner, D., Werneck, R.: Route planning in transportation networks. MSR-TR-2014-4 8, Microsoft Research (1 2014)
3. Bit-Monnot, A., Artigues, C., Huguet, M.J., Killijian, M.O.: Carpooling : the 2 synchronization points shortest paths problem. In: 13th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS). vol. 13328, p. 12. Sophia Antipolis, France (Sep 2013)
4. Delling, D., Dibbelt, J., Pajor, T., Wagner, D., Werneck, R.: Computing multimodal journeys in practice. In: Bonifaci, V., Demetrescu, C., Marchetti-Spaccamela, A. (eds.) Experimental Algorithms, Lecture Notes in Computer Science, vol. 7933, pp. 260–271. Springer Berlin Heidelberg (2013)
5. Liu, L., Yang, J., Mu, H., Li, X., Wu, F.: Exact algorithms for multi-criteria multi-modal shortest path with transfer delaying and arriving time-window in urban transit network. Applied Mathematical Modelling 38(9-10), 2613–2629 (2014)
6. Müller-Hannemann, M., Schulz, F., Wagner, D., Zaroliagis, C.: Timetable information: Models and algorithms. In: Geraets, F., Kroon, L., Schoebel, A., Wagner, D., Zaroliagis, C. (eds.) Algorithmic Methods for Railway Optimization, Lecture Notes in Computer Science, vol. 4359, pp. 67–90. Springer Berlin Heidelberg (2007)
7. Pyrga, E., Schulz, F., Wagner, D., Zaroliagis, C.: Efficient models for timetable information in public transportation systems. J. Exp. Algorithmics 12, 2.4:1–2.4:39 (Jun 2008)