Carpooling: the 2 Synchronization Points Shortest Paths Problem
Arthur Bit-Monnot, Christian Artigues, Marie-José Huguet, Marc-Olivier Killijian

To cite this version:
Arthur Bit-Monnot, Christian Artigues, Marie-José Huguet, Marc-Olivier Killijian. Carpooling: the 2 Synchronization Points Shortest Paths Problem. 13th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS), Sep 2013, Sophia Antipolis, France. 12 p, 10.4230/OASIcs.ATMOS.2013.150. hal-00843628v2

HAL Id: hal-00843628
https://hal.science/hal-00843628v2
Submitted on 1 Sep 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Carpooling: the 2 Synchronization Points Shortest Paths Problem *

Arthur Bit-Monnot1,2, Christian Artigues1,2, Marie-José Huguet1,3, and Marc-Olivier Killijian1,2

1 CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France
2 Université de Toulouse, LAAS, F-31400 Toulouse, France
3 Université de Toulouse, INSA, F-31400 Toulouse, France
{bit-monnot, artigues, huguet, killijian}@laas.fr

Abstract
Carpooling is an appropriate solution to address traffic congestion and to reduce the ecological footprint of the car use. In this paper, we address an essential problem for providing dynamic carpooling: how to compute the shortest driver’s and passenger’s paths. Indeed, those two paths are synchronized in the sense that they have a common subpath between two points: the location where the passenger is picked up and the one where he is dropped off the car. The passenger path may include time-dependent public transportation parts before or after the common subpath. This defines the 2 Synchronization Points Shortest Path Problem (2SPSPP). We show that the 2SPSPP has a polynomial worst-case complexity. However, despite this polynomial complexity, one needs efficient algorithms to solve it in realistic transportation networks. We focus on efficient computation of optimal itineraries for solving the 2SPSPP, i.e. determining the (optimal) pick-up and drop-off points and the two synchronized paths that minimize the total traveling time. We also define restriction areas for reasonable pick-up and drop-off points and use them to guide the algorithms using heuristics based on landmarks. Experiments are conducted on real transportation networks. The results show the efficiency of the proposed algorithms and the interest of restriction areas for pick-up or drop-off points in terms of CPU time, in addition to its application interest.

1998 ACM Subject Classification "G.2.1 Combinatorics", "G.2.2 Graph Theory", "F.2.2 Nonnumerical Algorithms and Problems"

Keywords and phrases Dynamic Carpooling - Shortest Path Problem - Synchronized Paths

Digital Object Identifier 10.4230/OASIcs.xxx.yyy.p

1 Introduction
Due to the demographic evolution and the urban spread off during the last decades, people have moved away from urban centers and now live in residential areas. In order to decrease the urban traffic congestion and its societal issues, transport strategies have encouraged to park private cars near multimodal hubs (i.e. park and ride stations) and to use the public transport system to reach downtown destinations. However, congestion problems have moved from urban to sub-urban areas where people commute with their cars either to reach the employment areas or to connect to the public transport system. An appropriate solution, requiring little investment and reducing the ecological footprint of the car use, is

* This work was partially supported by LAAS, CNRS and ANR French national program for Security and Informatics (grant #ANR-11-INSE-010).
the promotion of shared transport, like carpooling, which enables private cars to become part of the public transport system. The main restraints of carpooling development are insecurity, payment transaction of the shared journey, low number of matches and lack of flexibility, as well as constraint feelings. For instance, regular (i.e., static) carpooling forces the driver to directly go home after work or to plan his trip in advance. Dynamic carpooling relaxes some of these constraints (few matches, lack of flexibility and constraint feelings). Dynamic carpooling should enable automatic (or semi-automatic) destination guessing and trip proposals for drivers. Regarding users, it should help real-time matching with drivers. In this paper, we address the issue of computing journeys for a driver and a passenger to carpool together in a complete trip. The two synchronized paths can be decomposed into 5 subpaths. The trip is composed of two convergent paths towards a first synchronization point, i.e. the meeting point, a shared path towards the second synchronization point, i.e., the drop-off point and two divergent paths from this drop-off point towards each destination, henceforth the name 2 Synchronization Points Shortest Path Problem (2SPSPP).

In the problem definition, we can distinguish two types of users. The driver drives his car and is willing to take a detour in order to pick up a passenger and drive him for some part of the trip. The passenger can walk or use public transportation to join a pick-up point in order to be driven. For example, as in the AMORES project[2], we can consider that the users use smartphones to communicate carpooling requests and offers, to find matches between those, and possibly to compute their optimal itineraries. In this paper, we focus explicitly on the computation of optimal itineraries for the 2SPSPP, i.e. the (optimal) pick-up and drop-off points and the 5 paths which compose the full trip as in figure 1. We consider the objective of minimizing the total travel time for both users.

2 Problem Statement

A multimodal transportation network is modeled with an edge-labeled graph \( G = (V, E, \Sigma) \) where \( V \) is the set of nodes, \( \Sigma \) the set of modes (for instance foot, car or public transportation) and \( E \) is the set of labeled edges. A labeled edge \((i, j, m)\) is a route from a node \( i \) to a node \( j \) having the mode \( m \). Moreover, a cost function \( c_{ijm} \) is associated to each edge \((i, j, m)\) representing the travel time. These costs may be static or time-dependent, in this case \( c_{ijm}(\tau) \) gives the travel time from \( i \) to \( j \) in mode \( m \) when leaving \( i \) at time \( \tau \). A path \( P_{ij} \) is an ordered list of nodes from \( i \) to \( j \). Its cost, denoted by \( len(P_{ij}, \tau) \), is the sum of the cost of each edge when leaving node \( i \) at time \( \tau \).

\[\textbf{Definition 1 (2SPSPP).}\] Consider an edge-labeled graph \( G = (V, E, \Sigma) \), a car driver \( c \) and a pedestrian \( p \) with their own origins and destinations, denoted by \( o_c, d_c \) and \( o_p, d_p \), and with their departure times \( \tau_c \) and \( \tau_p \) respectively. One aims to determine a pick-up point \( x_{up} \) and a drop-off point \( x_{off} \), and five paths \( P_{o_p x_{up}}, P_{o_c x_{up}}, P_{x_{up} x_{off}}, P_{x_{off} d_p}, P_{x_{off} d_c} \) such that a carpooling cost is minimized.

This problem is depicted in figure 1; in this figure edges’ labels represent the allowed modes in each part of the network: \{c\} (ie. car) for the driver and \{f, pt\} (ie. foot or public transportation) for the pedestrian.

A solution \( S \) of the carpooling problem is a pair of pick-up and drop-off points \( (x_{up}, x_{off}) \) and five paths. The considered cost of a carpooling itinerary is the sum of travel times for the two users from their origin to their destination, i.e. difference between arrival and departure time for both users. Let us define \( \tau_{x_{up}}^{(u)} \) the arrival time of user \( u \) at point \( x \), for instance \( \tau_{d_p}^{(p)} \) is the arrival time of the passenger at \( d_p \). For the considered overall carpooling cost, we
The first line corresponds to the cost of the forward search and one from a lower bound of the exact cost. In bidirectional search, two algorithms run: one from the current node and the destination the Dijkstra algorithm is guided towards the destination using an estimate cost between the origin is reached. Speed-up techniques were introduced to improve the efficiency of this algorithm. In this algorithm, a label with minimal cost is extracted from the current cost from or inserted in the path from the current node. Given a weighted graph \( G = (V, E) \), an origin node \( o \) and a destination node \( d \), the Shortest Path Problem from \( o \) to \( d \) (SPP) is solved in polynomial time with the well-known Dijkstra algorithm. In this algorithm, a label \( l_x = (\pi_x, p_x) \) is associated to a node \( x \), where \( \pi_x \) is the current cost from \( o \) to \( x \) and \( p_x \) the reference of the predecessor node for the current best path from \( o \) to \( x \). A queue \( Q \) is used for exploring the labels in an increasing order of their costs: the label with minimal cost is extracted from \( Q \), settled and its successors are updated or inserted in \( Q \). The algorithm stops when node \( d \) is settled, \( \pi_d \) then gives the cost of the shortest path from \( o \) to \( d \) and the path is obtained by exploring the predecessor \( p_d \) until the origin is reached. Speed-up techniques were introduced to improve the efficiency of this algorithm for solving the one-to-one shortest path problem. In the \( A^* \) goal directed search, the Dijkstra algorithm is guided towards the destination using an estimate cost between the current node and the destination \( d \). The optimal solution is obtained if the estimation is a lower bound of the exact cost. In bidirectional search, two algorithms run: one from \( o \) to \( d \) (forward search) and one from \( d \) to \( o \) on the reverse graph (backward search). When a

\[ \text{Definition 2 (Carpooling Cost).} \quad \text{Given a solution } S \text{ of the 2SPSPP, we aim at minimizing} \]
\[ \text{cost}(S) = \left( \tau_{dc}^{(c)} - \tau_{oc}^{(c)} \right) + \left( \tau_{dp}^{(p)} - \tau_{op}^{(p)} \right), \]
\[ \text{the total time spent traveling by both users.} \]
\[ \text{The first line corresponds to the cost of the 2 paths } P_{opxup} \text{ and } P_{octxup} \text{ plus the waiting time,} \]
\[ \text{the second line is the cost of the path } P_{xupxoff} \text{ counted twice since it is made by both users} \]
\[ \text{and the third one is the cost of the 2 paths } P_{xoffd_p} \text{ and } P_{xoffd_c}. \]

We remark that we are dealing with a polynomial problem as, for fixed synchronization points, 5 calls to the Dijkstra algorithm (two of them with the time-dependent variant) are sufficient to obtain the optimal solution. As there are \( O(|V|^2) \) possible synchronization points, the complexity result follows. However, a naive method of enumerating all possible pairs of synchronization points is not applicable on transportation networks having realistic size. The aim of this study is to propose an efficient algorithm for solving the 2SPSPP.

### 3 Related Work

Given a weighted graph \( G = (V, E) \), an origin node \( o \) and a destination node \( d \), the Shortest Path Problem from \( o \) to \( d \) (SPP) is solved in polynomial time with the well-known Dijkstra algorithm. In this algorithm, a label \( l_x = (\pi_x, p_x) \) is associated to a node \( x \), where \( \pi_x \) is the current cost from \( o \) to \( x \), and \( p_x \) the reference of the predecessor node for the current best path from \( o \) to \( x \). A queue \( Q \) is used for exploring the labels in an increasing order of their costs: the label with minimal cost is extracted from \( Q \), settled and its successors are updated or inserted in \( Q \). The algorithm stops when node \( d \) is settled, \( \pi_d \) then gives the cost of the shortest path from \( o \) to \( d \) and the path is obtained by exploring the predecessor \( p_d \) until the origin is reached. Speed-up techniques were introduced to improve the efficiency of this algorithm for solving the one-to-one shortest path problem. In the \( A^* \) goal directed search, the Dijkstra algorithm is guided towards the destination using an estimate cost between the current node and the destination \( d \). The optimal solution is obtained if the estimation is a lower bound of the exact cost. In bidirectional search, two algorithms run: one from \( o \) to \( d \) (forward search) and one from \( d \) to \( o \) on the reverse graph (backward search). When a
connection is found between the forward and the backward algorithms a feasible solution is obtained. However, this solution may not be optimal and the two algorithms run until there is no better solution connecting the forward and the backward labels.

In addition, different preprocessing techniques were proposed. The objective is to compute and store informations on the graph to speed-up the shortest path queries. An overview of various efficient preprocessing techniques such as landmarks, contraction hierarchy or flags is given in [4]. We only present one of them, the ALT algorithm [5] that we use later. ALT is based on landmarks and consists in computing the shortest paths from all the nodes to a (small) subset of landmarks. These precomputed shortest paths are then combined with the $A^*$ search and triangular inequality to provide strong lower bounds on the shortest paths.

Some extensions of the SPP were proposed to deal with time-dependency of travel times. When the cost function on arcs satisfies the FIFO property, the time-dependent SPP remains polynomially solvable [7] and a straightforward adaptation of the Dijkstra algorithm can be done. The FIFO property guarantees that, along any edge, it is never possible to depart later and arrive earlier. In the time-dependent Dijkstra algorithm, when the destination is reached, one has both the minimal cost of the shortest path and the minimal arrival time at the destination. However, many efficient techniques based on bidirectional search cannot be easily extended in the time-dependent case as the start time is given at only one node (at the origin or at the destination). For instance, an adaptation of the bidirectional ALT was proposed in [9] by considering a lower bound of travel time in the backward search. Each connection then needs to be re-evaluated to obtain the exact cost from the connection point to the destination, increasing the complexity of the problem.

When taking multimodality into account, one has to model the transportation network and the constraints on transportation modes (for instance a passenger may wish to avoid a given sequence of modes). In [3], the authors use an edge-labeled graph where a mode $m$ is associated to each edge. They propose to use a regular language $L$ to model constraints on modes and define the regular language constrained shortest path problem (RegLCSP). Their algorithm, called $D_{RegLC}$, is an extension of the Dijkstra algorithm constrained by the regular language. Product-nodes are simply a pair $(x, s)$ where $x$ is a node and $s$ a state in the automaton. The algorithm should be stopped as soon as a product-node $(d, s_f)$ is settled, where $d$ is the destination and $s_f$ is an accepting state in the automaton.

We are not aware of research addressing problems similar to the 2SPSPP. In carpooling papers, the authors usually consider variants of vehicle routing problems for solving static or long term carpooling problems (to collect several people at their home for instance and drive them at work each week or each day). Dynamic carpooling problems were also considered and several authors (see for instance [1, 10]) proposed a multi-agent architecture in which some heuristics are used to solve the matching problem between drivers and requesters. But, to the best of our knowledge, the driver is not derouted for collecting a user.

In [6], the authors propose a method for synchronizing two itineraries in a point such that the global cost of the two paths is minimized. The problem under study is the 2-Way Multi Modal Shortest Path problem in which two itineraries are defined for the same user at different times of the day between a given origin and destination. The proposed method is based on 4 multi-directional algorithms (forward and backward search) to obtain the optimal parking node such as the sum of an outgoing path and a return path is minimized. As already mentioned, the main difficulty arises when facing time-dependency, an expensive re-evaluation process is added to the 4 algorithms to obtain the exact cost of paths.
4 The proposed approach for the 2SPSPP

4.1 General principle

In the problem under study, we consider that travel times for the car and foot modes are time-independent, unlike travel times for the public transportation mode that are time-dependent. Moreover, departure times are given at the origins. The proposed method aims to overcome the difficulty due to time-dependency, i.e. the use of lower bounds in algorithms where start times are unknown and the need for re-evaluation. Indeed, as public transportation is time-dependent, the use of backward search from the pedestrian’s destination or from the potential drop-off points requires some (time consuming) re-evaluation. Therefore, in our method, forward search (from the origin to the destination) is used as long as it is possible to obtain the exact value of travel time and not a lower bound.

We propose a method combining 4 forward algorithms and 1 backward algorithm without any need of re-evaluation. In Figure 2, the arrows on the arcs indicates the direction of the algorithms. First, we launch 2 forward algorithms \(A_1\) and \(A_2\) from the origins and 1 backward algorithm \(A_4\) from the driver’s destination. Each node reached by the 2 forward algorithms \(A_1\) and \(A_2\) is a potential pick-up point. A forward algorithm \(A_3\) is then launched from the set of potential pick-up points towards potential drop-off points. The aim of \(A_3\) is to find the best origin between a set of potential origin nodes (here the pick-up points) and a set of destination node (here the drop-off points). Then, each time a node is reached by algorithm \(A_3\) and the backward algorithm \(A_4\), a potential drop-off point is determined. Finally, another forward algorithm \(A_5\) is launched from the set of drop-off points towards the pedestrian’s destination. The aim is to determine the best origin between a set of potential origin nodes (drop-off points) and a single destination node (pedestrian’s destination).

Algorithms \(A_1\), \(A_2\) and \(A_4\) are standard \(D_{RegLC}\) for solving the one-to-all SPP in a multimodal and time-dependent network. The multimodal constraints only state that car must be used in \(A_2\), \(A_3\) and \(A_4\) and that either foot or public transportation can be used in \(A_1\) and \(A_5\). Algorithms \(A_3\) and \(A_5\) are dedicated to solving the best origin problem. We present in the next section how this problem can be solved.

4.2 The Best-Origin Problem

Given a set \(S\) of several origin nodes with individual costs and arrival times (ie. \(\pi_x\) and \(\tau_x\) \(\forall x \in S\)) and a set of target nodes \(D\), we aim at selecting the best origin to minimize the cost at the destinations.
Definition 3 (Best Origin Problem (BOP)). Given a weighted directed graph \( G = (V, E) \), a set of origins \( S \) and a set of destination nodes \( D \), the expected output is, for every \( d \in D \), an origin \( x \) having label \((\pi_x, \tau_x)\) such that, for any other origin \( y \in S \) with label \((\pi_y, \tau_y)\), it holds that: \( \pi_x + \text{len}(P_{xd}, \tau_x) \leq \pi_y + \text{len}(P_{yd}, \tau_y) \).

Solving BOP in time-independent networks has been done implicitly for decades using the forward Dijkstra algorithm from the origins. Each time a node is touched by the algorithm, it is updated with the best available cost. The only predecessor kept is the one providing the best cost. This problem can therefore be solved by inserting all potential origins with their initial costs into the priority queue and let the Dijkstra algorithm run until every \( d \in D \) is settled. The last predecessor of \( d \) in the optimal path would be the best origin.

In the time-dependent context, when there is consistency between cost and arrival time (see Definition 4), we can consider that the label with the best cost is the one with the best arrival time. Using the FIFO-property, it is easy to see that the only label we are interested in is the one with the lowest cost. The Dijkstra approach (dropping all labels with greater cost) can therefore be applied to solve this problem.

Definition 4 (Consistency between cost and arrival time). Given a shortest path solver using cost and arrival time labels, we say that cost and arrival times are consistent if and only if, for any two labels \((\pi_x, \tau_x)\) and \((\pi_y, \tau_y)\), \(\pi_x \leq \pi_y \iff \tau_x \leq \tau_y\).

However, the classical solution approach does not hold when costs and arrival times are not consistent. Figure 3 gives an example of the BOP with inconsistent costs and arrival times. Let \( S = \{o_1, o_2\} \) be a set of origins having respective inconsistent labels \((8:05, 2)\) and \((8:06, 0)\), and a destination \( d \). Travel times are time-dependent and are detailed in Figure 3. Two labels are obtained for \( v \): \((8:06, 3)\) due to \( o_1 \) and \((8:07, 1)\) due to \( o_2 \). The best origin for \( d \) is \( o_1 \) (with a cost of 12), but the best label for \( v \) is the one from \( o_2 \). Applying the Dijkstra algorithm on this instance would discard the \((8:06, 3)\) label in \( v \) and \( o_2 \) would be selected as the best origin, giving a suboptimal result.

To solve this problem, we propose an algorithm performing forward search only and not doing any reevaluation. This algorithm is inspired by Martins’ algorithm [8] to keep track of costs and arrival times. However, we note that this algorithm stays mono-objective since we are only interested in finding the best cost in \( d \). Labels are sorted by cost only and priority queues for the Dijkstra algorithm can be used. Moreover, the extension of this algorithm

![Figure 3](image-url)
to a multimodal network is straightforward using the product network of the graph and the automaton representing constraints on modes.

To prune labels during the search, we introduce the following dominance rule that allows discarding labels that can not be part of an optimal solution.

**Definition 5 (Exact Dominance rule (1)).** Given a node \( x \) and two labels \( l = (\pi_x, \tau_x) \) and \( l' = (\pi'_x, \tau'_x) \), we say that \( l \) dominates \( l' \) if and only if \( \tau_x \leq \tau'_x \) and \( \pi_x - \pi'_x \leq \tau_x - \tau'_x \).

**Proposition 6.** At least one optimal solution is reachable if dominance rule 1 is applied.

**Proof.** Let us select an optimal path \( P \) from an origin \( o \) to \( d_p \) that verifies

1. \( i \) is the first node on the path such that the label \((\pi', \tau')\), extended to obtain \( P \), is dominated according to rule 1 by another label \((\pi, \tau)\) of node \( i \).
2. Among the optimal paths, \( P \) is the one with the smallest number of arcs between \( i \) and \( d_p \).

According to rule 1 we have \( \pi \leq \pi' - (\tau' - \tau) \) and \( \tau \leq \tau' \). Let \( P_{id_p} \) the subpath of \( P \) from \( i \) to destination \( d_p \). It comes: \( \pi + \text{len}(P_{id_p}, \tau) \leq \pi' + \text{len}(P_{id_p}, \tau') - (\tau' - \tau) \). Since \( \text{len}(P_{id_p}, \tau) \leq \text{len}(P_{id_p}, \tau') + (\tau' - \tau) \) in FIFO networks, we finally have \( \pi + \text{len}(P_{id_p}, \tau) \leq \pi' + \text{len}(P_{id_p}, \tau') \).

It follows that extending label \((\pi, \tau)\) from node \( i \) yields an optimal path.

As it will be shown in Section 6, solving the BOP with this dominance rule is not always efficient. We, then, introduce a second dominance rule which is heuristic. The pros and cons of those two dominance rules will be discussed further.

**Definition 7 (Heuristic Dominance rule (2)).** Given a node \( x \) and two labels \( l = (\pi_x, \tau_x) \) and \( l' = (\pi'_x, \tau'_x) \), we say that \( l \) dominates \( l' \) if and only if \( \tau_x \leq \tau'_x \) and \( \pi_x \leq \pi'_x \).

This second dominance rule is heuristic as it may prune labels leading to optimal solutions. An illustration of this situation is given in figure 4 where nodes \( x_1 \) and \( x_2 \) are potential drop-off points, nodes \( y \) and \( z \) are explored by the pedestrian to reach his destination. Labels of \( x_1 \) and \( x_2 \) are extended to \( y \). When considering the second dominance rule, at node \( y \), the blue label \((9:12; 43)\) is dominated by the green one \((9:05; 40)\). However, the extension of the blue label to node \( z \) gives a better solution \((9:17; 48)\) due to the time-dependent arc from \( y \) to \( z \). This models a situation where the passenger would have to wait for the same bus whether it was dropped at \( x_1 \) or at \( x_2 \). As a consequence, the second dominance rule may filter labels that could lead to optimal solutions in terms of cost since it doesn’t account for those situations.

**Figure 4** An instance where dominance rule 2 would yield suboptimal results.

In our mono-objective variant of MARTINS algorithm, at first, all potential origins are inserted in a queue \( Q \) with their original costs and arrival times. At each iteration, the
undominated label with lowest cost in $Q$ is selected, settled and its edges are relaxed. The generated labels are inserted in $Q$. Either dominance rule can be used to prune dominated labels.

**Proposition 8.** At each iteration, a label settled has a cost greater than or equal to the cost of any label previously settled.

**Proof.** Given that edge weights are non-negative, the cost of a label will be greater than or equal to its predecessor’s. Since the priority queue selects labels with lowest cost first, all labels inserted in queue will have a cost greater or equal than the one currently selected. ▶

**Corollary 9.** In the Mono-Objective Martins algorithm, the lowest cost of a node is the one of its first settled label.

Using Corollary 9, we can stop the algorithm as soon as a label is settled for all $d \in D$. By looking at predecessors, we deduce the best origins $o$ and the paths $P_{od}$.

**Proposition 10.** There can be at most $|S|$ undominated labels per node, $|S|$ being the number of potential origins.

**Proof.** Given a potential origin $o$ with label $(\pi_o, \tau_o)$ and a node $v$, we suppose there are two paths $P$ and $P'$ from $o$ to $v$. The label generated in $v$ by following those paths would be $l_v = (\pi_o + \text{len}(P, \tau_o), \tau_o + \text{len}(P, \tau_o))$ and $l'_v = (\pi_o + \text{len}(P', \tau_o), \tau_o + \text{len}(P', \tau_o))$. Note that the second inequality of dominance rule 1 is always verified since $(\pi_o + \text{len}(P, \tau_o)) - (\pi_o + \text{len}(P', \tau_o)) = (\tau_o + \text{len}(P, \tau_o)) - (\tau_o + \text{len}(P', \tau_o))$. Thus, if $\text{len}(P, \tau_o) \leq \text{len}(P', \tau_o)$, $l_v$ dominates $l'_v$. Otherwise $l_v$ is dominated by $l'_v$. Therefore, each potential origin generates at most one undominated label per node. ▶

**Complexity.** Using Proposition 10, we deduce that there can be at most $|E| \cdot |S|$ labels inserted in $Q$. When extracted from the queue, these labels need to be checked for dominance, which can be done in $|S|$. Hence, the worst-case complexity of this algorithm is $O(|E| \cdot |S| \cdot r_Q + |E| \cdot |S| \cdot e_Q + |E| \cdot |S|^2)$ where $r_Q$ is the cost of reordering the queue after inserting one label and $e_Q$ is the complexity of extracting the next label.

We note that this worst-case complexity is greater than the one of running $|S|$ DIJKSTRA algorithms. However, in our experiments, these two rules allow to discard many labels.

## 5 Algorithm for the 2SPSPP

### 5.1 A sequential approach

In our method, we split the carpooling problem into three One-to-All Shortest Path Problems and two Best Origin Problems. The two BOP are using the nodes settled by the shortest path algorithms as their potential origins. We call $A_i$ the algorithm used to solve the $i^{th}$ problem and $N_i$ the set of nodes it settles. A specification of the algorithms and the problems they have to solve is given in Table 1. All five algorithms are to be executed sequentially. The 2SPSPP is solved when $d_p$ is settled by algorithm $A_5$: we are able to retrieve $x_{off}$ (best origin of $d_p$ in $A_5$) and $x_{ap}$ (best origin of $x_{off}$ in $A_3$).

We saw in section 4.2, that the consistency between costs and arrival times has an impact on which algorithm can be used to solve the BOP. We will therefore study this consistency for each part of the proposed method. We call $\pi_x^{(i)}$ the cost of node $x$ in algorithm $A_i$ and $\tau_x^{(i)}$ the arrival time at $x$ for the algorithm $A_i$. We are also given $\tau_{ap}^{(p)}$ and $\tau_{ap}^{(c)}$, respectively the departure times of the passenger and the driver. Note that in $A_1$, $A_2$ costs and arrival
times are consistent and then in $A_4$ we do not consider the time since it is executed on a time-independent graph and the arrival time has no impact on the rest of the method. Then, for $A_3$ and $A_5$, initial costs and arrival times of nodes in $X_{up}$ and $X_{off}$ derive from Definition 2 and are defined as follow:

\[
\begin{align*}
\text{in } A_3, \text{ for } x \in X_{up} : \tau_x^{(3)} &= \max(\tau_x^{(1)}, \tau_x^{(2)}); \quad \text{and } \pi_x^{(3)} = \pi_x^{(2)} + \tau_x^{(1)} + |\tau_x^{(1)} - \tau_x^{(2)}| \\
\text{in } A_5, \text{ for } x \in X_{off} : \tau_x^{(5)} &= \tau_x^{(3)}; \quad \text{and } \pi_x^{(5)} = \pi_x^{(3)} + \pi_x^{(4)} (2)
\end{align*}
\]

Given this definition and recalling that cost is counted twice in $A_3$, it is fairly easy to show that, for any node $x \in N_3$, \(\pi_x^{(3)} = 2 \times \tau_x^{(3)} - \tau_x^{(p)} - \tau_x^{(c)}\). Hence, costs and arrival times are consistent in $N_3$. However, breaking down the cost of a node $x \in X_{off}$ leads us to \(\pi_x^{(5)} = 2 \times \tau_x^{(3)} - \tau_x^{(p)} - \tau_x^{(c)} + \pi_x^{(4)}\), showing that costs and arrival times are not consistent in $N_5$ (since $X_{off}$ is a subset of $N_5$).

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Source</th>
<th>Target</th>
<th>Settled Nodes</th>
<th>Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_1$</td>
<td>$o_p$</td>
<td>All</td>
<td>$N_1$</td>
<td>Shortest Path (forward)</td>
</tr>
<tr>
<td>$A_2$</td>
<td>$o_c$</td>
<td>All</td>
<td>$N_2$</td>
<td>Shortest Path (forward)</td>
</tr>
<tr>
<td>$A_3$</td>
<td>$X_{up} = N_1 \cap N_2$</td>
<td>All</td>
<td>$N_3$</td>
<td>Best Origin</td>
</tr>
<tr>
<td>$A_4$</td>
<td>$d_c$</td>
<td>All</td>
<td>$N_4$</td>
<td>Shortest Path (backward)</td>
</tr>
<tr>
<td>$A_5$</td>
<td>$X_{off} = N_3 \cap N_4$</td>
<td>$d_p$</td>
<td>$N_5$</td>
<td>Best Origin</td>
</tr>
</tbody>
</table>

**Table 1**: Specification of the algorithms used to solve the 2SPSPP for carpooling.

According to those results, a Dijkstra like algorithm can be used for solving BOP in $A_3$. However, because of the inconsistency between costs and arrival times in $A_5$, Mono-Objective Martins has to be used to make sure no solution is discarded.

The complexity of this approach falls back on the one of four Dijkstra algorithms and one Mono-Objective Martins. Since any node of the graph can be a potential drop-off point, the worst-case complexity of our method is $O(|E| \cdot |V|^2)$ when using a binary heap.

### 5.2 Integrated Approach

The sequential approach raises the problem of exploring four times the whole graph. In this section, we present a method integrating the five algorithms to speed up the search. The idea is to have all five algorithms initialized and select the one with the lowest cost in its heap for execution as illustrated in the algorithm given in Listing 1. When a pick-up (resp. drop-off) point is discovered, it is dynamically inserted into $A_3$ (resp. $A_5$)’s heap.

**Listing 1**: Integrated approach: the algorithm with lowest cost is selected for execution.

```c
// Init : insert $o_p$ in $A_1$’s heap, $o_c$ is $A_2$’s heap and $d_c$ in $A_4$’s heap while not all heaps empty
   $k =$ number of algorithm with smallest cost in heap
// run one iteration in selected algorithm
// and retrieve the settled node
   $x =$ Algo[$k$].make_one_iteration() // $x$ is settled in $A_k$
   if $x = d_p$ and $k=5$ then stop // Problem solved
   if $k = 1$ or $k=2$ then check pick-up point
   if $k = 3$ or $k=4$ then check drop-off point
stop // no solution found
```
Initialization is done by inserting the origin of the passenger, the origin of the driver and the destination of the driver into, respectively, $A_1$, $A_2$ and $A_4$’s heaps with a zero cost and departure times from the origins.

An iteration of our method starts by selecting $k$ such that the next label to be settled in $A_k$ has the lowest cost among all algorithms’ heaps. Then, $A_k$ makes one iteration (setting the next label and relaxing its edges) and yields the node $x$ it just settled. If $d_p$ was settled by $A_5$, the problem is solved. Otherwise, we check if $x$ can be used as a pick-up or drop-off point. A node $x$ is admissible as a pick-up point if it has been settled by $A_1$ and $A_2$. If that’s the case, a new label $(x, \pi_x^{(3)}, \tau_x^{(3)})$ is inserted in $A_3$’s heap (computed with first line of Equation 2). A similar approach is taken for drop-off points: if $x$ was settled by $A_3$ and $A_4$, a label $(x, \pi_x^{(5)}, \tau_x^{(5)})$ is inserted in $A_4$’s heap (second line of Equation 2).

When executed on a product network, one has to make sure pick-up (resp. drop-off) nodes correspond to start states in the automaton modeling constraints on modes. Furthermore, they have to derive from nodes with accepting states in $A_1$ and $A_2$ (resp. $A_3$ and $A_4$).

**Proposition 11.** In Listing 1, a label settled by the algorithm has a cost greater than or equal to the cost of any label previously settled.

**Proof.** There are two ways of inserting a label in our algorithm: when executing one step of Dijkstra or Mono-Objective Martins and when creating a new pick-up or drop-off label. In both Dijkstra and Mono-Objective Martins, no node with lower cost might appear as an effect of settling a node. Insertion of pick-up and drop-off points is done when a node $n_i^{(j)}$ is settled and the cost of the newly created label is always greater than $\pi_n^{(i)}$ (see the previous section for the costs expressions). Thus, every newly created label’s cost will be greater or equal than the ones previously settled. Since we select the lowest label of all heaps, labels are settled by increasing cost.

**Corollary 12.** (Correctness) When the node $d_p$ is settled in $A_5$, $\pi_{d_p}^{(5)}$ is the minimal carpooling cost.

### 5.3 Restrictions on pick-up and drop-off points

A carpooling problem usually comes with preferences about where the pick-up and drop-off can occur. In this part, we present how such preferences can be used for guiding and stopping our method.

Let $Z_{up}$ be a set of nodes accessible by both the passenger and the driver. When restricting pick-up points to be in $Z_{up}$, it is easy to see that the goal of $A_1$ and $A_2$ is to settle all nodes in $Z_{up}$ and that they can stop once they have done it. This defines a stop-condition.

Furthermore, we would like to guide $A_1$ and $A_2$ towards $Z_{up}$. Suppose we have a set of consistent heuristic $h_t(u)$ that gives a lower bound of the distance from $u$ to $t$. To guide towards an area $Z$, we define $H_Z(u) = \min_{z \in Z} h_z(u)$. Combining consistent heuristics with min results in a consistent heuristic. Furthermore, $\forall z \in Z : H_Z(z) \leq 0$. Hence, $H_Z(u)$ can be used in the $A^*$ algorithm for guiding towards a set of nodes $Z$. In practice, using this heuristic results in guiding towards the closest node of the area.

However, this raises the problem of computing $|Z|$ heuristics at every iteration of the algorithm. We note as $d(u,v)$ the length of the shortest path from $u$ to $v$. For every landmark $L$ and every node $t$, algorithm ALT [5] provides us with two consistent heuristics: $h^+_t(u) = d(u,L) - d(t,L)$ and $h^-_t(u) = d(L,t) - d(L,u)$. Taking the minimum of each of those functions leads us to $H^+_Z(u) = \max_{z \in Z} d(z,L) - d(L,u)$ and $H^-_Z(u) = \min_{z \in Z} d(L,z) - d(L,u)$.

Note that $\max_{z \in Z} d(z,L)$ and $\min_{z \in Z} d(L,z)$ are not dependent on $u$ and are to be computed only...
once per carpooling problem. The final heuristic we propose to use is given by taking the max of $H^+_Z$ and $H^-_Z$ over all landmarks.

We can use this heuristic in $A_1$ and $A_2$ to guide towards $Z_{up}$. A similar approach can be taken when we are given a set $Z_{off}$ of potential drop-off points to (a) stop $A_3$ and $A_4$ once they have settled all potential drop-off points (b) guide $A_3$ and $A_4$ towards $Z_{off}$.

6 Experimental study and discussion

All experiments were conducted under Ubuntu 13.04 on an HP Pavilion g6 with 4GB of RAM. The processor is a 2.10GHz Pentium-4 with 2MB of L2 cache. Algorithms are implemented in C++ and compiled with gcc with optimization level 2. The source code is available as free software under a CeCILL-B license\(^1\). We use a multi-modal graph modeling the French regions of Aquitaine and Midi-Pyrénées. All transportation data used in these experiments are free data. Road network corresponds to the OpenStreetMap\(^2\) datasets and were provided by GeoFabrik\(^3\). Our public transportation network is based on The General Transit Feed Specification\(^4\) format. When converted into an edge-labeled multi-modal graph, it contains 629,765 nodes (21,439 of them being public transportation nodes) and about 5 millions edges (edges are duplicated for every transportation mode).

We consider 3 cities to define our instances: Toulouse, Albi and Bordeaux\(^5\). Both users start their journey in Bordeaux, the passenger is willing to go to Toulouse and the driver goes to Albi. Origins and destinations are randomly chosen in the respective cities and the start times of both users are identical during daytime (to have access to public transportation). In this configuration, passenger and driver typically meet in Bordeaux. The passenger is dropped-off in Toulouse and the driver continues his journey towards Albi. All presented results are an average over 50 of those instances using the presented integrated approach.

To measure the efficiency of stop conditions and guiding, we use two different restrictions on pick-up and drop-off points:

- **Cities**: $Z_{up}$ (resp. $Z_{off}$) contains all car accessible nodes within 20 minutes walk from Bordeaux (resp. Toulouse)’s public transports. Those areas contain respectively 29,865 and 46,584 nodes. In practice, these correspond to the whole cities.

- **10-min**: $Z_{up}$ (resp. $Z_{off}$) contains all car accessible nodes within 10 minutes by foot or public transportation from $o_p$ (resp. to $d_p$). Areas defined this way contain, on average, a few hundred nodes.

The three tested configurations are (a) **unrestricted**: the integrated approach defined in Section 5.2, (b) **stop-conditions**: stop conditions based on the areas **Cities** or **10-min** (c) **stop-conditions-guided**: stop conditions and landmarks in $A_2$, $A_3$, and $A_4$ to guide towards the pick-up and drop-off areas.

Tables 2 and 3 give the results of our method using respectively dominance rule 1 and dominance rule 2. In the first column, we give the tested configuration. The second, third, fourth and fifth columns present the runtime in ms, the number of settled labels, the number of labels per node in $A_5$ (for solving the Best Origin Problem) and the average carpooling cost over the 50 instances. There is a significant gap between the two dominance rules,

\(^1\) http://projects.laas.fr/MuPaRo/
\(^2\) http://www.openstreetmap.org/
\(^3\) http://www.geofabrik.de/
\(^4\) https://developers.google.com/transit/gtfs/
\(^5\) Bordeaux-Toulouse is a two hours and a half drive while Toulouse-Albi takes about one hour.
especially without restrictions on pick-up and drop-off areas. The average runtime with the exact dominance rule without restriction is about 48 sec. and goes down to 4.3 sec. with the heuristic rule, for a cost increasing of only 14 seconds. This gap comes from algorithm $A_5$ that has a high number of labels per nodes. With restrictions, this gap is shortened and the two rules are very close for the 10-min restriction with guided heuristic as they provide the same carpooling cost with a similar runtime. With restrictions, our algorithm has acceptable runtime with the two dominance rules. Moreover, one can see the interest of the heuristic dominance rule that leads to a small number of labels per node in $A_5$ giving a runtime performance close to Dijkstra’s on an equivalent BOP with consistency. In our instances, the two dominance rules discard many labels as, in the unrestricted configuration, there is on average 366,745 drop-off points evaluated as potential origins in the BOP.

In these tables, the stop conditions yield a major improvement. The gain of guiding our algorithms is much more noticeable for the 10-min restriction than for the Cities restriction. This difference is mainly due to the quality of our guidance-heuristic increasing with smaller areas. For the Cities restriction, carpooling solutions are mainly identical to solutions for the unrestricted variant. When considering the 10-min restriction, the cost of carpooling solutions is increasing of 259 seconds comparatively to the unrestricted variant.

<table>
<thead>
<tr>
<th>Restrictions</th>
<th>Runtime (ms)</th>
<th>Settled labels</th>
<th>Labels/node in $A_5$</th>
<th>Cost (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>48377</td>
<td>5 610 354</td>
<td>21.52</td>
<td>24 607</td>
</tr>
<tr>
<td>cities</td>
<td>6 212</td>
<td>1 135 823</td>
<td>13.99</td>
<td>24 610</td>
</tr>
<tr>
<td>cities-guided</td>
<td>5 910</td>
<td>928 487</td>
<td>13.99</td>
<td>24 610</td>
</tr>
<tr>
<td>10-min</td>
<td>603</td>
<td>374 974</td>
<td>4.54</td>
<td>24 881</td>
</tr>
<tr>
<td>10-min-guided</td>
<td>220</td>
<td>122 706</td>
<td>4.54</td>
<td>24 881</td>
</tr>
</tbody>
</table>

**Table 2** Results with dominance rule 1 (exact)

<table>
<thead>
<tr>
<th>Restrictions</th>
<th>Runtime (ms)</th>
<th>Settled labels</th>
<th>Labels/node in $A_5$</th>
<th>Cost (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>4 316</td>
<td>1 793 205</td>
<td>1.17</td>
<td>24 621</td>
</tr>
<tr>
<td>cities</td>
<td>1 139</td>
<td>585 760</td>
<td>1.26</td>
<td>24 623</td>
</tr>
<tr>
<td>cities-guided</td>
<td>853</td>
<td>378 404</td>
<td>1.26</td>
<td>24 623</td>
</tr>
<tr>
<td>10-min</td>
<td>571</td>
<td>372 959</td>
<td>1.15</td>
<td>24 881</td>
</tr>
<tr>
<td>10-min-guided</td>
<td>195</td>
<td>120 126</td>
<td>1.15</td>
<td>24 881</td>
</tr>
</tbody>
</table>

**Table 3** Results with dominance rule 2 (heuristic)

Table 4 and 5 give the average cost\(^6\) and, in brackets, number of nodes settled by each algorithm. The waiting times are respectively 97, 103, 103, 439, 439.

\(^{6}\) Recall that the cost in algorithm $A_3$ is counted twice
algorithm of our method for each dominance rule. Firstly, one can see the interest of the integrated approach comparatively to the sequential one on \(A_1\), whose exploration is limited to a small part of the network. It is not the case for \(A_2\) and \(A_4\), they both settle all nodes because the use of the car allows exploring the graph while keeping the cost low. The benefits of using the integrated approach increases with the size of the network.

Secondly, these tables show that, in terms of number of settled nodes, restrictions have an impact on all algorithms but this impact is more important for \(A_2\), \(A_4\) and \(A_5\). Algorithms \(A_2\) and \(A_4\) only consider the car mode and can, when there is no restrictions, explore the whole graph with low cost before a solution is found. Moreover, in terms of number of settled labels, the guiding variant has a light impact on \(A_2\) but a large impact on \(A_3\) and \(A_4\) since considered paths are longer.

It should be noted that the optimal drop-off point is the passenger’s destination in 29 instances (over 50) in all configurations. This leads to the average cost in \(A_5\) being small. However the passenger’s origin is never selected as the pick-up point since any waiting time is considered as part of the cost. As expected, restrictions limit the cost of the passenger’s trips, this cost being transferred on waiting time and driver’s costs.

## 7 Conclusions

In this paper, we propose a new algorithm to efficiently solve the 2SPSPP problem aiming at computing two synchronized paths for a driver and a pedestrian in a carpooling application, while minimizing the total travel time. Obtaining a solution is a matter of seconds on a large regional network. We also study the Best Origin Problem and exhibit precise conditions for which the problem can be challenging and benefits from a multi-label algorithm. For this problem, we propose exact and heuristic dominance rules.

Furthermore, it is worth noting that our approach of splitting the 2SPSPP into several Shortest Path and Best Origin Problems is very flexible and can easily be used to solve related problems. For instance, to solve two subproblems of the 2SPSPP: where the two users have the same origin or the same destination. Moreover, our approach is flexible enough so that other carpooling costs can be considered as long as consistency between costs and arrival times is preserved. But, one should notice that, even if the consistency is not preserved, the proposed method can be adapted by running our mono-objective variant of MARTINS algorithm for the best origin subproblems, leading to a more time-consuming approach.

We propose to use restricted drop-off and pick-up areas and we introduce a heuristic based on landmarks to guide the search towards these areas. These restrictions allow to obtain good solutions in less than one second, taking advantage of —highly desirable in practice— user-defined pick-up and drop-off areas with very low impact on optimality.

Future research directions include a better definition of restriction areas and integration of other acceleration techniques such as contraction hierarchies to speed up the algorithm.
We suppose in this paper that a matching was done (usually based on geographical information) between a driver and a pedestrian. Nevertheless, our method can be included to improve the matching. While we only consider two agents, our approach may be extended to a few number of pedestrians and one driver, either by using the same pick-up and drop-off points or by enumerating the set of pick-up and drop-off points. In addition, extension to a greater number of pedestrians may introduce an higher level of complexity by increasing the number of pick-up and drop-off points and by the need of re-evaluation of some paths from drop-off points to pedestrians’ destinations in the time-dependent part of the network. Further studies may then be conducted to evaluate the computational time of such approaches.

In the 2SPSPP, we consider the minimization of a carpooling cost representing the total travel time of the two itineraries for pedestrian and driver. This carpooling cost introduces a cooperation between the two agents who both aim at reducing the total cost. However, it would be interesting to consider other objectives such as cost or profit for pedestrians and drivers in addition to the cost based on travel time. The proposed method can be seen as a first step towards a multi-objective approach or a fair optimization combining combinatorial optimization and game theory.

Acknowledgements We would like to thank Dominik Kirchler for his precious comments on the Best Origin Problem and on the dominance rule.

References