Optimal 1-Request Insertion for the Pickup and Delivery Problem with Transfers and Time Horizon

José-L. Figueroa, Alain Quilliot, Hélène Toussaint and Annegret Wagler
LIMOS INP, Université Clermont Auvergne, France

Keywords: Pickup and Delivery Problem with Transfers, 1-Request Exact Insertion, Constrained Shortest Path, Constraint Propagation, Time Expanded Networks.

Abstract: In this paper, we deal with a subproblem of the Pickup and Delivery Problem with Transfers (PDPT) where a finite set of transportation requests has been assigned to a homogeneous fleet of limited capacity vehicles, while satisfying some constraints imposed by a set of pre-scheduled tours. Then, a new transportation request appears, and we also have to serve it. For that, we can modify the current tours by performing a finite sequence of changes allowing to pick up, transport, transfer or deliver this new request. The resulting tours must serve all the requests and satisfy the original constraints, but within a given time horizon. To solve this problem, we present an empirical Dijkstra algorithm that computes tentative solutions whose consistence is checked through constraint propagation, and an exact algorithm which is an adaptation of the well-known A* algorithm for robot planning, that performs an exhaustive search in a tree of partial solutions and reduces the combinatorial explosion by pruning some unfeasible/redundant tree nodes. We conclude by comparing the performance of both algorithms.

1 INTRODUCTION

In this first section, we give an informal description of the problem that we are going to treat in this paper, which is about the way that one additional request can be inserted into the current solution of a pickup and delivery instance, and we survey briefly some literature related to this problem which we call *1-Request Insertion PDPT*. This problem will be presented formally in Section 2. In Section 3, we will describe two algorithms for dealing with it. Finally, in Section 4 we compare the quality of the found solutions and the performance of both algorithms over several data sets of pseudorandom instances.

A standard *Pick up and Delivery Problem* (PDP) most often involves a finite set V of identical vehicles with capacity c, which must be scheduled in order to perform a set of Pick and Delivery tasks inside some network G. Time windows may be involved, but, in most cases, a *time horizon* $[0, T_{max}]$ is imposed for the whole schedule. Vehicles must meet a set R of requests: a request $r \in R$ consists in an origin o_r, a destination d_r and a load ℓ_r, and r has to be served by exactly one vehicle. In the *PDP with transfers* (PDPT), a request may be served by several vehicles, in the sense that load ℓ_r may start from origin o_r with some vehicle v_1, and next shift to some vehicle v_2 at some relay vertex x, and so on, until reaching destination d_r into some vehicle v_p. Depending on the context, a transfer from v_1 to v_2 may take different forms: one may impose either vehicles to meet (*strong synchronization constraint*) at relay vertex x, or only forbid vehicle v_2 from leaving x before the arrival of v_1 (*weak synchronization constraint*). In other words, a weak synchronization constraint implies that v_1 and v_2 are obliged to meet only when v_2 arrives first to the relay vertex x.

In this last case, a convenient way to model PDPT is through the use of time expanded networks (see, for example, (Gouveia et al., 2019) and (Bsaybes et al., 2019)). We construct a network (see Figure 1) whose vertex set consists of a dummy *source* vertex s, a dummy *sink* vertex p, and a copy (x,t_i) of every vertex x of G and for any time value $t_i \in [0, T_{max}]$ (note that in practice, we usually consider only a finite set of relevant time values t_i). We represent any feasible move along an arc of G, from a vertex x to a vertex y between time t_i and time $t_i + \delta$ (where $\delta \geq 0$), by some arc $\big((x,t_i),(y,t_i+\delta)\big)$. If $x = y$ then such a move becomes a *waiting* move. Then, we represent vehicles circulation by a unique integral flow vector $\mathbf{F}$ indexed over the set A of the expanded network arcs, and re-

Figueroa, J., Quilliot, A., Toussaint, H. and Wagler, A.
Optimal 1-Request Insertion for the Pickup and Delivery Problem with Transfers and Time Horizon.
DOI: 10.5220/0010787000003117
In *Proceedings of the 11th International Conference on Operations Research and Enterprise Systems (ICORES 2022)*, pages 17-27
ISBN: 978-989-758-548-7; ISSN: 2184-4372

quests circulation comes as a multi-commodity flow $\{f_r : A \to \mathbb{R}, r \in R\}$, such that for every arc $a \in A$ of the expanded network, the sum $\Sigma_{r \in R} f_r(a)$ is less than or equal to the value F_a of vector $\mathbf{F}$ on arc a.

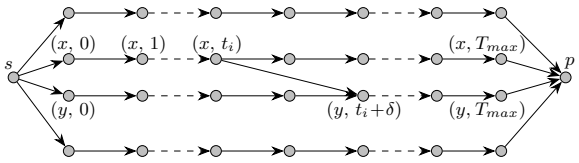

Figure 1: A *Time Expanded Network*.

Still, resulting models are not very well-fitted to numerical handling, both because of their size and because PDP tends to arise in dynamic contexts, with requests not completely known in advance, and which must be managed in a flexible way. It comes that most often, PDPT is handled in a heuristic way and a common strategy is to rely on an insertion (or build & destroy) approach: requests are successively inserted into some current schedule, and possibly removed and reinserted in order to improve the related cost.

According to this paradigm, the key issue becomes the related insertion process. In case no transfer is allowed, the insertion is trivial in the sense that it can be performed through enumeration. It is not the case when transfers are allowed. As a matter of fact, inserting a request r becomes then significantly more difficult than simply searching for some path from origin o_r to destination d_r inside some ad hoc network (or even time expanded network) because synchronization (strong or weak) constraints requirements tend to impact the whole current schedule.

So the purpose of this work is to thoroughly study the one request insertion problem that occurs as a part of the Pickup and Delivery Problem with Transfers (with time horizon), and that we will denote by *1-Request Insertion PDPT*. We are first going to solve it in an exact way, without imposing any restriction neither on the number of transfers nor on the characteristics on the transfer parameters, through a combination of constraint propagation techniques and an A* like algorithm.

Let us recall that A* algorithm (see (Nilsson, 1980) or (Dechter and Pearl, 1985)) is an artificial intelligence oriented version of Dijkstra's algorithm for shortest path, designed in order to deal with huge networks whose nodes represent the possible states of a system.

Next, in order to fit with the purpose of practical use in realistic dynamic contexts, we shall describe a heuristic algorithm, which also relies on the use of Dijkstra's algorithm, augmented with a "closure" mechanism.

Finally, we shall observe the behavior of the algorithms once we impose some common sense restrictions to the way that transfers can be performed.

Related Works

Surveys for Pickup and delivery problems can be found, for example, in (Berbeglia et al., 2007), (Berbeglia et al., 2010), and (Ho et al., 2018).

The notion of PDP with transfers was introduced by (Laporte and Mitrović-Minić, 2006) in the so-called Pickup and Delivery Problem with Time Windows and Transshipment, which is a PDP characterized for the presence of transshipment points, where the vehicles can drop some objects or split their loads to allow other vehicles to pick them up later.

An exact method for some PDPT was presented by (Contardo et al., 2010). In their paper, the authors construct a complex mixed-integer linear programming formulation, and they confirm that it works correctly by solving an example instance with a method based on Benders decomposition.

In (Bouros et al., 2011), the authors address the dynamic PDPT and propose a graph-based formulation that treats each request independently as a constrained shortest path problem. They compare this approach against a relatively conventional local search algorithm based on insertion heuristics and tabu search, and conclude that their method is significantly faster with the inconvenience that solutions quality is marginally lower.

The computational complexity of checking the feasibility for the insertion of one request in the PDPT was studied by (Lehuédé et al., 2013). In this work, the authors determined that if we perform some preprocessing of the current state of a PDPT network, we can test the feasibility of insertions in constant time, and the complexity to update the preprocessed information after insertion/deletion of one request is quadratic in the size of the network.

Above-mentioned contributions (Bouros et al., 2011) and (Lehuédé et al., 2013) are the closest ones to our contribution. Both rely on the construction of auxiliary graphs, which represent the current state of vehicle paths together with some specific constraints related to transfers. Still (Bouros et al., 2011) does not care of time's feasibility (i.e. time windows) nor the impact of restrictions imposed to the transfers. At the opposite, (Lehuédé et al., 2013), is a theoretical contribution which focuses on the complexity of testing the feasibility of an insertion (with one transfer at most), after some preprocessing that allows the management of slack time variables. Our contribution lays in between: while we also rely on precom-

putation of length path values, we relax the restrictions on the transfer vertices, impose no restriction on the number of transfers, and propose both exact and heuristic algorithms for the computation of a best feasible insertion for a given request. Then we analyze through numerical experiments, the impact on both the behavior of the algorithms and the nature of the solutions.

2 FORMAL MODEL

We first suppose that we are provided with a finite set X of "points" representing physical locations, given together with:

- A *distance function* $d : X \times X \to \mathbb{R}^+$, such that for any $x, y \in X$, the value $d(x,y)$ means the time required for a vehicle to move in X from x to y according to a shortest path strategy.
- A function $\mu : X \times X \to X$ is going to be involved in order to determine "relay vertices", that means the points where two vehicles v_1, v_2 will meet in order to perform some transfer. Intuitively, for any $(x_1, x_2) \in X \times X$, the element $u = \mu(x_1, x_2)$ corresponds to a place that is more or less at the same distance from x to y and minimizes the sum $d(x_1,u) + d(u,x_2)$. For example, if X contains points from a Euclidean space, we can define $u = \mu(x_1, x_2)$ as an element in X with minimal distance to the midpoint of the segment from x_1 to x_2.

2.1 PDPT Problem

A PDPT instance consists of a finite set of points X together with the two functions $d : X \times X \to \mathbb{R}^+$ and $\mu : X \times X \to X$ that we have just defined, a finite set of vehicles V with a homogeneous capacity $c \in \mathbb{R}^+$, a *time horizon* $[0, T_{max}]$, and a finite set $R \subset X \times X \times \mathbb{R}^+$ of *requests* $r = (o_r, d_r, \ell_r)$. The point o_r is the *origin* of r, the point d_r is the *destination* of r, and ℓ_r is the *load* of r. Also, each vehicle $v \in V$ has assigned two distinguished points: one *starting depot point* $D_s^v \in X$ and one *ending depot point* $D_e^v \in X$. Figure 2 shows an example of PDPT instance.

Now we will briefly describe the PDPT. We are not going to deal here with the whole problem, however, for completeness reasons it is important to give at least some intuition and clarify some details involved. This will set the stage later to present our formal model for the insertion subproblem.

The PDPT consists in finding a schedule for the vehicles fleet, allowing to transport the requests from their origins to their destinations. Vehicles capacity must be respected at any moment, however, vehicles are allowed to transfer requests to other vehicles through a *weak synchronization* mechanism. This is, if a vehicle v is transferring a load ℓ_r to another vehicle w at a relay vertex z, and the receiving vehicle w arrives first to z, then w has to wait for the arriving of v with the load ℓ_r. The time duration for traversing every path in the schedule (considering waiting times), must be within the given time horizon.

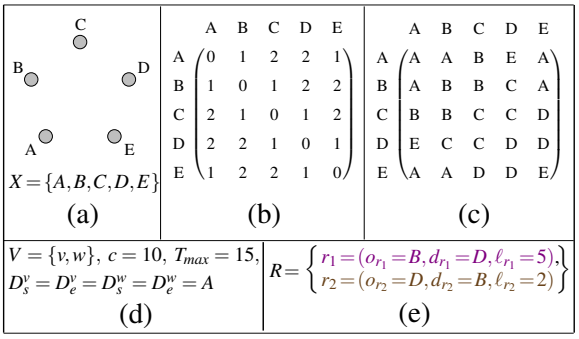

Figure 2: An example of PDPT instance. (a) The set $X = \{A,B,C,D,E\}$ of points. (b) A symmetric matrix whose entries define the distance function d. (c) A matrix defining the function μ. (d) The set of vehicles $V = \{v,w\}$, the capacity c, the upper limit of the time horizon $[0, T_{max}]$, and the starting and ending depot points of vehicles. (e) The set R of requests consisting of $r_1 = (B,\ D,\ 5)$ and $r_2 = (D,\ B,\ 2)$.

Depending on the context, there can be several ways to define the cost of such schedules. Here, we put the focus on the point of view of the users, and consider that the cost of a schedule is the sum of the distances traversed by the requests, and then we aim to find a minimal cost schedule. This choice is not particularly restrictive because the algorithms that we will present can be adapted to handle other types of costs (e.g. total mileage covered by the vehicles/ requests, or the time when they achieve their respective trips).

In the following subsection, we proceed to describe our formal model for the insertion problem involved in the PDPT.

2.2 Formal Description of a PDPT Feasible Solution

Let G_X be the directed graph whose set of vertices is X and whose set of arcs is $\{(x,y) \in X \times X,\ x \neq y\}$.

According to the above definition of a PDPT instance, we define a *solution* (Γ, Π) for the PDPT instance, which we also call a *PDPT schedule* as

- A collection $\Gamma = \{\Gamma(v),\ v \in V\}$ of paths on G_X, where $\Gamma(v) = (x_0^v, x_1^v, \ldots, x_{|\Gamma(v)|-1}^v)$ is the path followed by vehicle v in G_X.

- A collection $\Pi = \{\Pi(r),\ r \in R\}$, of paths on G_X. The directed path $\Pi(r) = (y_0^r, y_1^r, \ldots, y_{|\Pi(r)|-1}^r)$ is the path followed by load ℓ_r when moving from its origin o_r to its destination d_r. Points y_q^r belong to the set $\{x_i^v : v \in V, 0 \le i \le |\Gamma(v)|-1\}$ and we may distinguish two classes of moves for request r
  - If y_q^r and y_{q+1}^r are related to two consecutive vertices of path $\Gamma(v)$, then they have to be consecutive in $\Pi(r)$ and so we talk about vehicle move inside vehicle v.
  - If $y_q^r = x_i^v$ and $y_{q+1}^r = x_j^w$ are related to two distinct paths $\Gamma(v)$ and $\Gamma(w)$, then they refer to the same vertex in X, and so we talk about transfer move from vehicle v to vehicle w.

Denoting by $X(\Gamma)$ the set $\{(v,i) : v \in V, 0 \le i \le |\Gamma(v)|-1\}$, we see that such a solution induces an oriented graph structure $G(\Gamma,\Pi)$ on the vertex set $X(\Gamma)$, with arc set $A(\Gamma,\Pi)$ defined as follows

- With any $v \in V$ and any $0 \le i \le |\Gamma(v)|-1$, we associate a vehicle arc $e = ((v,i),(v,i+1))$, with length $d^G(e) = d(x_i^v, x_{i+1}^v)$. Following paths $\Pi(r)$ allows the computation of the load ℓ_i^v of this arc, which is the sum of loads ℓ_r for requests r which move through this arc. Of course ℓ_i^v does not have to exceed the capacity c (*Load Constraints*).
- With any $r \in R$ and $1 \le q \le |\Pi(r)|-1$, such that moving from $y_q^r = x_i^v$ to $y_{q+1}^r = x_j^w$ corresponds to a transfer move from vehicle v to vehicle w, we associate a transfer arc $e = ((v,i),(w,j))$, with length $d^G(e) = d(y_q^r, y_{q+1}^r) = 0$ (because in this case, y_q^r and y_{q+1}^r correspond to the same place). We denote by A' the set of those transfer arcs.

Figure 3 shows an example of solution (Γ,Π) for the instance defined in Figure 2, and the associated graph $G(\Gamma,\Pi)$.

Feasibility of (Γ,Π)**:** As we just told, (Γ,Π) must meet the above Load Constraints. Also it must satisfy time consistency constraints defined as follows:

- With any vertex (v,i) in the graph $G(\Gamma,\Pi)$ we associate a time value t_i^v which represents the earliest time when vehicle v may leave vertex (v,i). Then we see that
  - For any vehicle arc $((v,i),(v,i+1))$, we have that $t_{i+1}^v \ge t_i^v + d(x_i^v, x_{i+1}^v)$.
  - Weak synchronization implies that $t_j^w \ge t_i^v$ for any transfer arc $((v,i),(w,j))$.

They can be summarized as: (E1*: Time Consistency Constraints*)

- For any arc $e = ((v,i),(w,j))$ in the graph $G(\Gamma,\Pi)$, we have that $t_j^w \ge t_i^v + d^G(e)$.

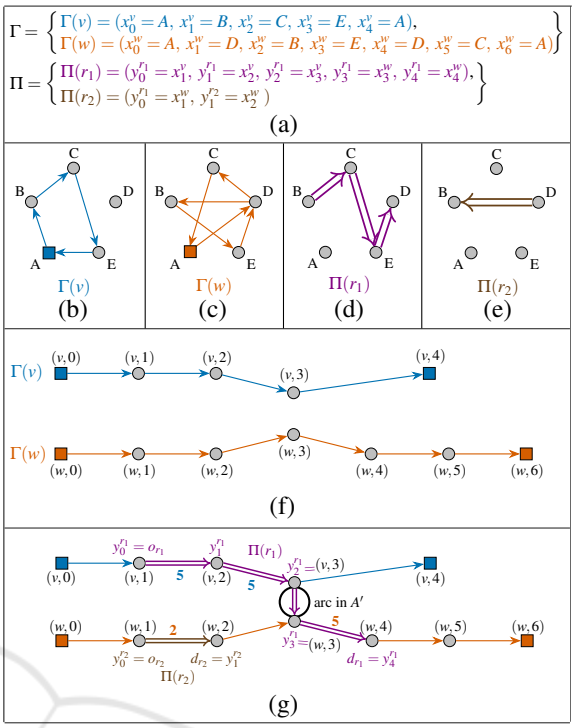

Figure 3: (a) An example of solution for the instance defined in Figure 2. (b)-(e) Subgraphs of G_X edge-induced by the paths $\Gamma(v)$, $\Gamma(w)$, $\Pi(r_1)$ and $\Pi(r_2)$, respectively. (f) The paths on the directed graph $G(\Gamma,\Pi)$ corresponding to the directed paths $\Gamma(v)$ and $\Gamma(w)$ of G_X. (g) The directed graph $G(\Gamma,\Pi)$. We have also depicted with double arrows the directed paths corresponding to $\Pi(r_1)$ and $\Pi(r_2)$. Numbers indicate positive loads traversing through arcs in $A(\Gamma,\Pi)\setminus A'$.

- Due to the time horizon $[0, T_{max}]$, for any (v,i) we have that $0 \le t_i^v \le T_{max}$.

They impose that: (E2*: No Cycles Constraint*)

- Arc set $A(\Gamma,\Pi)$ does not contain any cycle.

If (Γ,Π) satisfies those constraints, we can associate, with every (v,i) in $G(\Gamma,\Pi)$, some time windows $[Min_i^v, Max_i^v]$ related to variables t_i^v, constrained by (E1).

Cost of a Solution (Γ,Π)**:** As we said in 2.1, we are going to define the *cost* of a solution as the sum of the distances traversed by the requests.

2.3 The 1-Request Insertion PDPT Model

Those preliminaries allow us to define in a formal way the 1-Request Insertion PDPT, about the insertion of a new request into a current feasible PDPT schedule (Γ,Π). So we start from such a feasible schedule (Γ,Π), and from an additional request $r = (o_r, d_r, \ell_r)$.

Intuitively, inserting request r means building a suitable sequence of the following five types of moves:

(1) Start from o_r and enter into some path $\Gamma(v)$ at the level of some vertex $(v,i+1)$, and so imposing vehicle v to make a deviation between (v,i) and $(v,i+1)$. Such a move will be performed once as the initial move (See Figure 4 (a)).

(2) Leave some path $\Gamma(w)$ at the level of some vertex (v,i) in order to reach d_r, and so impose a vehicle to make a deviation between (v,i) and $(v,i+1)$. Such a move will be performed once as the final move (See Figure 4 (b)).

(3) Keep on inside vehicle v, while moving from some (v,i) to its successor $(v,i+1)$. Such a type of move will be possibly performed several times (See Figure 4 (c)).

(4) Move from vehicle v to vehicle w while using some arc $((v,i),(w,j))$ of A' and some shared point $x_i^v = x_j^w$. Such a type of move will be possibly performed several times (See Figure 4 (d)).

(5) Move from vehicle v to vehicle w while getting out of v while running between some vertex (v,i) and its successor and entering into w at the level of some vertex (w,j), and while w is moving from $(w,j-1)$ to its successor. Such a move will be possibly performed several times (See Figure 4 (e)).

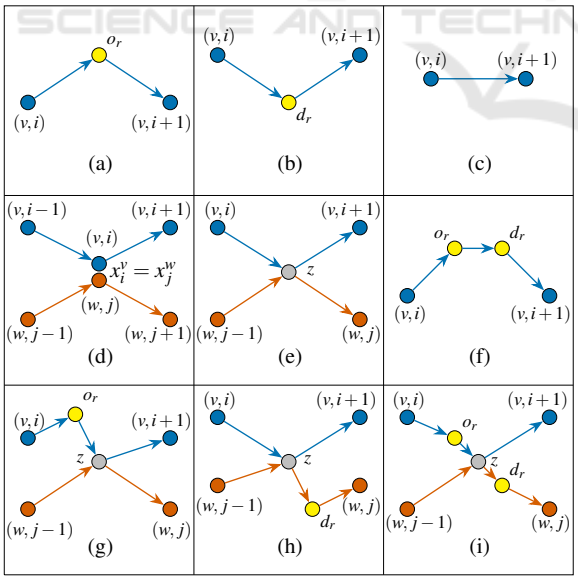

Figure 4: Types of vehicle's moves for transporting a request $r = (o_r, d_r, \ell_r)$.

Remark: For simplicity, in this work we are going to restrict our study to these 5 types of moves. However, we note that it is also possible to combine:

(6) moves of types (1) and (2), in such a way that r is inserted through a simple deviation of a vehicle v between two successive vertices of $\Gamma(v)$. This is, while traversing $\Gamma(v)$, vehicle v performs a deviation from (v,i) to transport the load ℓ_r from o_r directly to d_r, and then v returns to its original path at the level of vertex $(v,i+1)$ (See Figure 4 (f)).

(7) movements of types (1) and (5), in such a way that, request r starts from o_r in a vehicle v and then is transferred directly to another vehicle w at some relay vertex z, while v is moving from (v,i) to $(v,i+1)$, and while w is moving from $(w,j-1)$ to (w,j) (See Figure 4 (g)).

(8) movements of types (2) and (5), in such a way that r leaves some path $\Gamma(v)$ (at the level of a vertex (v,i)), to be transferred from v to w at some relay vertex z. Then r reaches d_r directly from z in vehicle w. The first part of this process happens while v is moving from (v,i) to $(v,i+1)$, and the second one while w is moving from $(w,j-1)$ to (w,j) (See Figure 4 (h)).

(9) moves of types (1), (2) and (5), in such a way that r starts from o_r in a vehicle v, then r is transferred directly to another vehicle w at some relay vertex z, and finally r reaches d_r directly from z inside vehicle w, while v is moving from (v,i) to $(v,i+1)$, and while w is moving from $(w,j-1)$ to (w,j) (See Figure 4 (i)).

While moves of types 1, 2, 3, and 4 are easy to understand, we need to better explain the moves of type 5. Clearly, a move of type 5 involves some relay vertex z: both vehicle v and w are going to perform a deviation through z, vehicle v will drop load ℓ_r in z, and vehicle w will take it as soon as possible. In order to identify z, we use function μ and we take:

$$z = \mu\big(\mu(x_i^v, x_j^w), \mu(x_{i+1}^v, x_{j-1}^w)\big). \quad \text{(E3)}$$

Please notice that (E3) restricts our freedom to choose the relay vertex, and so it has to be considered as a hypothesis of the model.

Now, given a PDPT schedule (Γ,Π) and one additional request r, we define the following graph $H(\Gamma,\Pi,r)$

- Vertices of $H(\Gamma,\Pi,r)$ are vertices of $G(\Gamma,\Pi)$, plus two vertices o_r and d_r.
- Arcs of $H(\Gamma,\Pi,r)$ are
  - *In-arcs*: they have the form $e = (o_r,(v,i))$, with $0 \le i < |\Gamma(v)| - 1$, length $d^H(e) = d(o_r, x_{i+1}^v)$, and such that $\ell_i^v + \ell_r \le c$.
  - *Out-arcs*: with the form $e = ((w,j),d_r)$, with $1 \le j \le |\Gamma(w)| - 1$, length $d^H(e) = d(x_j^w, d_r)$, and such that $\ell_j^w + \ell_r \le c$.

– *Vehicle-arcs*: arcs $e = ((v,i),(v,i+1))$ in $G(\Gamma,\Pi)$, with length $d^H(e) = d^G(x_i^v, x_{i+1}^v)$, and such that $\ell_i^v + \ell_r \leq c$.
– *A′-arcs*: the arcs $e = ((v,i),(w,j)) \in A'$, with length $d^H(e) = d^G(e) = 0$.
– *Transfer-arcs*: with the form $e = ((v,i),(w,j))$, length $d^H(e) = d(x_i^v, z) + d(z, x_j^w)$ where z is defined as in (E3). These arcs are such that $\ell_i^v + \ell_r \leq c$ and $\ell_{j-1}^w + \ell_r \leq c$.

Let π be a path from o_r to d_r in $H(\Gamma,\Pi,r)$. Note that π may be interpreted as a sequence of moves of types $(1,\ldots,5)$ allowing to transport the request r from o_r to d_r.

If t_i^v denotes the time when vehicle v leaves vertex (v,i), then every in-arc, out-arc and transfer-arc of π is going to impose additional constraints, to be added to constraints (E1)

- An in-arc $\big(o_r,(v,i)\big)$ imposes one additional constraint $t_{i+1}^v \geq t_i^v + d(x_i^v, o_r) + d(o_r, x_{i+1}^v)$. (E4)
- An out-arc $\big((w,j),d_r\big)$ imposes one additional constraint $t_{j+1}^w \geq t_j^w + d(x_j^w, d_r) + d(d_r, x_{j+1}^w)$. (E5)
- A transfer-arc $\big((v,i),(w,j)\big)$ imposes three additional constraints (here, z is taken as in (E3))

(E6)

1. $t_{i+1}^v \geq t_i^v + d(x_i^v, z) + d(z, x_{i+1}^v)$. (E6.1)
(*Deviation for vehicle v*)
2. $t_j^w \geq t_{j-1}^w + d(x_{j-1}^w, z) + d(z, x_j^w)$. (E6.2)
(*Deviation for vehicle w*)
3. $t_j^w \geq t_i^v + d(x_i^v, z) + d(z, x_j^w)$. (E6.3)
(*Weak Synchronization*)

Then a path π in $H(\Gamma,\Pi,r)$ is going to be *time-consistent* if it allows the existence of time vectors $\mathbf{t}^v = (t_0^v,\ldots,t_{|\Gamma(v)|-1}^v)$, $v \in V$, which meet constraints (E1) and additional constraints (E4, E5, E6) related to π. Notice that if π is given, checking the time consistence of π can be performed through a simple constraint propagation process.

Figure 5 shows an example of construction of the graph $H(\Gamma,\Pi,r)$ from a PDPT schedule (Γ,Π) and one additional request $r = (o_r, d_r, \ell_r)$.

The 1-Request Insertion PDPT can be summarized as follows:

1-Request Insertion PDPT: Given a PDPT schedule (Γ,Π) and one request $r = (o_r, d_r, \ell_r)$, compute a shortest (in the sense of length function d^H) time-consistent path π from o_r to d_r in the graph $H(\Gamma,\Pi,r)$.

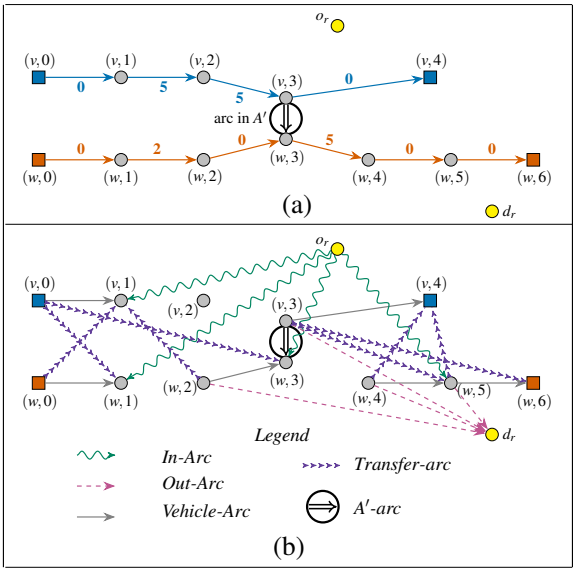

Figure 5: Deriving graph $H(\Gamma,\Pi,r)$ from the PDPT schedule (Γ,Π) corresponding to Figure 3, and one additional request $r = (o_r = D, d_r = B, \ell_r = 9)$. (a) The graph $G(\Gamma,\Pi)$ and the additional request $r = (o_r, d_r, \ell_r)$. (b) The graph $H(\Gamma,\Pi,r)$. Notice that the capacity requirement forbids some vehicle-arcs of $G(\Gamma,\Pi)$.

3 ALGORITHMS

We are first going to work (in 3.1 and 3.2) considering all the transfers-arcs with enough capacity to handle the given request. However, we can notice that the number of transfer-arcs increases as $\sum_{v,w \in V, v \neq w} |\Gamma(v)| \cdot |\Gamma(w)|$, and then we shall adapt in 3.3 the algorithms described in 3.1 and 3.2 in such a way that they consider only well-fitted eligible transfer-arcs.

3.1 An Exact A*-like Algorithm

The A* algorithm was introduced in (Hart et al., 1968) as an adaptation of Dijkstra's algorithm in order to deal with search path in very large state networks, like those that one may have to handle in robotics. It is our case here since at any time during the resolution process, we shall deal not only with a current vertex (v,i) of the graph $H(\Gamma,\Pi,r)$, but also with time windows related to vectors $\mathbf{t}^v$, $v \in V$.

We suppose that we are provided, as consequence of some constraint propagation preprocess, with

- For any pair $\big((v,i),(w,j)\big)$ of vertices of $G(\Gamma,\Pi)$, the length $\Phi\big((v,i),(w,j)\big)$ of a longest path from (v,i) to (w,j) in the graph $G(\Gamma,\Pi)$; variables t_i^v and t_j^w are constrained by $t_j^w \geq t_i^v + \Phi\big((v,i),(w,j)\big)$.

- For any vertex u in the graph $H(\Gamma,\Pi,r)$, the length ω_u of a shortest path from u to destination vertex d_r in $H(\Gamma,\Pi,r)$. In case $u \in V(\Gamma(v))$ for some $v \in V$ we are also provided with a time window $[Min_i^v, Max_i^v]$ derived from constraints (E1).

Then, as in standard A*, we perform a Breadth First Search (BFS) process, driven by some list *LS* of states, according to the algorithmic scheme described in Algorithm 1.

Algorithm 1: A* 1-PDPT algorithmic scheme.

input : Graph $H(\Gamma,\Pi,r)$, request $r = (o_r, d_r, \ell_r)$.
output: A time-consistent (o_r, d_r)-path with best cost or a failure message.

```
LS ← {(o_r, 0, 0, 0, Nil)}   [Initialize LS] ;
Success ← False ;
while LS Not empty and Not Success do
    (u, λ, δ, ω, S) ← Head(LS) [Current state];
    Remove Head(LS) from LS;
    if u ≠ d_r then
        Expand current state (u, λ, δ, ω, S)  (I1);
    else
        Success ← True;
    end
end
if Not Success then
    Print "No path found";
else
    return: Path retrieved from current state.
end
```

State's Definition: Every element s of *LS* is a *state*, that means a 5-uple $s = (u,\lambda,\delta,\omega,\mathcal{S})$, where

- u is a vertex of the graph $H(\Gamma,\Pi,r)$.
- $\lambda \in \mathbb{R}$ is the value of the current length of the path $\pi(u)$ from origin o_r to u which has been achieved.
- $\delta \in \mathbb{R}$ is a lower bound on the arriving time to u. In case $u = (v,i) \in V(\Gamma(V))$ for some $v \in V$, we will write δ_i^v instead of δ; feasibility of state s implies that $Min_i^v \le \delta_i^v \le Max_i^v$.
- $\omega \in \mathbb{R}$ is the sum $\omega_u + \lambda$, that is a lower bound for any path π which would extend current path $\pi(u)$; *LS* remains sorted in increasing order according to ω values.
- $\mathcal{S}$ is a list of transfer-arcs $\big((v,i),(w,j)\big)$ which have been involved in $\pi(u)$, together with lower bounds $\delta_i^v, \delta_{i+1}^v, \delta_{j-1}^w, \delta_j^w$, on t_i^v, t_{i+1}^v, t_{j-1}^w, and t_j^w, respectively.

Domination Rule: Let $\pi(u)$ and $\pi(u')$ be time-consistent paths retrieved from two different states $s = (u,\lambda,\delta,\omega,\mathcal{S})$ and $s' = (u',\lambda',\delta',\omega',\mathcal{S}')$, respectively. For any $v \in V$, let $\mathbf{t}_s^v$, $\mathbf{t}_{s'}^v$, respectively, be the lexicographically smallest time vectors (meeting E1 and E4-E6) related to $\pi(u)$ and $\pi(u')$. Let $V_u = \{v \in V : V(\pi(u)) \cap V(\Gamma_v) \neq \emptyset\}$ and $V_{u'} = \{v \in V : V(\pi(u')) \cap V(\Gamma_v) \neq \emptyset\}$.

We say that the state s *dominates* the state s' if $u = u'$, $\lambda < \lambda'$, $\delta \le \delta'$, $V_u \subseteq V_{u'}$ and for all $v \in V_u \cap V_{u'}$ we have that $\max\{j : (v,j) \in V(\pi(u))\} \le \max\{j : (v,j) \in V(\pi(u'))\}$, and for every $v \in V$ we have that $\mathbf{t}_s^v \le \mathbf{t}_{s'}^v$. This rule can be used to remove some redundant states when we are exploring the BFS tree of states.

Expand Mechanism `(I1)` **:** Expanding state $s = (u,\lambda,\delta,\omega,\mathcal{S})$, means consider the arcs $e = (u,v)$ of $H(\Gamma,\Pi,r)$ with tail u and, for any such an arc, computing resulting state $s'_e = (v, \lambda' = \lambda + d(e), \delta', \omega' = \lambda' + \omega_v, \mathcal{S}')$, where δ' and $\mathcal{S}'$ derive from δ and $\mathcal{S}$, respectively, after propagation of constraints E1, E4, E5, E6. If the path $\pi(v)$, retrieved from s'_e, is time-consistent and s'_e is not dominated by any state in *LS*, then we insert s'_e into *LS* and possibly remove from *LS* any state dominated by s'_e. Note that, the constraint propagation mechanism which yields δ' and $\mathcal{S}'$ is induced by the transfer-arcs which have been used in order to reach u, and which are contained into the list $\mathcal{S}$.

Figure 6 contains a representation of a state from a list *LS* and gives some intuition about the expanding mechanism `(I1)`.

Now, we note that

- The A* algorithm performs an exhaustive search on the tree of states corresponding to time-consistent paths.
- A path which is not time-consistent cannot be extended into a time-consistent one.
- The domination rule removes only redundant states.

Hence we have the following.

Proposition 1. *Above A* 1-PDPT solves 1-Request Insertion PDPT in an exact way (taking into account the (E3) hypothesis), in a time which does not exceed* $O\big(\big(\Delta^+(H(\Gamma,\Pi,r))\big)^{m+1} \cdot Size(H(\Gamma,\Pi,r))\big)$ *where* $\Delta^+(H(\Gamma,\Pi,r))$ *is the maximum outdegree of a vertex in* $H(\Gamma,\Pi,r)$*, and where m is the maximum number of arcs in a path* $\pi(u)$ *retrieved from a state* $s = (u, \lambda, \delta, \omega, \mathcal{S})$ *that was involved in LS.*

Remark: The complexity of the 1-Request Insertion PDPT remains an issue. Still, in the case we impose a small constant (e.g. 3 or 5) as a bound on the number of transfer-arcs of the constructed paths, a simple counting argument can show that it becomes time-polynomial.

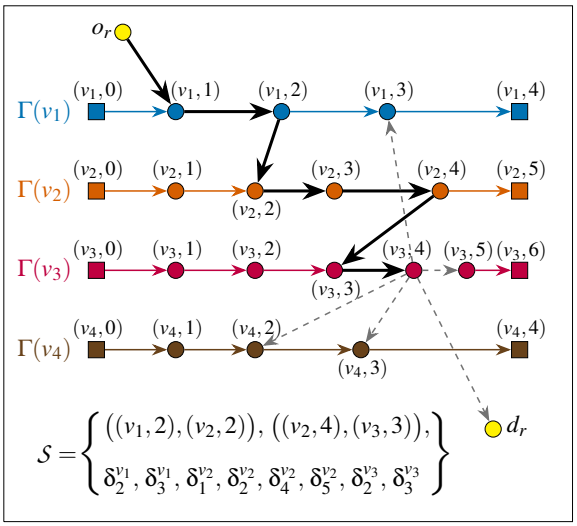

Figure 6: Representation of a particular state $s = (u = (v_3,4), \lambda, \delta, \omega, \mathcal{S})$ from a list LS. Bold arcs indicate the corresponding path $\pi(u)$ from origin o_r to u which has been achieved. The information stored in $\mathcal{S}$ allow us to determine which vertices are involved in path $\pi(u)$, and also give us estimations for the arriving times to those vertices. Dashed arcs allow extending the current path $\pi(u)$, so they correspond to new states that can be derived from s via the expanding mechanism `(I1)`.

3.2 A Practical Dijkstra-like Algorithm

Since our problem is likely to arise in a dynamic context, we must try to propose approaches which are less time consuming than the previous one. In order to do it, we fix some integer parameter $m > 0$, and next for every (v,i) with $v \in V$, $i \in \{0,\ldots,|\Gamma(v)|-1\}$, we compute shortest paths $\pi(v,i)$, in the graph $H(\Gamma,\Pi,r)$, from vertex (v,i) to destination d_r, while using Dijkstra's algorithm (see Figure 7 (a)). We denote by $\omega(v,i)$ the length of $\pi(v,i)$. Notice that such computations are involved in A* 1-PDPT algorithm, in order to provide us with ω values involved in Algorithm 1.

Then, we close those computed paths $\pi(v,i)$ (see Figure 7 (b)) to obtain (o_r,d_r)-paths $\pi^*(v,i) = (o_r,(v,i)) + \pi(v,i)$ (i.e. we create a path from o_r to d_r whose first arc is the in-arc $(o_r,(v,i))$ and the remaining arcs are the ones of path $\pi(v,i)$). For each constructed path $\pi^*(v,i)$, we compute the value $d\big(o_r,(v,i)\big) + \omega(v,i)$, and then we select the m paths $\pi^*(v_1,i_1),\ldots,\pi^*(v_m,i_m)$ with the lowest values $d\big(o_r,(v,i)\big) + \omega(v,i)$. Finally, we check for every selected path $\pi^*(v_k,i_k)$, $1 \le k \le m$, the consistence of the additional constraints induced by the in-arc, the out-arc and the transfer-arcs of $\pi^*(v_k,i_k)$, and we keep the path π^* which meets this consistence test and which is related to the smallest $d\big(o_r,(v,i)\big) + \omega(v,i)$ value. This process is summarized in Algorithm 2.

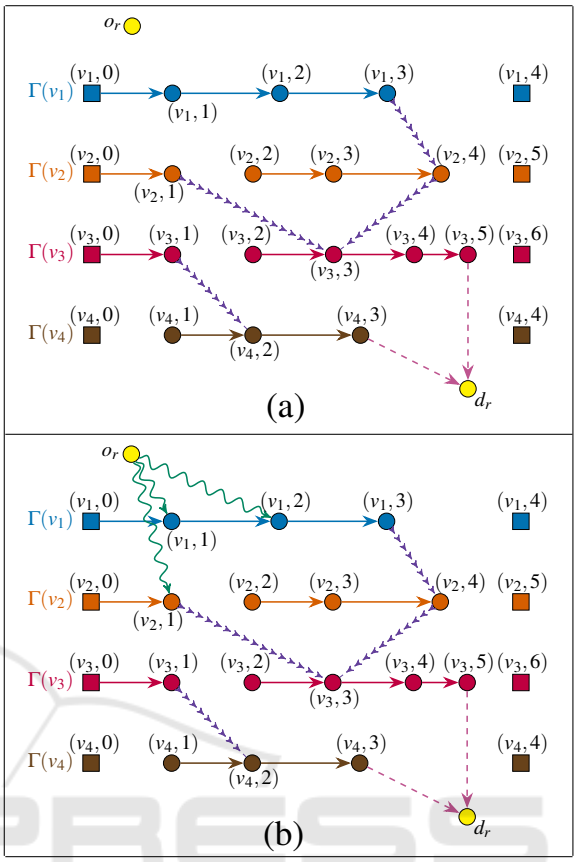

Figure 7: Deriving solutions from a Dijkstra's shortest path computation. (a) A subgraph Ψ of a graph $H(\Gamma,\Pi,r)$ obtained by Dijkstra's algorithm while computing a shortest path $\pi(v,i)$ from every vehicle vertex (v,i) to d_r. (b) Graph Ψ is extended by adding some in-arcs from $H(\Gamma,\Pi,r)$ (wavy arrows). Every path from o_r to d_r in the resulting graph gives rise to a solution for handling request r, and the time feasibility of such a solution is then checked through constraint propagation.

3.3 Controlling Transfer-arcs Number

As mentioned at the beginning of Section 3, the number of transfer-arcs is an issue, since most arcs of $H(\Gamma,\Pi,r)$ are transfer-arcs, while at the end, the best time-consistent paths usually contain very few of such arcs. In order to deal with the number of transfer-arcs issue, we impose to transfer-arcs some eligibility requirement, which depends on a threshold parameter $\varepsilon \ge 0$.

ε-eligibility of a transfer-arc $\big((v,i),(w,j)\big)$: we say that transfer-arc $e = \big((v,i),(w,j)\big)$ is ε-*eligible* if $d(x_i^v,x_j^w) \le \varepsilon$ and the intersection of time windows $[Min_i^v + d(e), Max_i^v + d(e)]$ and $[Min_j^w, Max_j^w]$ is non-empty.

By fixing ε and imposing transfer-arcs in both A* 1-PDPT and Dijkstra 1-PDPT algorithms to be ε-

Algorithm 2: *Dijkstra 1-PDPT*.

input : Directed graph $H(\Gamma,\Pi,r)$ together with the distance function d^H.
output: A time-consistent (o_r, d_r)-path π^*, or a failure message.

$\pi^* \leftarrow$ Nil ;
For every (v,i) with $v \in V$ and $i \in \{0,\ldots,|\Gamma(v)|-1\}$, compute through Dijkstra's algorithm, shortest paths $\pi(v,i)$ from (v,i) to destination d_r, together with their corresponding length $\omega(v,i)$ in the graph $H(\Gamma,\Pi,r)$;
Close every path $\pi(v,i)$ computed in step 2 by adding the corresponding *in-arc* $\big(o_r,(v,i)\big)$ and select the m resulting paths $\pi^*(v_1,i_1),\ldots,\pi^*(v_m,i_m)$ with best $d\big(o_r,(v,i)\big)+\omega(v,i)$ values;
For every path $\pi^*(v_k,i_k), 1 \le k \le m$ selected in step 3, test its time-consistence (through constraint propagation) and keep as π^* the time-consistent path $\pi^*(v_k,i_k)$ with best $d\big(o_r,(v_k,i_k)\big)+\omega(v_k,i_k)$ value;
if $\pi^* =$ Nil **then**
    Print "No path found";
else
    return : π^*
end

eligible, we become able to control running times. As Section 4 will show, this control is not usually going to induce any significant loss in solutions quality.

4 NUMERICAL EXPERIMENTS

Purpose: We have been performing experiments to

1. Analyze the performance of both A* 1-PDPT and Dijkstra 1-PDPT algorithms from the point of view of running costs and, in the case of Dijkstra 1-PDPT, of gap to optimality.
2. Evaluate the impact of the ε-eligibility on those algorithms.
3. Estimate the potential interest of transfers, through both the number of transfer-arcs involved in an optimal path π, and the gain induced by those arcs.

Technical Context: Points of X are randomly generated on an integral grid with size $n \times n$, distance function *dist* is the upper rounding of the Euclidean distance (indicated by E) or the Manhattan distance (indicated by M). The upper limit of the time horizon $[0, T_{max}]$ is indicated by T_{max} and is generated as a linear function of n, also we will consider a fixed capacity $c = 10$.

While designing the paths, we follow several strategies

- *Strategy* F: For every $v,w \in V$ we that $D_s^v = D_e^v = D_s^w = D_e^w$. This is, all of the vehicles start and end their paths in a common depot point. Paths $\Gamma(v)$ are generated in a pseudorandom way.
- *Strategy* C: For every $v,w \in V$ we have $D_s^v = D_e^v$ and $D_s^w = D_e^w$. This is, every vehicle start and end its path in a same depot point, but different vehicles can have distinct depot points. Paths $\Gamma(v)$ are generated by following some linear or circular orientation.
- *Strategy* O: Every vehicle v starts its path in a depot point D_s^v and ends its path in a depot point D_e^v (possibly different of D_s^v); furthermore, different vehicles can have distinct origin depot points and different destination depot points. Paths $\Gamma(v)$ are generated by following some linear or circular orientation.

The designed paths have around ten arcs on average, and the arcs loads are generated in a uniform random way by choosing elements from the set $\{1,2,\ldots,10\}$.

Another important path feature is the mean ratio

$$\rho = \sum_{v \in V} \frac{T_{max}}{Length(\Gamma(v))},$$

which gives us an approximate idea about the difficulty of an instance.

The sets of arcs A' are created in the following way: for every $(v,i) \in \Gamma(v)$ and every $(w,j) \in \Gamma(w)$, such that $x_i^v = x_i^w$ and $\big((w,j),(v,i)\big) \notin A'$ we create an arc $\big((v,i),(w,j)\big)$ and we add it to A' with a probability of 0.5 if the resulting graph is acyclic and does not imply exceeding the time horizon.

Requests are also generated in a random way, for this, we have selected two different points o_r and d_r from X, and a random load value ℓ_r from the set $\{1,2,3,4,5\}$.

Instances: We have that, an instance is summarized by the following characteristics: the side's length n of the squared integer grid, the number $|X|$ of points, the distance function *dist* (E= Euclidean, or M= Manhattan), the number of vehicles $|V|$, the number $|A'|$ of arcs in A', the upper limit T_{max} of the time horizon, the mean ratio ρ, the path generation strategy St, and a request $(o_r,d_r,\ell_r) \in X \times X \times \{1,2,3,4,5\}$.

Instances which are going to be used here are summarized by the rows of Table 1. Note that every instance with an odd index k (i.e. the number in the Id column), shares all of its elements, up to the distance

function and the mean ratio ρ, with the instance with index $k+1$.

Table 1: Instance characteristics.

Id	n	$\|X\|$	$dist$	$\|V\|$	$\|A'\|$	T_{max}	ρ	St	ℓ_r
1	25	80	M	3	1	75	1.75	F	3
2	25	80	E	3	1	75	1.90	F	3
3	30	90	M	4	1	90	1.20	C	2
4	30	90	E	4	1	90	1.34	C	2
5	35	100	M	5	2	105	1.43	O	4
6	35	100	E	5	2	105	1.82	O	4
7	40	110	M	6	4	120	1.31	F	5
8	40	110	E	6	4	120	1.53	F	5
9	45	120	M	7	3	135	1.22	C	2
10	45	120	E	7	3	135	1.41	C	2
11	50	130	M	8	4	150	1.41	O	4
12	50	130	E	8	4	150	1.64	O	4
13	55	140	M	9	5	165	1.26	F	3
14	55	140	E	9	5	165	1.52	F	3
15	60	150	M	10	3	180	1.18	C	1
16	60	150	E	10	3	180	1.38	C	1
17	65	160	M	11	6	195	1.41	O	4
18	65	160	E	11	6	195	1.67	O	4
19	70	170	M	12	7	210	1.26	F	3
20	70	170	E	12	7	210	1.49	F	3

Outputs: For any instance we compute

1. The value VA* computed by A* 1-PDPT, together with related running time TA* in seconds and without including preprocessing, the number TrA* of transfer arcs of the solution.
2. The value VD computed by Dijkstra 1-PDPT, together with related running time TD in seconds and without including preprocessing, the number TrD of transfer arcs of the solution.
3. The value NTr of the solution obtained through enumeration when forbidding the transfer-arcs.
4. The same values VA*(ε), TA*(ε), TrA*(ε), VD(ε), TD(ε), TrD(ε) obtained while taking the eligibility threshold ε as $\frac{n}{2}$.

Experiments were performed on a computer with a 2.3GHz Intel Core i5 processor and 16GB 1333MHz RAM. The implementations were built in C++ 11 by using the Apple Clang compiler. Note that some values are not available due to the absence of solutions.

Results and Comments: We note that the associated running times of both algorithms are all less than two seconds. However, these results must be taken carefully: although it is well known that Dijkstra's algorithm has a time complexity that is always polynomially bounded on the size of the input, for the A* 1-PDPT this is not the case; as a matter of fact, we have found manually an unfeasible instance with 12 vehicles, similar to instances 19 and 20 of Table 1, where the A* 1-PDPT needs to explore many states and takes around 9 seconds to assert infeasibility.

More extensive computational studies should confirm the exponential nature of the A* 1-PDPT complexity (here, we mean complexity in the worst case), and the polynomial complexities that can be achieved by limiting the number of transfer-arcs of the constructed paths.

Regarding the solutions quality we can verify that, most of the time Dijkstra's algorithm has found an optimal or close to optimal solution. However, for instances 11 and 19 of Table 2, and instances 9, 11, and 13 of Table 3 it was not able to find any feasible solution.

Table 2: Behavior of A* 1-PDPT and Dijkstra 1-PDPT without any eligibility restriction.

Id	VA*	TA*	TrA*	VD	TD	TrD	NTr
1	52	0.0022	0	52	0.0006	0	52
2	43	0.0106	0	43	0.0006	0	43
3	24	0.0036	0	24	0.0008	0	24
4	21	0.0034	0	21	0.0008	0	21
5	60	0.0157	1	60	0.0007	1	66
6	52	0.0397	1	52	0.0007	1	55
7	50	0.1937	0	50	0.0011	0	50
8	41	0.2045	0	41	0.0011	0	41
9	83	0.0488	1	83	0.0013	1	130
10	64	0.1753	0	64	0.0013	0	64
11	93	0.0195	0		0.0019		93
12	81	0.1108	0	81	0.0012	0	81
13	92	0.0028	1	92	0.0024	1	110
14	69	1.3397	1	69	0.0015	1	74
15	68	0.1309	0	68	0.0019	0	68
16	57	0.3576	0	57	0.0019	0	57
17	78	0.1754	0	78	0.0016	0	78
18	70	0.4568	0	72	0.0017	0	70
19	99	0.0340	2		0.0017		
20	72	0.3319	1	72	0.0018	1	107

5 CONCLUSION

We have been analyzing transfer mechanisms for PDP problems, while proposing exact and heuristic algorithms for the 1-Request Insertion PDPT case related to the insertion process. We could check that, while allowing transfers is often useful, the number of transfers is usually small (see the 4th and the 7th columns of Table 2 and Table 3), and that it is important to a priori filter transfer moves which are allowed. As open issues, remain the case of strong synchronization (both vehicles which perform a transfer must meet at the time when the transfer occurs), and the problem of ensuring the robustness of a schedule in case some vehicle involved in an exchange is late.

Table 3: Behavior of A* 1-PDPT and Dijkstra 1-PDPT with eligibility threshold $\varepsilon = \frac{n}{2}$.

Id	VA*(ε)	TA*(ε)	TrA*(ε)	VD(ε)	TD(ε)	TrD(ε)	NTr
1	52	0.0010	0	52	0.0006	0	52
2	43	0.0057	0	43	0.0008	0	43
3	24	0.0019	0	24	0.0007	0	24
4	21	0.0029	0	21	0.0008	0	21
5	62	0.0024	1	62	0.0006	1	66
6	52	0.0108	1	52	0.0010	1	55
7	50	0.1508	0	50	0.0011	0	50
8	41	0.1841	0	41	0.0011	0	41
9	100	0.0130	2		0.0013		130
10	64	0.0927	0	64	0.0013	0	64
11	93	0.0045	0		0.0016		93
12	81	0.0433	0	81	0.0012	0	81
13	98	0.0135	1		0.0021		110
14	69	0.6883	1	69	0.0015	1	74
15	68	0.0321	0	68	0.0019	0	68
16	57	0.1828	0	57	0.0018	0	57
17	78	0.1052	0	78	0.0026	0	78
18	70	0.2830	0	72	0.0016	0	70
19		0.0291			0.0025		
20	87	0.0061	4	87	0.0025	4	107

ACKNOWLEDGEMENTS

This work was supported by the French National Research Agency in the Framework of the LabEx IMobS3 and the Région Auvergne-Rhone-Alpes.

REFERENCES

Berbeglia, G., Cordeau, J.-F., Gribkovskaia, I., and Laporte, G. (2007). Static pickup and delivery problems: A classification scheme and survey. *TOP: An Official Journal of the Spanish Society of Statistics and Operations Research*, 15:1–31.

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

Bouros, P., Dalamagas, T., Sacharidis, D., and Sellis, T. (2011). Dynamic pickup and delivery with transfers. *Proceedings of the 12th International Symposium on Spatial and Temporal Databases.*

Bsaybes, S., Quilliot, A., and Wagler, A. K. (2019). Fleet management for autonomous vehicles using flows in time-expanded networks. *TOP*.

Contardo, C., Cortés, C., and Matamala, M. (2010). The pickup and delivery problem with transfers: Formulation and a branch-and-cut solution method. *European Journal of Operational Research*, 200:711–724.

Dechter, R. and Pearl, J. (1985). Generalized best-first search strategies and the optimality of a*. *J. ACM*, 32:505–536.

Gouveia, L., Leitner, M., and Ruthmair, M. (2019). Layered graph approaches for combinatorial optimization problems. *Computers & Operations Research*, 102:22–38.

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

Ho, S. C., Kuo, Y.-H., Leung, J. M., Petering, M., Szeto, W., and Tou, T. W. (2018). A survey of dial-a-ride problems: Literature review and recent developments. *Transportation Research Part B: Methodological*, 111:395–421.

Laporte, G. and Mitrović-Minić, S. (2006). The pickup and delivery problem with time windows and transshipment. *INFOR: Information Systems and Operational Research*, 44(3):217–227.

Lehuédé, F., Masson, R., and Péton, O. (2013). Efficient feasibility testing for request insertion in the pickup and delivery problem with transfers. *Operations Research Letters*, 41(3):211–215.

Nilsson, N. J. (1980). *Principles of Artificial Intelligence*. Morgan Kaufmann, San Francisco (CA).