Dynamic Pickup and Delivery with Transfers

Panagiotis Bouros[1], Dimitris Sacharidis[2], Theodore Dalamagas[2], and Timos Sellis[3]

[1] National Technical University of Athens, Greece
pbour@dblab.ece.ntua.gr
[2] Institute for the Management of Information Systems, R.C. "Athena", Greece
{dsachar,dalamag}@imis.athena-innovation.gr
[3] National Technical University of Athens, Greece, and
Institute for the Management of Information Systems, R.C. "Athena", Greece
timos@imis.athena-innovation.gr

Abstract. In the dynamic Pickup and Delivery Problem with Transfers (dPDPT), a set of transportation requests that arrive at arbitrary times must be assigned to a fleet of vehicles. We use two cost metrics that capture both the company's and the customer's viewpoints regarding the quality of an assignment. In most related problems, the rule of thumb is to apply a two-phase local search algorithm to heuristically determine a good requests-to-vehicles assignment. This work proposes a novel solution based on a graph-based formulation of the problem that treats each request independently. Briefly, in this conceptual graph, the goal is to find a shortest path from a node representing the pickup location to that of the delivery location. However, we show that efficient Bellman-Ford or Dijkstra-like algorithms cannot be applied. Still, our method is able to find dPDPT solutions significantly faster than a conventional two-phase local search algorithm, while the cost of the solution is only marginally increased.

1 Introduction

The family of pickup and delivery problems covers a broad range of optimization problems that appear in various logistics and transportation scenarios. Broadly speaking, these problems look for an assignment of a set of transportation requests to a fleet of vehicles in a way that satisfies a number of constraints and at the same time minimizes a specific cost function. In this context, a transportation request is defined as *picking up* an object (e.g., package, person, etc.) from one location and *delivering* it to another; hence the name.

In its simplest form, the *Pickup and Delivery Problem* (PDP) only imposes two constraints. The first, termed *precedence*, naturally states that pickup should occur before delivery for each transportation request. The second, termed *pairing*, states that both the pickup and the delivery of each transportation request should be performed by the same vehicle. The *Pickup and Delivery Problem with Transfers* (PDPT) [11, 25] is a PDP variant that eliminates the pairing constraint. In PDPT, objects can be transferred between vehicles. *Transfers* can

occur in predetermined locations, e.g., depots, or in arbitrary locations as long as the involved vehicles are in close proximity to each other at some time. We refer to the latter case as *transfer with detours*, since the vehicles may have to deviate from their routes.

Almost every pickup and delivery problem comes in two flavors. In *static*, all requests are known in advance and the goal is to come up with the best vehicle routes from scratch. On the other hand, in *dynamic*, a set of vehicle routes, termed the *static plan*, has already been established. Then, additional requests arrive ad hoc, i.e., at arbitrary times, and the plan must be modified to satisfy them. While algorithms for static problems can also solve the dynamic counterpart, they are rarely used as they take a lot of time to execute. Instead, common practice is to apply two-phase local search algorithms. In the first phase, a quick solution is obtained by assigning each standing request to the vehicle that results in the smallest cost increase. In the second phase, the obtained solution is improved by reassigning requests.

This paper proposes an algorithm for the *dynamic Pickup and Delivery Problem with Transfers* (dPDPT). Although works for the dynamic PDP can be extended to consider transfers between vehicles, to the best of our knowledge, this is the first work targeting dPDPT. Our solution processes requests independently, and does not follow the two-phase paradigm. Satisfying a request is treated as a shortest path problem in a conceptual graph. Intuitively, the object must travel from the pickup to the delivery location following the vehicles' routes and schedules.

The primary goal in pickup and delivery problems is to minimize the total operational cost required to satisfy the requests, i.e., the company's expenses. Under our dPDPT formulation, a satisfied request is represented as a path p. We define its *operational cost* O_p as the additional cost (total delay), with respect to the static plan, incurred by the vehicles in order to accommodate the solution p. In addition, we consider the promptness of satisfying the request. We define the *customer cost* C_p of a path p as the delivery time of the object. These costs are in general conflicting, as they represent two distinct views. For example, the path with the earliest delivery time may require significant changes in the schedule of the vehicles and cause large delays on the static plan. In contrast, the path with the smallest operational cost could result in late delivery. Our algorithm can operate under any monotonic combination of the two costs. However, in this work, we consider operational cost as more important; customer cost is used to solve ties.

Finding the shortest path (according to the two costs) in the conceptual graph is not straightforward. The reason is that the weights of the edges depend on both the operational and customer cost of the path that led to this edge. In fact, an important contribution of this paper is that we show, contrary to other time-dependent networks, that the conceptual graph does not exhibit the *principle of optimality*, which is necessary to apply efficient Bellman-Ford or Dijkstra-like algorithms. Hence, one has to enumerate all possible paths. However, despite

this fact, extensive experimental results show that our method is significantly faster than a two-phase local search algorithm adapted for dPDPT.

The remainder of this paper is organized as follows. Section 2 reviews related work and Section 3 formally introduces dPDPT. Section 4 presents our graph-based formulation and algorithm. Then, Section 5 presents an extensive experimental evaluation and Section 6 concludes this work.

2 Related work

This work is related to pickup and delivery, and shortest path problems.

Pickup and Delivery Problems.

In the *Pickup and Delivery Problem* (PDP) objects must be transported by a fleet of vehicles from a pickup to a delivery location with the minimum cost, under two constrains: (1) pickup occurs before delivery (*precedence*), and (2) pickup and delivery of an object is performed by the same vehicle (pairing). PDP is NP-hard since it generalizes the well-known Traveling Salesman Problem (TSP). Exact solutions employ column generation [14, 32, 34], branch-and-cut [9, 30] and branch-and-cut-and-price [29] methodologies. On the other hand, the heuristics for the approximation methods take advantage of local search [2, 22, 25, 31].

Other PDP variations introduce additional constraints. For instance, in the *Pickup and Delivery Problem with Time Windows* (PDPTW), pickups and deliveries are accompanied with a time window that mandates when the action can take place. In the *Capacitated Pickup and Delivery Problem* (CPDP), the amount of objects a vehicle is permitted to carry at any time is bounded by a given capacity. In the *Pickup and Delivery Problem with Transfers* (DPDT), studied in this paper, the pairing constraint is lifted. [11] proposes a branch-and-cut strategy for DPDT. [25] introduces the Pickup and Delivery Problem with Time Windows and Transfers and employs a local search optimization approach. In all the above problems, the transportation requests are known in advance, hence they are characterized as *static*. A formal definition of static PDP and its variants can be found in [3, 10, **?**].

Almost all PDP variants also have a *dynamic* counterpart. In this case, a set of vehicle routes, termed the *static plan*, has already been established, and additional requests arrive ad hoc, i.e., at arbitrary times. Thus, the plan must be modified to satisfy them. A survey on dynamic PDP can be found in [4]. Typically, two-phase local search methods are applied for the dynamic problems. The first phase applies an insertion heuristic [28], whereas the second employs tabu search [15, 23, 24]. To the best of our knowledge our work is the first to address the *dynamic Pickup and Delivery Problem with Transfers* (dPDPT).

Shortest Path Problems.

Bellman-Ford and Dijkstra are the most well-known algorithms for finding the *shortest path* between two nodes in a graph. The ALT algorithms [**?**,16, 27] perform a bidirectional A* search and exploit a lower bound of the distance between two nodes to direct the search. To compute this bound they construct an

embedding on the graph. There exist a number of materialization techniques [1, 19, 20] or encoding/labeling schemes [6, 7] that can be used to efficiently compute the shortest path. Both the ALT algorithms and the materialization and encoding methods are mostly suitable for graphs that are not frequently updated, since they require expensive precomputation.

In *multi-criteria shortest path problems* the quality of a path is measured by multiple metrics, and the goal is to find all paths for which no better exists. Algorithms are categorized into three classes. The methods of the first class (e.g., [5]) apply a user preference function to reduce the original multi-criteria problem to a conventional shortest path problem. The second class contains the interactive methods (e.g., [17]) that interact with a decision maker to come up with the answer path. Finally, the third class includes label-setting and label-correcting methods (e.g., [18, 21, 33]). These methods construct a label for every path followed to reach a graph node. Then, at each iteration, they select the path with the minimum cost, defined as the combination of the given criteria, and expand the search extending this path.

In time-dependent shortest path problems the cost of traveling from node n_i to n_j in a graph (e.g., the road network of a city) depends on the departure time t from n_i. [8] is the first attempt to solve this problem using a Bellman-Ford based solution. However, as discussed in [13], Dijkstra can also be applied for this problem, as long as the earliest possible arrival time at a node is considered. In the context of transportation systems, the *FIFO* (a.k.a. *non-overtaking*) property of a road network is considered as a necessity in order to achieve an acceptable level of complexity. According to this property delaying the departure from a graph node n_i to reach n_j cannot result in arriving earlier at n_j. However, even when the FIFO property does not hold it is possible to provide an efficient solution [12, 26] by properly adjusting the weights in graph edges [12].

3 Problem Formulation

Section 3.1 provides basic definitions and introduces the dynamic Pickup and Delivery Problem with Transfers. Section 3.2 details the actions allowed for satisfying a request and their costs.

3.1 Definitions

Assume that a company has already scheduled its fleet of vehicles to service a number of requests. We refer to this schedule as the *static plan*, since we assume that it is given as input. The static plan consists of a set of vehicles following some routes; we overload notation r_a to refer to both a vehicle and its route. The *route* of a vehicle r_a is a sequence of distinct spatial locations, where each location n_i is associated with an *arrival time* A_i^a and a *departure time* D_i^a. Note that the requirement for distinct locations within a route is introduced to simplify notation and avoid ambiguity when referring to a particular location. Besides, if a vehicle visits a location multiple times, its route can always be

represented as a set of distinct-locations routes. The difference $D_i^a - A_i^a$ is a non-negative number; it may be zero indicating that vehicle r_a just passes by n_i, or a positive number corresponding to some service at n_i, e.g., pickup, delivery, mandatory stop, etc. For two consecutive locations n_i and n_j on r_a, the difference $A_j^a - D_i^a \geq 0$ corresponds to the travel time from n_i to n_j.

An ad-hoc dPDPT *request* is a pair of locations n_s and n_e, signifying that a package must be picked up at n_s and be delivered at n_e. In order to complete a request, it is necessary to perform a series of modifications to the static plan. There are five types of modifications allowed, termed *actions*: pickup, delivery, transport, transfer without detours, and transfer with detours. Each action, described in detail later, results in the package changing location and/or vehicle. A sequence of actions is called a *path*. If the initial and final location of the package in a path are n_s and n_e, respectively, the path is called a *solution* to the request.

There are two costs associated with an action. The *operational cost* measures the time spent by vehicles in order to perform the action, i.e., the delay with respect to the static plan. The *customer cost* represents the time when the action is completed. Furthermore, the operational cost O_p of a path p is defined as the sum of operational costs for each action in the path, and the customer cost C_p is equal to the customer cost of the final action in p. Therefore, for a solution p, O_p signifies the company's cost in accommodating the request, while C_p determines the delivery time of the package according to p.

Any monotonic combination (e.g., weighted sum, product, min, max, average etc.) of the two costs could be a meaningful measure for assessing the quality of a solution. In the remainder of this paper, we assume that the operational cost is more important, and that the customer cost is of secondary importance. Therefore, a path p is preferred over q, if $O_p < O_q$, or $O_p = O_q \wedge C_p < C_q$. Equivalently, we may define the *combined cost* of a path p as:

$$cost(p) = M \cdot O_p + C_p \ , \tag{1}$$

where M is a sufficiently large number (greater than the largest possible customer cost divided by the smaller possible operational cost) whose sole purpose is to assign greater importance to the operational cost. Based on this definition, the *optimal solution* is the one that has the lowest combined cost, i.e., the minimum customer cost among those that have the least operational cost. The *dynamic Pickup and Delivery with Transfers* (dPDPT) problem is to find the optimal solution path.

3.2 Actions

It is important to note that, throughout this paper, we follow the convention that an action is completed by vehicle r_a at a location n_i just before r_a departs from n_i. Since r_a can have multiple tasks to perform at n_i according to the static plan, this convention intuitively means that we make no assumptions about the order in which a vehicle performs its tasks. In any case, the action will have concluded by the time r_a is ready to depart from n_i.

Consider a path p with operational and customer costs O_p and C_p, respectively. Further, assume that the last action in p results in the package being onboard vehicle r_a at location n_i. Let p' denote the path resulting upon performing an additional action E on p. In the following, we detail each possible action E, and the costs of the resulting path p', denoted as $O_{p'}$, $C_{p'}$, which may depend on the current path costs O_p, C_p.

Pickup

The pickup action involves a single vehicle, r_a, and appears once as the first action in a solution path. Hence, initially the package is at the pickup location n_s of the request, p is empty, and $O_p = C_p = 0$.

We distinguish two cases for this action. First, assume that n_s is included in the vehicle's route, and let A_s^a, D_s^a denote the arrival and departure times of r_a at n_s according to the static plan. In this case, the pickup action is denoted as E_s^a. No modification in r_a's route is necessary, and thus there is zero additional operational cost for executing E_s^a. The customer cost for the resulting path p' becomes equal to the scheduled (according to the static plan) departure time D_s^a from n_s; without loss of generality, we make the assumption that the request arrives at time 0. Therefore,

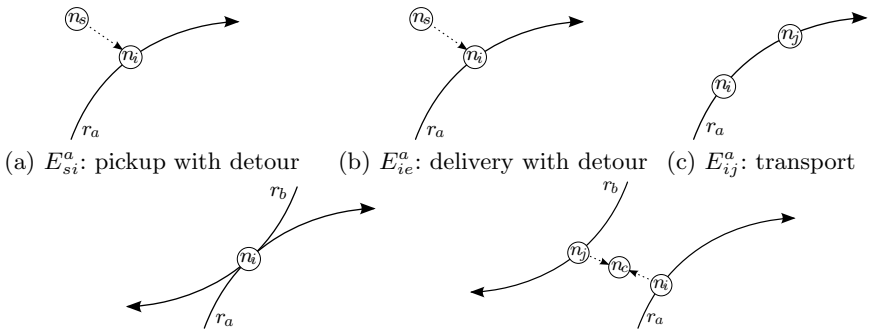
$$\left.\begin{array}{l} O_{p'} = 0 \\ C_{p'} = D_s^a \end{array}\right\} \text{ for } p' = E_s^a. \tag{2a}$$

In the second case, the pickup location n_s is not in the r_a route. Let n_i be a location in the r_a route that is *sufficiently close* to n_s; then, r_a must take a detour from n_i to n_s. A location is sufficiently close to n_s if the detour is short, i.e., its duration, denoted as T_{si}^a, is below some threshold (a system parameter). Hence, it is possible that a sufficiently close location does not exists for route r_a; clearly, if no such location exists for any route, then the request is unsatisfiable. When such a n_i exists, the pickup action is denoted as E_{si}^a. Figure 1(a) shows a pickup action with detour. The solid line in the figure denotes the vehicle route r_a and the dashed line denotes the detour performed by r_a from n_i to n_s to pickup the package. The operational cost of a pickup action with detour is equal to the delay T_{si}^a due to the detour. The customer cost of p' is the scheduled departure time from n_i incremented by the delay. Therefore,

$$\left.\begin{array}{l} O_{p'} = T_{si}^a \\ C_{p'} = D_i^a + T_{si}^a \end{array}\right\} \text{ for } p' = E_{si}^a. \tag{2b}$$

Delivery

The delivery action involves a single vehicle, r_a, and appears once as the last action in a solution path. Similar to pickup, two cases exist for this action. In the first case, n_e appears in the route r_a, and delivery is denoted as E_e^a. The costs for path p' are shown in Equation 3a. In the second case, a detour of length T_{ie}^a at location n_i is required, and delivery is denoted as E_{ie}^a. Figure 1(b) presents an E_{ie}^a delivery action with detour. The costs for p' are shown in Equation 3b,



(a) E^a_{si}: pickup with detour (b) E^a_{ie}: delivery with detour (c) E^a_{ij}: transport

(d) E^{ab}_i: transfer without detour (e) E^{ab}_{ij}: transfer with detour

Fig. 1. Actions allowed for satisfying a dPDPT request.

where we make the assumption that it takes $T^a_{ie}/2$ time to travel from n_i to n_e.

$$\left.\begin{array}{l} O_{p'} = O_p \\ C_{p'} = C_p \end{array}\right\} \text{ for } p' = pE^a_e. \tag{3a}$$

$$\left.\begin{array}{l} O_{p'} = O_p + T^a_{ie} \\ C_{p'} = C_p + T^a_{ie}/2 \end{array}\right\} \text{ for } p' = pE^a_{ie}. \tag{3b}$$

Transport

The transport action involves a single vehicle, r_a, and corresponds to the carrying of the package by a vehicle between two successive locations on its route. Figure 1(c) illustrates such a transportation action from location n_i to n_j onboard vehicle r_a. As assumed, path p results in the package being onboard r_a at location n_i. The transport action, denoted as E^a_{ij}, has zero operational cost, as the vehicle is scheduled to move from n_i to n_j anyway. The customer cost is incremented by the time required by vehicle r_a to travel from n_i to n_j and finish its tasks at n_j. Therefore,

$$\left.\begin{array}{l} O_{p'} = O_p \\ C_{p'} = C_p + D^a_j - D^a_i \end{array}\right\} \text{ for } p' = pE^a_{ij}. \tag{4}$$

Transfer without detours

The transfer without detours action, denoted as E^{ab}_i, involves two vehicles, r_a and r_b, and corresponds to the transfer of the package from r_a to r_b at a common location n_i, e.g., a depot, drop-off/pickup point, etc. For example, Figure 1(d) shows such a transfer action via the common location n_i. Let A^i_b, D^i_b be the arrival and departure times of vehicle r_b at location n_i. We distinguish three cases.

In the first, the last action in path p concludes after vehicle r_b arrives and before it departs from n_i, i.e., $A^b_i \leq C_p \leq D^b_i$. Since there is no delay in the schedule of vehicles, the action's operational cost is zero, while the customer

cost of the resulting path p' becomes equal to the scheduled departure time of r_b from n_i. Therefore,

$$\left.\begin{array}{l} O_{p'} = O_p \\ C_{p'} = D_i^b \end{array}\right\} \text{ for } p' = pE_i^{ab}, \text{ if } A_i^b \leq C_p \leq D_i^b. \tag{5a}$$

In the second case, the last action in path p concludes before vehicle r_b arrives at n_i, i.e., $C_p < A_i^b$. For the transfer to proceed, vehicle r_a must wait at n_i until r_b arrives. The operational cost is incremented by the delay, which is equal to $A_i^b - C_p$. On the other hand, the customer cost becomes equal to the scheduled departure time of r_b from n_i. Therefore,

$$\left.\begin{array}{l} O_{p'} = O_p + A_i^b - C_p \\ C_{p'} = D_i^b \end{array}\right\} \text{ for } p' = pE_i^{ab}, \text{ if } C_p < A_i^b. \tag{5b}$$

In the third case, the last action in p concludes after vehicle r_b is scheduled to depart from n_i, i.e., $C_p > D_i^b$. This implies that r_b must wait at n_i until the package is ready for transfer. The delay is equal to $C_p - D_i^b$, which contributes to the operational cost. The customer cost becomes equal to the delayed departure of r_b from n_i, which coincides with C_p. Therefore,

$$\left.\begin{array}{l} O_{p'} = O_p + C_p - D_i^b \\ C_{p'} = C_p \end{array}\right\} \text{ for } p' = pE_i^{ab}, \text{ if } C_p > D_i^b. \tag{5c}$$

Transfer with detours

Consider distinct locations n_i on r_a and n_j on r_b. Assume that short detours from n_i and n_j are possible, i.e., the detour durations are below some threshold, and that they have a common rendezvous point. The transfer with detours action, denoted as E_{ij}^{ab}, involves the two vehicles, r_a and r_b, and corresponds to the transportation of the package on vehicle r_a via the n_i detour to the rendezvous location, its transfer to vehicle r_b, which has taken the n_j detour, and finally its transportation to n_j. Figure 1(e) illustrates a transfer action between vehicles r_a and r_b via a detour to their common rendezvous point n_c. Notice the difference with Figure 1(d) where the transfer action occurs without a detour. To keep the notation simple, we make the following assumptions: (1) the n_i detour travel time of r_a is equal to that of the n_j detour of r_b, denoted as T_{ij}^{ab}; and (2) it takes $T_{ij}^{ab}/2$ time for both r_a and r_b to reach the rendezvous location.

Similar to transferring without detours, we distinguish three cases. In the first, the package is available for transfer at the rendezvous location, at time $C_p + T_{ij}^{ab}/2$, after the earliest possible and before the latest possible arrival of r_b, i.e., $A_j^b + T_{ij}^{ab}/2 \leq C_p + T_{ij}^{ab}/2 \leq D_j^b + T_{ij}^{ab}/2$. Both vehicles incur a delay in their schedule by T_{ij}^{ab}. Therefore,

$$\left.\begin{array}{l} O_{p'} = O_p + 2 \cdot T_{ij}^{ab} \\ C_{p'} = D_j^b + T_{ij}^{ab} \end{array}\right\} \text{ for } p' = pE_{ij}^{ab}, \text{ if } A_j^b \leq C_p \leq D_j^b. \tag{6a}$$

In the second case, the package is available for transfer before the earliest possible arrival of r_b at the rendezvous location, i.e., $C_p + T_{ij}^{ab}/2 < A_j^b + T_{ij}^{ab}/2$.

Vehicle r_a must wait for $A_j^b - C_p$ time. Therefore,

$$\left.\begin{array}{l} O_{p'} = O_p + A_j^b - C_p + 2 \cdot T_{ij}^{ab} \\ C_{p'} = D_j^b + T_{ij}^{ab} \end{array}\right\} \text{ for } p' = pE_{ij}^{ab}, \text{ if } C_p < A_j^b. \tag{6b}$$

Finally, in the third case, the package is available for transfer after the latest possible arrival of r_b at the rendezvous location, i.e., $C_p + T_{ij}^{ab}/2 > D_j^b + T_{ij}^{ab}/2$. Vehicle r_b must wait for $C_p - D_j^b$ time. Therefore,

$$\left.\begin{array}{l} O_{p'} = O_p + C_p - D_j^b + 2 \cdot T_{ij}^{ab} \\ C_{p'} = C_p + T_{ij}^{ab} \end{array}\right\} \text{ for } p' = pE_{ij}^{ab}, \text{ if } C_p > D_j^b. \tag{6c}$$

4 Solving dynamic Pickup and Delivery with Transfers

Section 4.1 models dynamic Pickup and Delivery with Transfers as a dynamic shortest path graph problem. Section 4.2 introduces the SP algorithm that identifies the solution to a dPDPT request.

4.1 The Dynamic Plan Graph

We construct a weighted directed graph, termed *dynamic plan graph*, in a way that a sequence of actions corresponds to a simple path on this graph. A vertex of the graph corresponds to a spatial location. In particular, a vertex V_i^a represents the spatial location n_i of route r_a. Additionally, there exist two special vertices, V_s and V_e, which represent the request's pickup and delivery, respectively, locations. Therefore, five types of edges exist:

(1) A *pickup edge* E_{si}^a connects V_s to V_i^a, and represents a pickup action by vehicle r_a with a detour at n_i. Edge E_{ss}^a from V_s to V_s^a (two distinct vertices that correspond to the same spatial location n_s) represents the case of pickup with no detour.
(2) A *delivery edge* E_{ie}^a connects V_i^a to V_e, and represents a delivery action by vehicle r_a with a detour at n_i. Edge E_{ee}^a from V_e to V_e^a represents the case of pickup with no detour.
(3) A *transport* edge E_{ij}^a connects V_i^a to V_j^a, and represents a transport action by r_a from n_i to its following location n_j on the route.
(4) A *transfer without detours* edge E_i^{ab} connects V_i^a to V_i^b, and represents a transfer from r_a to r_b at common location n_i.
(5) A *transfer with detours* edge E_{ij}^{ab} connects V_i^a to V_j^b, and represents a transfer from r_a to r_b at a rendezvous location via detours at n_i and n_j.

Based on the above definitions, a simple path on the graph is a sequence of distinct vertices that translates into a sequence of actions. Further, a solution for the request is a path that starts from V_s and ends in V_e.

The final issue that remains is to define the weights $\mathcal{W}$ of the edges. We assign edge E a pair of weights $w(E) = \langle w_O(E), w_C(E) \rangle$, so that $w_O(E)$ (resp.

Pickup	Delivery	Transport
$w(E^a_{si}) = \langle T^a_{si}, D^a_i + T^a_{si} \rangle$ (7)	$w(E^a_{ie}) = \langle T^a_{ie}, T^a_{ie}/2 \rangle$ (8)	$w(E^a_{ij}) = \langle 0, D^a_j - D^a_i \rangle$ (9)

Transfer

$$w(E^{ab}_{ij}) = \begin{cases} \langle 2 \cdot T^{ab}_{ij}, D^b_j - C_p + T^{ab}_{ij} \rangle, & \text{if } A^b_j \leq C_p \leq D^b_j \\ \langle A^b_j - C_p + 2 \cdot T^{ab}_{ij}, D^b_j - C_p + T^{ab}_{ij} \rangle, & \text{if } C_p < A^b_j \\ \langle C_p - D^b_j + 2 \cdot T^{ab}_{ij}, T^{ab}_{ij} \rangle, & \text{if } C_p > D^b_j. \end{cases} \quad (10)$$

Table 1. Edge weights

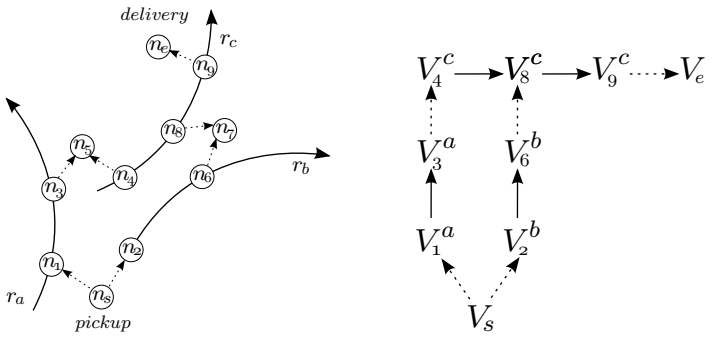
$w_C(E)$) corresponds to the operational (resp. customer) cost of performing the action associated with the edge E. Recall from Section 3.2 that the costs of the last action in a sequence of actions depends on the total costs incurred by all previous actions. Consequently, the weights of an edge E from V to V' are *dynamic*, since they depend on the costs of the path p that lead to V. Assuming O_p and C_p are the costs of p, and $O_{p'}$ and $C_{p'}$ those of path $p' = pE$ upon executing E, we have that $w(E) = \langle O_{p'} - O_p, C_{p'} - C_p \rangle$. Table 1 summarizes the formulas for the weights of all edge types; note that the weights for actions with no detours are obtained by setting the corresponding T value to zero. In the formulas, A^b_j, D^a_i, D^a_j and D^b_j have fixed values determined by the static plan. On the other hand, C_p depends on the path p that leads to V^a_i.

Clearly, a path from V_s to V_e that has the lowest combined cost according to Equation 1 is an optimal solution.

Proposition 1. *Let R be a collection of vehicles routes and (n_s, n_e) be a dPDPT request over R. The solution to the request is the shortest path from vertex V_s to V_e on the dynamic plan graph G_R with respect to cost() of Equation 1.*

Example 1. Figure 2(a) pictures a collection of vehicle routes $R = \{r_a(n_1, n_3), r_b(n_2, n_6), r_c(n_4, n_8, n_9)\}$, and the pickup n_s and the delivery location n_e of a dPDPT request. Locations n_1 on route r_a and n_2 on r_b are sufficiently close to location n_s and thus, pickup actions E^a_{s1} and E^b_{s2} are possible. Similar, the E^c_{9e} delivery action is possible at location n_9 on route r_c. Finally, we also identify two transfer actions, E^{ac}_{34} and E^{bc}_{68}, as locations n_3, n_4 and n_6, n_8 have common rendezvous points n_5 and n_7, respectively.

To satisfy the dPDPT request (n_s, n_e) we define the dynamic plan graph G_R in Figure 2(b) containing vertices $V_s, V^a_1, V^b_2, \ldots, V_e$. Notice that the graph does not include any vertices for the rendezvous points n_5 and n_7. Dynamic plan graph G_R contains two paths from V_s to V_e which means that there two different ways to satisfy the dPDPT request: $p_1(V_s, V^a_1, V^a_3, V^c_4, V^c_8, V^c_9, V_e)$ and $p_2(V_s, V^b_2, V^b_6, V^c_8, V^c_9, V_e)$. Next, assume, for simplicity, that the detour cost is equal to T for all possible actions. Further, consider paths $p'_1(V_s, V^a_1, V^a_3)$ and



(a) Vehicle routes R (b) Dynamic plan graph G_R

Fig. 2. A collection of vehicles routes R, a dPDPT request (n_s, n_e) over R, and the dynamic plan graph G_R. The solid lines denote the vehicle routes/transport edges while the dashed lines denote the pickup, delivery and transfer with detour actions/edges.

$p'_2(V_s, V_2^b, V_6^b)$, i.e., just before the transfer of the package takes place, and assume that $A_4^c < C_{p'_1} < D_4^c$ and $C_{p'_2} > D_8^c$ hold. Note, that the operational cost of the two paths is exactly the same, i.e., $O_{p'_1} = O_{p'_2} = T$, coming from the pickup of the package at n_s. Now, according to Equation 10 and after the transfers E_{34}^{ac} and E_{68}^{bc} take place, we get path $p''_1 = p'_1 E_{68}^{bc}$ and $p''_2 = p'_2 E_{68}^{bc}$ with $O_{p''_1} = 3 \cdot T < O_{p''_2} = 3 \cdot T + C_{p'_2} - D_8^c$, and therefore, $cost(p''_1) < cost(p''_2)$. Finally, since no other transfer incurs in order to delivery the package, this holds also for paths p_1 and p_2, i.e., $cost(p_1) < cost(p_2)$, and thus, the solution to the dPDPT request (n_s, n_e) is path $p_1(V_s, V_1^a, V_3^a, V_4^c, V_8^c, V_9^c, V_e)$ with $O_{p_1} = 4 \cdot T$ and $C_{p_1} = D_9^c + \frac{3 \cdot T}{2}$.

4.2 The SP Algorithm

According to Proposition 1, the next step is to devise an algorithm that computes the two-criterion shortest path w.r.t to $cost()$ on the dynamic plan graph. Unfortunately, the dynamic weights of the edges in the graph violate the *subpath optimality*; that is, the lowest combined cost path from V_s to V_e that passes through some vertex V may not contain the lowest combined cost path from V_s to V. The following theorem supports this claim.

Theorem 1. *The dynamic plan graph does not have subpath optimality for any monotonic combination of the operational and customer costs.*

Proof. Let p, q be two paths from V_s to V_i^a, with costs O_p, C_p and O_q, C_q, respectively, such that $O_p < O_q$ and $C_p < C_q$, which implies that for any monotonic combination of the operational and customer costs, p has lower combined cost than q. Let p' and q' be the paths resulting after traversing a transfer without detours edge E_i^{ab}.

Assume that $C_p < C_q < A_i^b$, so that the second case of Equation 10 applies for the weight $w(E_i^{ab})$ (setting $T_{ij}^{ab} = 0$). Then, $O_{p'} = O_p + A_i^b - C_p$, $O_{q'} = O_q + A_i^b - C_q$, and $C_{p'} = C_{q'} = D_i^b$. Assuming that $O_p - C_p > O_q - C_q$, we obtain that $O_{q'} < O_{p'}$. Therefore, for any monotonic combination that considers both costs, q''s combined cost is lower that that of p''s. □

As a result, efficient algorithms based on the subpath optimality, e.g., Dijkstra and Bellman-Ford cannot be employed to compute the shortest path on the dynamic plan graph. In contrast, an exhaustive enumeration of all paths from V_s to V_e is necessary, and for this purpose, we introduce a label-setting algorithm called SP. Note that, in the sequel, we only discuss the case when actions occur with detours as it is more general.

The SP algorithm has the following key features. First, similar to all algorithms for multi-criteria shortest path, it may visit a vertex V_i^a more than once following multiple paths from the initial vertex V_s. For each of these paths p, the algorithm defines a *label* in the form of $\langle V_i^a, p, O_p, C_p \rangle$, where O_p is the operational and C_p the customer cost of path p as introduced in Section 4.1. Second, at each iteration, SP selects the label $\langle V_i^a, p, O_p, C_p \rangle$ of the most "promising" path p, in other words, the path with the lowest $cost(p)$, and expands the search considering the outgoing edges from V_i^a on the dynamic plan graph G_R. If vertex V_i^a has an outgoing delivery edge E_{ie}^a, SP identifies a path from initial vertex V_s to final V_e called *candidate* solution. The candidate solution is an *upper bound* to the final solution and it is progressively improved until it becomes equal to the final. The role of a candidate solution is twofold; it triggers the termination condition and prunes the search space. Finally, the algorithm terminates the search when $cost(p)$ of the most "promising" path p is equal to or higher than $cost(p_{cand})$ of the current candidate solution p_{cand} which means that neither p or any other path at future iterations can be better than current p_{cand}.

Figure 3 illustrates the pseudocode of the SP algorithm. SP takes as inputs: a dPDPT request (n_s, n_e) and the dynamic plan graph G_R of a collection of vehicle routes R. It returns the shortest path from V_s to V_e on G_R with respect to $cost()$. The algorithm uses the following data structures: (1) a priority queue $\mathcal{Q}$, (2) a path p_{cand}, and (3) a list $\mathcal{T}$. The priority queue $\mathcal{Q}$ is used to perform the search by storing every label $\langle V_i^a, p, O_p, C_p \rangle$ to be checked, sorted by $cost(p)$ in ascending order. The target list $\mathcal{T}$ contains entries of the form $\langle V_i^a, 0, T_{ie}^a \rangle$, where V_i^a is a vertex of the dynamic plan graph involved in a delivery edge E_{ie}^a, List $\mathcal{T}$ is used to construct or improve the candidate solution p_{cand}.

The execution of the SP algorithm involves two phases: the *initialization* and the *core* phase. In the initialization phase (Lines 1–4), SP first creates the pickup E_{si}^a and delivery edges E_{ie}^a on the dynamic plan graph G_R. For this purpose it identifies each location n_i on every vehicle route r_a that is sufficiently close to pickup location n_s (resp. delivery n_e), i.e., the duration T_{si}^a (resp. T_{ie}^a) of the detour from n_s to n_i (resp. n_i to n_e) is below some threshold (a system parameter). Then, the algorithm initializes the priority queue $\mathcal{Q}$ adding every vertex V_i^a involved in a pickup edge E_{si}^a on G_R and constructs the target list $\mathcal{T}$. In the core phase (Lines 5–17), the algorithm performs the search. It proceeds

Algorithm SP
Input: dPDPT request (n_s, n_e), dynamic plan graph G_R
Output: shortest path from V_s to V_e w.r.t. $cost()$
Parameters:
priority queue $\mathcal{Q}$: the search queue sorted by $cost()$ in ascending order
path p_{cand}: the candidate solution to the dPDPT request
list $\mathcal{T}$: the target list

Method:

1. **construct** pickup edges E^a_{si};
2. **construct** delivery edges E^a_{ie};
3. **for** each pickup edge $E^a_{si}(V_s, V^a_i)$ **do**
   push label $\langle V^a_i, E^a_{si}, T^a_{si}, D^a_i + T^a_{si}\rangle$ in $\mathcal{Q}$;
4. **for** each delivery edge $E^a_{ie}(V^a_i, V_e)$ **do**
   insert $\langle V^a_i, T^a_{ie}, T^a_{ie}/2\rangle$ in $\mathcal{T}$;
5. **while** $\mathcal{Q}$ is not empty **do**
6. **pop** label $\langle V^a_i, p, O_p, C_p\rangle$ from $\mathcal{Q}$;
7. **if** $cost(p) \geq cost(p_{cand})$ **then return** p_{cand};
8. ImproveCandidateSolution$(p_{cand}, \mathcal{T}, \langle V^a_i, p, O_p, C_p\rangle)$;
9. **for** each outgoing transport E^a_{ij} **or** transfer edge E^{ab}_{ij} on G_R **do**
10. **extend** path p and **create** p';
11. **compute** $O_{p'}$ and $C_{p'}$;
12. **if** $cost(p') < cost(p_{cand})$ **then**
    ignore path p';
13. **else**
14. **push** label $\langle V', p', O_{p'}, C_{p'}\rangle$ in $\mathcal{Q}$ **where** V' is the last vertex in p';
15. **end if**
16. **end for**
17. **end while**
18. **return** p_{cand} **if exists**, **otherwise null**;

Fig. 3. The SP algorithm.

iteratively popping, first, the label $\langle V^a_i, p, O_p, C_p\rangle$ from $\mathcal{Q}$ on Line 6. Path p has the lowest $cost(p)$ value compared to all others paths in $\mathcal{Q}$. Next, SP checks the termination condition (Line 7). If the check succeeds, i.e., $cost(p) \geq cost(p_{cand})$, then current candidate p_{cand} is returned as the final solution.

If the termination condition fails, the algorithm first tries to improve candidate solution p_{cand} calling the ImproveCandidateSolution$(p_{cand}, \mathcal{T}, \langle V^a_i, p, O_p, C_p\rangle)$ function on Line 8. The function checks if the target list $\mathcal{T}$ contains an entry $\langle V^a_i, T^a_{ie}, T^a_{ie}/2\rangle$ for the vertex V^a_i of the current label and constructs the path pE^a_{ie} from V_s to V_e. If $cost(pE^a_{ie}) < cost(p_{cand})$ then a new improved candidate solution is identified and thus, $p_{cand} = pE^a_{ie}$. Finally, SP expands the search considering all outgoing transport and transfer edges from V^a_i on G_R (Lines 9–17). Specifically, the path p of the current label is extended to $p' = pE^a_{ij}$ (transport edge) or to $p' = pE^{ab}_{ij}$ (transfer edge), and the operational $O_{p'}$ and the customer cost $C_{p'}$ of the new path p' are computed according to Equations 9 and 10. Then, on Line 12, the algorithm determines whether p' is a "promising" path and thus, it must be extended at a future iteration, or it must be discarded. The algorithm discards path p' if $cost(p') \geq cost(p_{cand})$ which means that p' cannot produce a better solution than current p_{cand}. Otherwise, p' is a "promising" path, and SP inserts label $\langle V', p', O_{p'}, C_{p'}\rangle$ in $\mathcal{Q}$ where V' is the last vertex in path p'.

Example 2. We illustrate the SP algorithm using Figure 2. To carry out the search we make the following assumptions, similar to Example 1. The detour cost is equal to T for all edges. For the paths $p'_1(V_s, V_1^a, V_3^a)$ and $p'_2(V_s, V_2^b, V_6^b)$, i.e., just before the transfer of the package takes place, $A_4^c < C_{p'_1} < D_4^c$ and $C_{p'_2} > D_8^c$ hold. Finally, we also assume that $D_1^a < D_3^a < D_2^b < D_6^b$.

First, SP initializes the priority queue $\mathcal{Q} = \{\langle V_1^a, (V_s, V_1^a), T, D_1^a+T\rangle, \langle V_2^b, (V_s, V_2^b), T, D_2^b+T\rangle\}$ and constructs the target list $\mathcal{T} = \{\langle V_9^c, T, T/2\rangle\}$. Note that the leftmost label in $\mathcal{Q}$ always contains the path with the lowest *cost*() value. At the first iteration, the algorithm pops label $\langle V_1^a, (V_s, V_1^a), T, D_1^a+T\rangle$, considers transport edge E_{13}^a, and pushes $\langle V_3^a, p'_1(V_s, V_1^a, V_3^a), T, D_3^a+T\rangle$ to $\mathcal{Q}$. Next, at the second iteration, SP pops label $\langle V_3^a, p'_1(V_s, V_1^a, V_3^a), T, D_3^a+T\rangle$ from $\mathcal{Q}$, considers the transfer edge E_{34}^{ac}, and pushes $\langle V_4^c, p''_1(V_s, V_1^a, V_3^a, V_4^c), 3 \cdot T, D_4^c + T\rangle$ to $\mathcal{Q}$ (remember $A_4^c < C_{p'_1} < D_4^c$). The next two iterations are similar, and thus, after the fourth iteration we have:

$$\mathcal{Q} = \{\langle V_4^c, p''_1(V_s, V_1^a, V_3^a, V_4^c), 3 \cdot T, D_4^c + T\rangle, \quad \text{and } p_{cand} = \textbf{null}$$
$$\langle V_8^c, p''_2(V_s, V_2^b, V_6^b, V_8^c), 3 \cdot T + C_{p'_2} - D_8^c, C_{p'_2} + T\rangle\}$$

Now, at the next two iterations, SP expands path p''_1 considering transport edges E_{48}^c and E_{89}^c as $O_{p''_1} < O_{p''_2}$. Therefore, at the seventh iteration, the algorithm pops label $\langle V_9^c, (V_s, V_1^a, V_3^a, V_4^c, V_8^c, V_9^c), 3 \cdot T, D_9^c + T\rangle$ from $\mathcal{Q}$. Since the target list $\mathcal{T}$ contains an entry for vertex V_9^c, SP identifies candidate solution $p_{cand} = p_1(V_s, V_1^a, V_3^a, V_4^c, V_8^c, V_9^c, V_e)$ with $O_{p_1} = 4 \cdot T$ and $C_{p_1} = D_9^c + \frac{3 \cdot T}{2}$. Finally, assuming without loss of generality that $D_6^b > D_8^c$ also holds and therefore, $C_{p'_2} - D_8^c = D_6^b + T - D_8^c > T$, at the eighth iteration, the algorithm pops $\langle V_8^c, p''_2(V_s, V_2^b, V_6^b, V_8^c), 3 \cdot T + C_{p'_2} - D_8^c, C_{p'_2} + T\rangle$ and terminates the search because $O_{p''_2} = 3 \cdot T + C_{p'_2} - D_8^c > 4 \cdot T = O_{p_1}$ and thus, $cost(p''_2) > cost(p_1)$. The solution to the dPDPT request (n_s, n_e) is $p_1(V_s, V_1^a, V_3^a, V_4^c, V_8^c, V_9^c, V_e)$.

5 Experimental Evaluation

In this section, we present an experimental study of our methodology for solving dynamic Pickup and Delivery Problem with Transfers (dPDPT). We compare SP against HT, a method inspired by [15, 24] that combines an insertion heuristic with tabu search. All methods are written in C++ and compiled with gcc. The evaluation is carried out on a 3Ghz Core 2 Duo CPU with 4GB RAM running Debian Linux.

5.1 The HT Method

Satisfying dPDPT requests with HT involves two phases. In the first phase, for every new dPDPT request, the method employs the cheapest insertion heuristic to include the pickup n_s and the delivery location n_e in a vehicle route. The idea is the following. HT examines every vehicle route r_a and for each pair of consecutive locations n_i and n_{i+1} in r_a (forming an insertion "slot"), it computes the detour cost $DS = dist(n_i, n_s) + dist(n_s, n_{i+1}) - dist(n_i, n_{i+1})$ for inserting

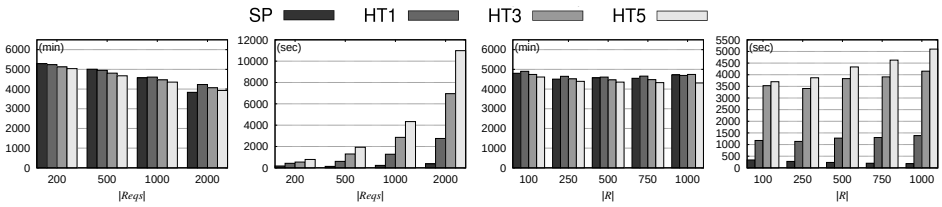
pickup n_s (resp. delivery n_e) in between n_i and n_{i+1}. The detour cost DS signifies the extra time vehicle r_a must spend and therefore, it increases the total operational cost. Then, HT selects the best overall insertion, i.e., the route r_a and the "slots", such that the combined detour cost for inserting both pickup n_s and delivery location n_e is minimized.

The second phase of HT takes place periodically after k requests are satisfied with the cheapest insertion. It involves a tabu search procedure that reduces the total operating cost. At each iteration, the tabu search considers every satisfied request and calculates what would be the change (increase or decrease) in the total operational cost removing the request from its current vehicle route r_a and inserting it to another r_b. Then, the tabu search selects the request with the best combination of removal and insertion, and performs these actions. Finally, the selected combination is characterized as tabu and cannot be executed for a number of future iterations.

5.2 Experiments

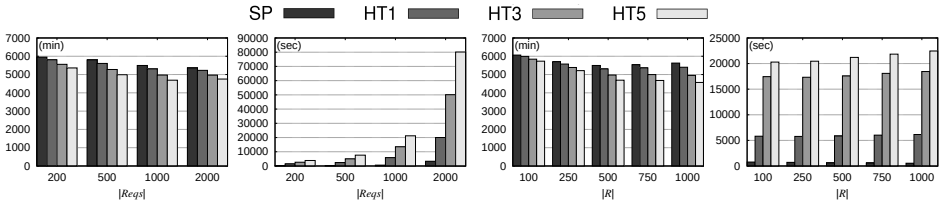
To conduct our experiments, we consider the road networks of two cities; Oldernburg (OL) with 6,105 spatial locations (Figure 4), and Athens (ATH) with 22,601 locations (Figure 5). First, we generate random pickup and delivery requests at each network and exploit the HT method to construct collections of vehicle routes varying either the number of routes $|R|$, from 100 to 1000, or the number of requests $|Reqs|$ involved, from 200 to 2000. Then, for each of these route collections, we generate 500 random dPDPT requests and employ the SP and the HT method to satisfy them. For HT, we introduce three variations HT1, HT3 and HT5 such that the tabu search is invoked once (after 500 requests are satisfied), three times (after 170) and five times (after 100), respectively. In addition, each time the tabu search is invoked, it performs 10 iterations. For each method, we measure (1) the increase in the total operational cost of the vehicles after all 500 requests are satisfied (sub-figures (a) and (c)) and (2) the total time needed to satisfy the requests. Finally, note that we store both the road network and the vehicle routes on disk and that, we consider a main-memory cache mechanism capable of retaining 10% of the total space occupied on disk.

Examining Figures 4 and 5 we make the following observations. The SP method requires significantly less time to satisfy the 500 ad-hoc dPDPT requests, for all the values of the $|Reqs|$ and $|R|$ parameters, and for both the underlying road networks. In fact, when varying $|R|$, SP is always one order of magnitude faster than all three HT variants. In contrast, SP results in slightly increased total operational cost compared to HT, in most of the cases, and especially for large road networks as ATH. However, this advantage of HT comes with a unavoidable trade-off between the increase of the total operational cost and the time needed to satisfy the ad-hoc dPDPT requests. The more often HT employs the tabu search, the lower the increase of the total operational cost of the vehicles is. But, on the other hand, since each iteration of the tabu search needs to examine every route and identify the best reassignment for all the existing requests, the total time of HT5 is higher than the time of HT3 and HT1.



(a) Operational cost increase (b) Total execution time (c) Operational cost increase (d) Total execution time

Fig. 4. City of Oldenburg (OL) Road Network.



(a) Operational cost increase (b) Total execution time (c) Operational cost increase (d) Total execution time

Fig. 5. City of Athens (ATH) Road Network.

Finally, we notice that as the number of pickup and delivery requests $|Reqs|$ involved in the initial static plan increases, satisfying the 500 ad-hoc dPDPT requests, either with HT or SP, results in a lower increase of the total operational cost but the total time needed to satisfy these requests increases. Notice that this is true regardless of the size of the underlying road network. As $|Reqs|$ increases and while $|R|$ remains fixed, the vehicle routes contain more spatial locations. This provides more insertion "slots" and enables both HT and SP to include the pickup and the delivery location of a dPDPT request with a lower cost. On the other hand, HT slows down since it has to examine the reassignment of more requests during the tabu search, and SP needs more time because the dynamic plan graph is larger. Similar observations can be made in case of varying the number of routes $|R|$.

6 Conclusions

This work studies the dynamic Pickup and Delivery Problem with Transfers (dPDPT). This is the first work addressing the dynamic flavor of the problem. We propose a methodology that formulates dPDPT as a graph problem and identifies the solution to a request as the shortest path from a node representing the pickup location to that of the delivery location. Our experimental analysis shows that our method is able to find dPDPT solutions significantly faster than

a conventional two-phase local search algorithm, while the cost of the solution is only marginally increased.

References

1. R. Agrawal and H. V. Jagadish. Materialization and incremental update of path information. In *ICDE*, pages 374–383, 1989.
2. R. Bent and P. V. Hentenryck. A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows. *Computers and Operations Research*, 33(4):875–893, 2006.
3. G. Berbeglia, J.-F. Cordeau, I. Gribkovskaia, and G. Laporte. Static pickup and delivery problems: a classification scheme and survey. *TOP*, 15:1–31, 2007.
4. G. Berbeglia, J.-F. Cordeau, and G. Laporte. Dynamic pickup and delivery problems. *European Journal of Operational Research*, 202(1):8–15, 2010.
5. R. L. Carraway, T. L. Morin, and H. Moskowitz. Generalized dynamic programming for multicriteria optimization. *European Journal of Operational Research*, 44(1):95–104, January 1990.
6. J. Cheng and J. X. Yu. On-line exact shortest distance query processing. In *EDBT*, pages 481–492, 2009.
7. E. Cohen, E. Halperin, H. Kaplan, and U. Zwick. Reachability and distance queries via 2-hop labels. In *SODA*, pages 937–946, 2002.
8. K. L. Cooke and E. Halsey. The shortest route through a network with time-dependent internodal transit times. *Journal of Mathematical Analysis and Applications*, 14(3):493–498, 1966.
9. J.-F. Cordeau. A branch-and-cut algorithm for the dial-a-ride problem. *Operations Research*, 54(3):573–586, 2006.
10. J.-F. Cordeau, G. Laporte, and S. Ropke. *Vehicle Routing: Latest Advances and Challenges*, chapter Recent Models and Algorithms for One-to-One Pickup and Delivery Problems, pages 327–357. Kluwer, 2008.
11. C. E. Cortés, M. Matamala, and C. Contardo. The pickup and delivery problem with transfers: Formulation and a branch-and-cut solution method. *European Journal of Operational Research*, 200(3):711–724, 2010.
12. B. Ding, J. X. Yu, and L. Qin. Finding time-dependent shortest paths over large graphs. In *EDBT*, pages 205–216, 2008.
13. S. E. Dreyfus. An appraisal of some shortest-path algorithms. *Operations Research*, 17(3), 1969.
14. Y. Dumas, J. Desrosiers, and F. Soumis. The pickup and delivery problem with time windows. *European Journal of Operational Research*, 54(1):7–22, September 1991.
15. M. Gendreau, F. Guertin, J.-Y. Potvin, and R. Séguin. Neighbourhood search heuristics for a dynamic vehicle dispatching problem with pick-ups and deliveries. *Transportation Research Part C: Emerging Technologies*, 14(3):157 – 174, 2006.
16. A. V. Goldberg and C. Harrelson. Computing the shortest path: A search meets graph theory. In *SODA'05: Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 156–165, 2005.
17. A. V. Goldberg, H. Kaplan, and R. F. Werneck. Reach for A*: Efficient point-to-point shortest path algorithms. In *Proc. of the 8th WS on Algorithm Engineering and Experiments (ALENEX). SIAM, Philadelphia*, pages 129–143, 2006.

18. J. Granat and F. Guerriero. The interactive analysis of the multicriteria shortest path problem by the reference point method. *European Journal of Operational Research*, 151(1):103–118, November 2003.
19. F. Guerriero and R. Musmanno. Label correcting methods to solve multicriteria shortest path problems. *Journal of Optimization Theory and Applications*, 111(3):589–613, 2001.
20. N. Jing, Y.-W. Huang, and E. A. Rundensteiner. Hierarchical encoded path views for path query processing: An optimal model and its performance evaluation. *TKDE*, 10(3):409–432, 1998.
21. S. Jung and S. Pramanik. An efficient path computation model for hierarchically structured topographical road maps. *TKDE*, 14(5):1029–1046, 2002.
22. H.-P. Kriegel, M. Renz, and M. Schubert. Route skyline queries: A multi-preference path planning approach. In *ICDE*, pages 261–272, 2010.
23. H. Li and A. Lim. A metaheuristic for the pickup and delivery problem with time windows. In *ICTAI*, pages 333–340, 2001.
24. S. Mitrović-Minić, R. Krishnamurti, and G. Laporte. Double-horizon based heuristics for the dynamic pickup and delivery problem. *Transportation Research Part B: Methodological*, 38(7):669–685, 2004.
25. S. Mitrović-Minić and G. Laporte. Waiting strategies for the dynamic pickup and delivery problem with time windows. *Transportation Research Part B: Methodological*, 38(7):635–655, 2004.
26. S. Mitrović-Minić and G. Laporte. The pickup and delivery problem with time windows and transshipment. *INFOR*, 44(3):217–228, 2006.
27. A. Orda and R. Rom. Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length. *J. ACM*, 37(3):607–625, 1990.
28. S. Parragh, K. Doerner, and R. Hartl. A survey on pickup and delivery problems, Part II: Transportation between pickup and delivery locations. *Journal für Betriebswirtschaft*, 58:81–117, 2008.
29. I. Pohl. Bi-directional search. *Machine Intelligence*, 6:127–140, 1971.
30. D. A. Popken. Controlling order circuity in pickup and delivery problems. *Transportation Research Part E: Logistics and Transportation Review*, 42(5):431–443, 2006.
31. S. Ropke and J.-F. Cordeau. Branch and cut and price for the pickup and delivery problem with time windows. *Transportation Science*, 43(3):267–286, 2009.
32. S. Ropke, J.-F. Cordeau, and G. Laporte. Models and branch-and-cut algorithms for pickup and delivery problems with time windows. *Networks*, 49(4):258–272, 2007.
33. S. Ropke and D. Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(4):455–472, 2006.
34. M. Sigurd, D. Pisinger, and M. Sig. Scheduling transportation of live animals to avoid spread of diseases. *INFORMS transportation science*, 38:197–209, 2004.
35. Y. Tian, K. C. K. Lee, and W.-C. Lee. Finding skyline paths in road networks. In *GIS*, pages 444–447, 2009.
36. H. Xu, Z.-L. Chen, S. Rajagopal, and S. Arunapuram. Solving a practical pickup and delivery problem. *Transportation Science*, 37(3):347–364, 2003.