Transportation Science

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

An Adaptive Large Neighborhood Search for the Pickup and Delivery Problem with Transfers

Renaud Masson, Fabien Lehuédé, Olivier Péton,

To cite this article:
Renaud Masson, Fabien Lehuédé, Olivier Péton, (2013) An Adaptive Large Neighborhood Search for the Pickup and Delivery Problem with Transfers. Transportation Science 47(3):344-355. https://doi.org/10.1287/trsc.1120.0432

TRANSPORTATION SCIENCE
Vol. 47, No. 3, August 2013, pp. 344–355
ISSN 0041-1655 (print) | ISSN 1526-5447 (online)

http://dx.doi.org/10.1287/trsc.1120.0432

An Adaptive Large Neighborhood Search for the Pickup and Delivery Problem with Transfers

Renaud Masson, Fabien Lehuédé, Olivier Péton
LUNAM Université, École des Mines de Nantes, IRCCyN UMR CNRS 6597, F-44307 Nantes cedex 3, France
{renaud.masson@mines-nantes.fr, fabien.lehuede@mines-nantes.fr, olivier.peton@mines-nantes.fr}

The pickup and delivery problem (PDP) consists in defining a set of routes that satisfy transportation requests between a set of pickup points and a set of delivery points. This paper addresses a variant of the PDP where requests can change vehicle during their trip. The transfer is made at specific locations called "transfer points." The corresponding problem is called the pickup and delivery problem with transfers (PDPT). Solving the PDPT leads to new modeling and algorithmic difficulties. We propose new heuristics capable of efficiently inserting requests through transfer points. These heuristics are embedded into an adaptive large neighborhood search. We evaluate the method on generated instances and apply it to the transportation of people with disabilities. On these real-life instances we show that the introduction of transfer points can bring significant improvements (up to 9%) to the value of the objective function.

Key words: pickup and delivery problem; transfers; adaptive large neighborhood search
History: Received: March 2011; revision received: January 2012; accepted: May 2012. Published online in *Articles in Advance* September 5, 2012.

1. Introduction

The pickup and delivery problem (PDP) consists in defining a set of minimum cost routes in order to satisfy a set of transportation requests. Each transportation request has a known origin called the *pickup point* and a destination called the *delivery point*. Several users can share a vehicle as long as its capacity is not exceeded. The PDP is a well-known extension of the vehicle routing problem (VRP). It can be expressed either in a static version, where each request is known in advance, or in a dynamic version, where new requests may be integrated at any time (Berbeglia, Cordeau, and Laporte 2010). If time windows on requests are considered, the problem is called the pickup and delivery problem with time windows (PDPTW). Requests can correspond to the transportation of goods or people. Depending on the context of the application, various objective functions may be considered: number of vehicles used, total ride time, total distance, cost or quality of service-related criteria. In what follows, we consider a static case problem related to the transportation of people with disabilities. The objective function is to minimize the total distance traveled.

This paper focuses on a recent extension of the PDPTW where requests can be transferred from one vehicle to another at intermediate points between the pickup and delivery points. In this case, a first vehicle drives the request to a predetermined location called *transfer point*. A second vehicle picks up the request at the transfer point and drives it to the delivery point. This extension is called the pickup and delivery problem with transfers (PDPT). In the PDPT, the implicit constraint stating that the pickup and delivery points of a given request should be serviced by the same vehicle is relaxed. New precedence constraints state that if a request uses a transfer point, it must be delivered at the transfer point by a first vehicle before being picked up by a second vehicle. Any pickup and delivery problem can be viewed as a PDPT with no transfer point, so the PDPT is NP-hard.

The PDP has been intensively studied over recent decades, Berbeglia et al. (2007) and Parragh, Doerner, and Hartl (2008) present detailed reviews. On the other hand, the PDPT as devised in this article has been the subject of very few works. Mitrović-Minić and Laporte (2006) propose a local search method for the uncapacitated PDPT with a Manhattan distance. They solve generated instances with up to 100 requests. This method was adapted in a tabu search by Masson, Lehuédé, and Péton (2011) for the dial-a-ride problem with transfers. In this study, the authors underline the complexity of enforcing the maximum ride time constraint. A comprehensive mathematical formulation of the PDPT is proposed by Cortés, Matamala, and Contardo (2010). The authors use a branch-and-cut algorithm for solving instances with six requests and two vehicles to optimality.

Regarding related exact approaches, Lin (2008) presents a PDPTW where all requests share the same

delivery location. Nevertheless, the delivery time windows are different. In this problem, it is considered that a transfer can occur on the last pickup before a delivery. Instances with up to 100 requests are solved to optimality using a commercial solver. Kerivin et al. (2008) consider a PDP where every request can be split as well as transferred from one vehicle to another at every node of the problem. This problem has no time window and is solved using branch and cut. Some instances with up to 15 requests are solved to optimality. Nakao and Nagamochi (2010) calculate a lower bound for the PDPT with a single transfer point. Transfers also appear in the school bus model of Fugenschuh (2009). In this work, the routes are already designed and the objective is to minimize the number of buses needed. Some passengers can be transferred from one bus to another but the problem is very different.

The use of transfer points has also been considered, for solving pickup and delivery problems with particular restrictions. This possibility was first raised by Shang and Cuff (1996). These authors consider that any point can be used as a transfer point. Transfers are only considered to insert a request that cannot be inserted in the current solution without resorting to an extra vehicle. A heuristic construction relies on the construction of miniroutes that are assigned to vehicles. Thangiah, Fergany, and Awam (2007) use the same principles in a real-time version of the problem. Oertel (2000) considers two variants where all the points or only a subset of locations can be considered as transfer points. A heuristic method and a column generation heuristic are presented. Gørtz, Nagarajan, and Ravi (2009) consider a version of the PDPT where the objective is to minimize the makespan. Heuristic methods are proposed to solve the uncapacitated and capacitated cases. A heuristic column generation method is proposed by Mues and Pickl (2005). They consider a problem with a single transfer point through which requests are systematically routed. The set of routes is composed of pickup routes ending at the transfer point and delivery routes starting at this transfer point. Instances with up to 70 requests are considered for experiments. Cortés and Jayakrishnan (2002) present a feasibility simulation for a demand-responsive transit system allowing a single transfer per passenger. Petersen and Ropke (2011) present another ALNS to handle freight transportation problems considering one transfer point and up to 982 requests.

The application that motivates this work is a demand responsive transport problem for people with disabilities. This concerns people who require daily trips from their home to schools, social centers, or vocational rehabilitation centers. We minimize the total distance traveled, which may deteriorate the quality of service provided to users. Therefore, time windows on pickup and delivery points have to be carefully chosen in order to ensure the desired quality of service. Without loss of generality, we consider outbound trips from home to a destination. In this case, the number of pickup points is significantly larger than the number of delivery points. Transfers correspond to a practice followed in some vocational rehabilitation centers that aims to reduce transportation costs. People that live in the same geographical area are picked up by one vehicle, independently of their destination. The passengers are delivered at the closest school or center where they spread in other vehicles that drive them to their final destination. However, the transfer operation may also have a negative impact on the quality of service because direct trips are generally preferred by passengers. Thus, the decision makers have to balance the expected benefits and drawbacks of resorting to transfers.

In this article, we focus on designing a method capable of efficiently exploring the search space of the problem. Therefore, a few practical constraints are relaxed. We consider a homogeneous fleet of vehicles and ignore the maximum ride-time constraint. In the instances considered, ride time is limited because of adequate time windows. We assume that the passengers take two vehicles at most and thus only one transfer per trip is authorized. Moreover, no fixed cost is associated with the use of a vehicle or with the transfer operation.

We propose new heuristics capable of efficiently inserting requests through transfer points. These heuristics are embedded into an adaptive large neighborhood search (ALNS). We evaluate the method on generated instances and apply it to the transportation of people with disabilities. One purpose of this study is to evaluate the potential output of transfers. To do so we compare solutions of the ALNS with and without considering transfer points. To provide a lower bound on the savings that can be achieved thanks to the use of transfer, we also compare upper bounds of the PDPT solutions with lower bounds of PDPTW solutions.

The remainder of this paper is organized as follows. Section 2 presents a formulation of the PDPT. Sections 3 and 4 are devoted to the solution method for the problem without and with transfer points, respectively. Computational results are presented in §5, followed by the conclusion in §6.

2. Problem Formulation

In this section, we give a description of the PDPT. We consider a set of n requests, $R = \{1, \ldots, n\}$, a set T of transfer points, and a set K of homogeneous vehicles of capacity Q. The pickup and delivery nodes of the request $i \in R$ are designated by $p(i)$

and $d(i)$. The sets of all pickup and delivery points are denoted P and D; $o(k)$ and $o'(k)$ represent the starting and the ending depot of route $k \in K$, and the set of all depot locations is represented by O. The PDPT is defined on a complete directed graph $G = (V, A)$, where $V = P \cup D \cup O \cup T$ represents the set of all vertices and $A = \{(v_i, v_j) \mid v_i, v_j \in V, i \neq j\}$ the set of all arcs. With each arc $(v_i, v_j) \in A$ is associated a nonnegative travel time $l_{i,j}$. A time window $[e_i, l_i]$ is associated with each vertex $v_i \in V$, where e_i and l_i represent the earliest and the latest time at which the service can begin. Each vertex has a known service time s_i representing the time needed to get users in or out of the vehicle. We consider that a request i concerns q_i passengers who have the same pickup and delivery nodes. At any moment, the number of users carried simultaneously by a vehicle cannot exceed Q. In addition, the service at each vertex j should begin in the interval $[e_j, l_j]$. A vehicle is allowed to wait at a vertex in order to service it within its time windows.

A solution of the PDPT is a set of $|K|$ routes that satisfies all the requests and such that route k starts at $o(k)$ and ends at $o'(k)$. For every request $i \in R$, vertices $p(i)$ and $d(i)$ can be served by the same route, $d(i)$ being serviced after $p(i)$. Vertices $p(i)$ and $d(i)$ can also be served by distinct routes $k_1 \in K$ and $k_2 \in K$. In this case, k_1 and k_2 both have to service one transfer point v_j such that

(i) v_j is serviced after $p(i)$ in route k_1;

(ii) v_j is serviced before $d(i)$ in route k_2; and

(iii) v_j is serviced in route k_1 before being serviced in route k_2.

A mixed integer linear program of the PDPT has been proposed by Cortés, Matamala, and Contardo (2010). Because this model contains 31 sets of constraints, we do not reproduce it here.

3. ALNS for the PDPTW

In this section, we describe the ALNS developed for this paper. The ALNS has been described extensively by Pisinger and Ropke (2007) and applied to the PDPTW by Ropke and Pisinger (2006). We use the same approach to solve the PDPTW as a basis for comparison with the PDPT. It also provides a basic framework in which the specific developments for the PDPT, described in §4, are integrated.

3.1. Main Scheme of the Large Neighborhood Search (LNS)

The large neighborhood search (LNS) was introduced by Shaw (1998) in a constraint programming framework to solve the vehicle routing problem with time windows (VRPTW). The LNS is similar to the *ruin and recreate* introduced by Schrimpf et al. (2000). An extensive description of the method and of its application to combinatorial optimization problems can be found in the recent review by Ropke and Pisinger (2010). The underlying principle of the LNS is to destroy and repair a solution iteratively in order to improve it. The general functioning of the LNS is depicted in Algorithm 1.

Algorithm 1 (LNS)
Require: *InitialSolution*
1: *BestSolution* ← *InitialSolution*
2: *CurrentSolution* ← *InitialSolution*
3: **while** the termination criterion is not satisfied **do**
4: Selection of *Destroy* and *Repair* heuristics
5: $S \leftarrow$ *CurrentSolution*
6: $S \leftarrow Destroy(S)$
7: $S \leftarrow Repair(S)$
8: **if** $S <$ *BestSolution* **then**
9: *BestSolution* ← S
10: *CurrentSolution* ← S
11: **else**
12: **if** *TransitionAccepted*(S, *CurrentSolution*) **then**
13: *CurrentSolution* ← S
14: **end if**
15: **end if**
16:
17: **end while**
18: **return** *BestSolution*

The latest LNS methods integrate several heuristics for destroying and repairing the current solution (lines 6 and 7); they accept deterioration of the objective during the search (line 12), using, e.g., a simulated annealing acceptance criterion (Kirkpatrick, Gelatt, and Vecchi 1983) or a record-to-record travel (RRT) algorithm (Laporte, Musmanno, and Vocaturo 2010). Ropke and Pisinger (2006) present an ALNS for the PDPTW. The search is called adaptive because the probability of choosing each destroy and repair method is reevaluated periodically depending on their efficiency in the past iterations. Its main components are reviewed in the following subsections.

At each iteration, r% of the requests are removed from the current solution. Requests are reinserted in the new solution using repair heuristics until all requests have been planned or no feasible insertion can be found. Partial solutions can be used as intermediate unfeasible solutions in the algorithm, unrouted requests are penalized in the objective function or placed in a so-called request bank.

3.2. Destruction Neighborhoods

We implemented four destruction heuristics:

1. *Worst removal*. This heuristic first computes the cost savings produced by the removal of each request; r% of the requests are then selected randomly to be removed. The probability of being selected increases with the savings.

2. *Random removal*. This heuristic randomly selects r% of the requests to be removed from the current solution.

3. *Related removal*. This heuristic aims at removing related nodes. The relatedness measure of two requests (Shaw 1998) depends on the distance between their pickups and their deliveries, the difference between the time of service at their nodes and the difference in load of the two requests. Complete description of this operator is given by Ropke and Pisinger (2006).

4. *History removal*. This heuristic is inspired from the two history removal heuristics presented by Pisinger and Ropke (2007). It aims to remove the requests that seem poorly placed in the current solution with regard to the best-known solutions. For each node j and j' of the problem, $\xi_{j,j'}$ is the number of solutions among the 50 best-known solutions in which the node j is directly followed by the node j' in a route. Let us denote $j-1$ and $j+1$ the predecessor and successor of a node j in the current solution. For each request i of the problem we define a score ϕ_i as follows:

$$\phi_i = \xi_{p(i)-1,p(i)} + \xi_{p(i),p(i)+1} + \xi_{d(i)-1,d(i)} + \xi_{d(i),d(i)+1}. \quad (1)$$

The m requests with the lowest ϕ_i are removed.

3.3. Repair Neighborhoods

We use two insertion heuristics based on best insertion and regret principles.

1. *Best insertion*. At each iteration, the best insertion cost is computed for each destroyed request and the request with the lowest insertion cost is inserted at its best position. The heuristic stops when all requests are routed or none can be inserted (Ropke and Pisinger 2006).

2. *Regret heuristics*. This heuristic is based on the notion of regret used, for example, by Potvin and Rousseau (1993) for the vehicle routing problem with time windows (VRPTW), and proposed in a parametrized version by Ropke and Pisinger (2006). For each request i in the set U of unplanned requests, Δf_i^j designates the insertion cost of the request i in the jth best route at its best position. At each iteration, the request $i^\star$ selected for insertion at its best position is chosen such that $i^\star = \arg\max_{i\in U}(\sum_{j=2}^{k} \Delta f_i^j - \Delta f_i^1)$. The heuristic stops when no more destroyed requests can be inserted in a route or if all requests are inserted. In this paper, we consider four regret-k heuristics with values of k between two and five.

3.4. Acceptance and Stopping Criteria

The decision to accept a new solution is taken according to a simulated annealing criterion. At each iteration, the temperature used by the simulated annealing is multiplied by u. The parameter u is chosen in $]0, 1[$ such that the temperature at the last iteration is equal to 0.2% of the starting temperature. Like Ropke and Pisinger (2006), we set the starting temperature in such a way that a solution 5% worse than the initial solution has a 50% chance of being accepted. In addition, as proposed by Pisinger and Ropke (2007), noise is applied to the objective function as a diversification operator. The stopping criterion for the whole process is the maximum number of iterations.

3.5. Adaptive Aspects

The solution is modified by a destruction and a repair neighborhood. With each neighborhood i is associated a weight w_i and a score π_i. The neighborhoods are selected using a roulette-wheel selection procedure. On the first iteration, all neighborhoods have the same weight. Each time a neighborhood is called, its score is updated depending on its performance. The interested reader can refer to Ropke and Pisinger (2006) for more detail. If the use of a neighborhood leads to a new best solution, its score is incremented by some value σ_1. If the solution generated is better than the current solution, the score is augmented by σ_2. Finally, if the new solution is accepted as a new current solution and has never been encountered before, the score of the neighborhood is increased by σ_3. The entire search is divided into time segments. Here we consider time segments of 100 iterations. At the end of each time segment, the weights of the neighborhoods are updated using the expression $w_i \leftarrow (1-\alpha)w_i + \alpha\pi_i/\theta_i$, where θ_i is the number of times the neighborhood has been called during the last time segment and $\alpha \in [0, 1]$ is a reaction factor that controls the inertia of the weight adjustment. In our implementation, we use the same values as Ropke and Pisinger (2006) for σ_1, σ_2, σ_3. We set $\alpha = 0.5$ instead of $\alpha = 0.1$ to ensure a greater reactivity when a small number of iterations is performed.

3.6. Efficiency of the Algorithm

This algorithm has been implemented and evaluated on the benchmark instances proposed by Li and Lim (2002) for the PDPTW. In these instances, the objective is first to minimize the number of vehicles used in the solution and then the total distance driven. Among the 354 instances, the number of vehicles has been improved in 18 cases and the distance traveled in 55 cases. For 36% of the instances, the implemented method returns the best-known solution. When we obtain the minimum number of vehicles but a suboptimal distance traveled, this distance is within 2% of the best-known result in 73% of the cases. In only 8% of the instances, we do not succeed in finding a solution with the minimum number of vehicles. According to these results we consider that the implemented ALNS gives satisfactory reference results for the PDP without transfer.

4. ALNS for the PDP with Transfers

In this section, we describe the new developments introduced to consider the specificities of the PDPT. This includes not only destruction and repair neighborhoods but also heuristics for selecting appropriate transfer points in neighborhoods (§4.3). Section 4.4 presents the feasibility test used in insertion procedures.

4.1. Destruction Neighborhoods

4.1.1. Transfer Point Removal Heuristic. This heuristic is inspired from a neighborhood for a hub routing problem (Lapierre, Ruiz, and Soriano 2004). The original idea consists of rerouting all requests from one hub to another hub. Requests that use a given transfer point are removed simultaneously to give them a chance to be rerouted through another transfer point.

We first select one active transfer point randomly. If the number of requests going through this transfer point is less than or equal to the number of requests to be destroyed, all these requests are removed. Otherwise, we randomly select a subset of requests to remove. If the number of removed requests is less than r, we remove requests that minimize the distance of their pickup and the closest pickup of an unplanned request plus the distance of their delivery and the closest delivery of an unplanned request. This can be seen as an extension of a relatedness measure based only on the distance between a request and the set of unplanned requests.

Figure 1 illustrates a case where this neighborhood leads to a more profitable solution. If all requests using t_2 are reinserted with a transfer first heuristic (see 4.2.2) on a subset of transfers limited to $\{t_1\}$ (see 4.3), a better solution is obtained.

4.1.2. Pickup/Delivery Cluster Removal Heuristic. This heuristic aims to remove simultaneously a given number of requests that can be efficiently routed through a common transfer point. Two requests, or more, may benefit from using one transfer point if their pickup or their delivery locations form a cluster. If the pickups of these requests are close, they can be serviced together by a first vehicle and then carried to a transfer point in order to be assigned to distinct vehicles. Symmetrically, if their deliveries are located in the same area, they can be serviced by various vehicles for the first part of their journey and then gathered into a single vehicle at a transfer point. The root node of the cluster is selected randomly. All the pickups (or deliveries) are listed from the closest to the farthest to the root node. We iteratively remove the request in position $\lfloor y^p \times (|P| - |U|) \rfloor$ in this list. In this expression, U is the set of unplanned requests, p is a deterministic parameter equal to nine in our implementation, and y is randomly chosen in $[0, 1[$.

4.1.3. History Removal. This heuristic is an extension of the PDPTW history removal presented in §3.2. Let us consider that a transfer point t is decomposed into two nodes for each transferred request j at this node: t_j^- corresponds to the delivery of j at t and t_j^+ is the pickup node of j at t. The score ϕ_i of a transferred request i is computed as follows:

$$\phi_i = \tfrac{1}{2}\big(\xi_{t_i^- - 1,\, t_i^-} + \xi_{t_i^-,\, t_i^- + 1} + \xi_{t_i^+ - 1,\, t_i^+} + \xi_{t_i^+,\, t_i^+ + 1} + \xi_{p(i)-1,\, p(i)} + \xi_{p(i),\, p(i)+1} + \xi_{d(i)-1,\, d(i)} + \xi_{d(i),\, d(i)+1}\big). \quad (2)$$

The 1/2 coefficient makes this score comparable with the score defined by Equation (1) in the case of nontransferred requests.

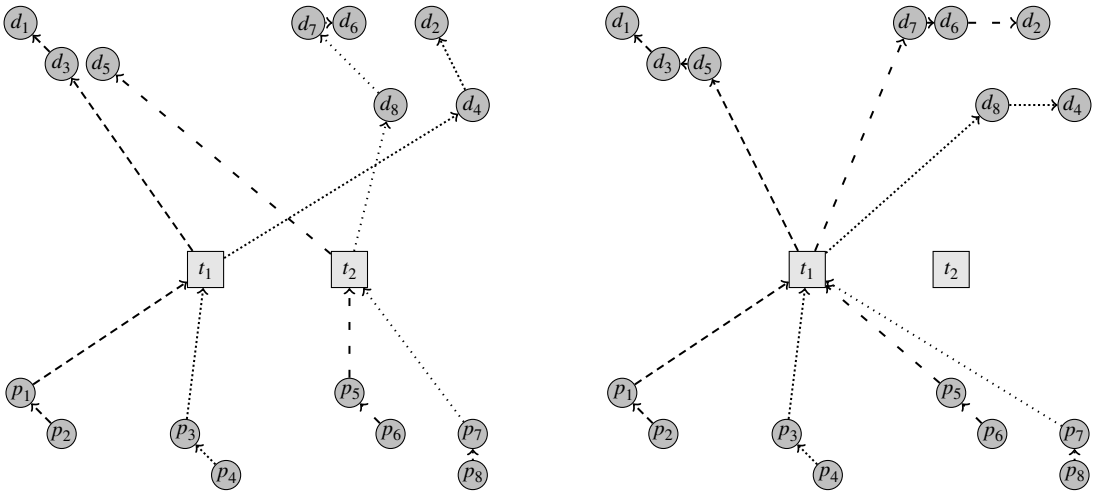

Figure 1 Example of Transfer Point Removal

4.2. Repair Neighborhoods

The heuristics presented in the next three sections are insertion heuristics specifically designed for the PDPT.

4.2.1. Best Insertion with Transfer. This neighborhood is an adaptation of the neighborhood of Mitrović-Minić and Laporte (2006) for the PDPT. For each unplanned request the best insertion without considering any transfer opportunity is first evaluated. Then for each transfer point t and each unplanned request $(p(i), d(i))$, we evaluate the insertion cost as follows:

(i) The insertion cost of the pair $(p(i), t)$ is evaluated and considered as inserted in its best position. Then, an evaluation of the insertion cost of the pair $(t, d(i))$ is done.

(ii) The insertion cost of the pair $(t, d(i))$ is evaluated and considered as inserted in its best position. Then, an evaluation of the insertion cost of the pair $(p(i), t)$ is done for every route.

Finally, the best insertion among all evaluated insertions is performed.

4.2.2. Transfer First. The best insertion with transfer evaluates the insertion of each request sequentially and may miss good opportunities to use transfer points. Figure 2 presents such a case.

It seems locally better to route requests $(p1, d1)$ or $(p2, d2)$ directly (cost = 10) rather than through the transfer point t (cost = 12). However regrouping the requests at transfer point t (using arcs $p1 \to t$, $p2 \to t$ and $t \to d1$) has a cost of 18 whereas the solution without transfers ($p1 \to d1$ and $p2 \to d2$) has a cost of 20.

The transfer first neighborhood gives priority to the use of transfer points. While unplanned requests remain, best insertions with transfers. If no feasible insertion with transfer can be found, the best insertion without transfer is considered. When all requests have been inserted, a post-processing step consists in iteratively removing each transferred request and trying to reinsert it without transfer. This step aims at detecting forced transfers that reduce the quality of the current solution.

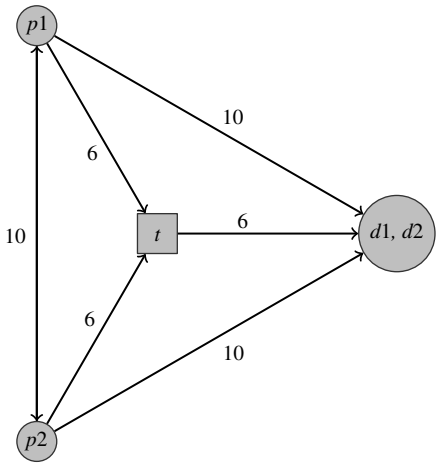

Figure 2 **Example Where Forcing the Use of Transfer Points Improves the Quality of the Solution**

4.2.3. Regret Insertion with Transfer. This heuristic facilitates the insertion of requests for which using a transfer point is cheaper than the insertion without transfer. For each destroyed request we compute the difference between the insertion cost of this request using a transfer point and the best insertion cost without transferring the request. The request that has the largest difference is inserted first in its best position.

4.3. Selecting a Subset of Transfer Points

When a problem has a large number of transfer points, considering each of these locations in insertion heuristics is time consuming. Moreover, in some cases, the use of a transfer point is only interesting if a large enough number of requests use it. Considering a small number of transfer points can help to find good solutions. Therefore, we propose to consider only a subset of transfer points in our heuristics.

We propose five methods to determine subsets of transfer points:

1. The first method consists in randomly selecting a subset of m potential transfer points that will be considered for each request.

2. We define the distance between a request and a transfer point as the length of the following trip: pickup point → transfer point → delivery point. For all requests, the second method consists in selecting only the m closest transfer points.

3. The third method exploits the idea that transfer points are advantageous only when several transshipments are performed at the same point. It considers the m transfer points in which the greatest number of requests is transferred after the destruction operation.

4. The fourth method uses a simplification of the PDPT, known as the k-star hub problem Blasum et al. (2007), which is solved using a commercial solver. It is defined as follows: we consider a set of requests and k transfer points. Each request can be delivered directly, with a given cost (equal to the distance between the pickup and the delivery), or it can be routed through a transfer point. In the latter case, a cost is associated with an arc between the pickup point and the transfer point and with the arc between the transfer point and the delivery point. If an arc between two nodes is already used, all requests that have these nodes in common can use it at cost zero. This problem is solved to determine which transfer points the unplanned requests should use. In our

problem, every pickup and delivery node are distinct. Therefore, we cannot apply this approach. In the first step, clusters of pickups and clusters of deliveries are built and considered as single nodes. In the existing routes, we consider that pickups and deliveries that are serviced in a row form a cluster. Once every node of the current solution is assigned to a cluster, we add each node of unplanned requests to their closest cluster. We solve the k-star hub problem where some arcs are already used (according to the destroyed solution). The cost of an arc between two clusters is set to the smallest distance between two nodes of these clusters. For each unplanned request, the transfer point to consider during the reconstruction is the one, if any, used for this request in the optimal solution of the k-star hub problem.

5. The last method is based on the historical performance of the transfer points. For each request i and each transfer point t, we define $h_{i,t}$ as the cost of the best-known solution where i is transferred through t. If the transfer point t has never been used by request i, then we set $h_{i,t} = \infty$. For each request, a roulette wheel is used to select the transfer point to be considered. The weight associated with a transfer point t for a request i is $\min_{t' \in T} h_{i,t'}/h_{i,t}$.

In practice, we considered transfer selection in inserting heuristics with $m = 1$. For each inserting heuristic with transfer, one neighborhood is created for each of the five transfer selection methods, plus one neighborhood that considers all transfer points. The adaptive aspect of the ALNS gives priority to the most efficient neighborhoods.

4.4. Neighborhood Evaluation

The consideration of transfer points raises new difficulties, for example, checking if a modification of the solution (e.g., inserting or removing a request from a route) is feasible or not. Inserting a request may impact on the feasibility of the time window constraints on more than one route. This is highlighted in Figure 3.

The insertion of p_1 in the dashed route can cause a delay on the dotted route, propagated up to t_2, and make the insertion of d_1 unfeasible. The feasibility of an insertion can be checked by computing the new service times at nodes that follow the insertion position. However, it is computationally expensive. We consider an extension of forward time slacks (Savelsbergh 1985) based on a square matrix Ω containing the sum of waiting times between each connected point of the problem. Each time a request is removed or inserted, this matrix has to be updated. This update has a worst-case complexity of $O(|V|^2)$. Each time a request is inserted or removed, the latest possible service time at each inserted node has to be updated. However, using Ω, an insertion can be checked in constant time. Let us consider the insertion of request i. Let us denote $p(i) + 1$ the node following the insertion position of $p(i)$, $d(i) - 1$ the node preceding the insertion position of $d(i)$, and $d(i) + 1$ the node following the insertion position of $d(i)$. The delay at a node j_2 generated by the insertion of a node j_1 is denoted δ_{j_1, j_2}. First, we check that inserting $p(i)$ does not violate the latest possible service time at $p(i) + 1$. Secondly, we compute the new service time at $d(i) - 1$, which is equal to its current service time plus $\max(0, \delta_{p(i), p(i)+1} - \Omega_{p(i)+1, d(i)-1})$. Hence, considering the delay at $d(i) - 1$ when inserting $p(i)$, the service time at $d(i)$ and $d(i) + 1$ can be computed in constant time.

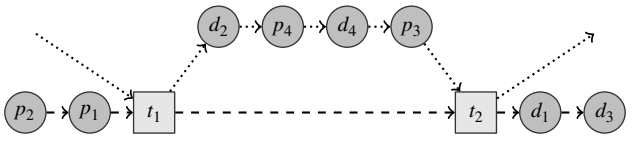

Figure 3 **Difficulties in Evaluating the Feasibility of an Insertion**

5. Computational Experiments

The method was coded in C++ and experiments were run on an i3-540 computer operated by Windows. We first discuss ALNS settings and use real-life data to evaluate the benefits of routing with transfer points. Considering the large run time obtained for the ALNS, the impact of reducing the number of iterations is then studied. Finally, the proposed ALNS is evaluated on related benchmark instances from the literature. For some of these generated instances, a lower bound on the savings achieved thanks to the use of transfer is computed.

5.1. Instances

We evaluate the algorithm on a set of 10 real-life instances arising from distinct sources: specialized centers for disabled children, vocational rehabilitation centers for adults, and schools receiving both able-bodied and disabled children. In the first two sources, each delivery location is a common destination for dozens of requests. The instances considered correspond to a few centers that wish to pool the organization of their transport. In the following, the names of these instances start with "centers." In the case of schools for disabled children, each school is the destination for a small number of people. Thus, the instances generally include all the schools of a given geographical area, possibly several dozen. The names of these instances start with "schools."

The considered instances include between 55 and 193 requests. They only consider outbound trips from home to the centers. Thus, the pickup points are the personal addresses of the transported people and the delivery and transfer points are the centers or schools.

All requests have distinct pickup points and correspond to individual demands ($q_i = 1$). The number of delivery points is comprised between two and five for specialized schools or vocational rehabilitation centers and between 13 and 33 for all other schools. All delivery points can be used as transfer points.

Instances are named "category-$|P|$-$|D|$" such that the number of requests and delivery locations is reflected in their name. For example, instance centers-193-5 is of the type "centers" and consists of 193 requests and 5 delivery locations.

The delivery points may have different opening hours. The time window $[e_i, l_i]$ for vehicle arrival is defined such that l_i corresponds to the beginning of the daily activity. The time window opening e_i is set at 15 to 30 minutes prior to l_i, depending on the center. We assume that the vehicles start from a dummy location located at distance zero from every pickup location. The traveling time matrices between locations are asymmetric and satisfy the triangular inequality.

We also use the instance set proposed by Mitrović-Minić and Laporte (2006). The 4,320 instances contain either 50 or 100 requests and are divided into 144 classes. The average results are given by Mitrović-Minić and Laporte (2006) for 64 classes of instances. The nodes of the problems are located in a 60 km × 60 km square. A constant speed of 60 km/h is considered for the vehicles. The Manhattan distance is used. A 10-hour planning horizon is considered and the capacity of the vehicles is not binding. Several types of node distribution are considered. "*Uniform*" represents instances where the nodes are positioned randomly in the square and "*Border* $(a \times a)$" represents instances where the nodes are located in four clusters of side length a located near the middle of each side of the 60 km × 60 km square. Finally, "*Random* $(a \times a)$" represents instances where the nodes are located in four clusters of side length a located randomly in the square. Various distributions of the time windows are considered; "10" represents the case where the requests have no time windows, "2–4–8" represents the case where 30% of the requests are two-hour requests, 50 % of the requests are four-hour requests and 20% of the requests are eight-hour requests.

Following Mitrović-Minić and Laporte (2006), we tested several configurations of transfer points layout. Type T0 means that no transfer point is considered. In *Uniform* instances, T4 means that four transfer points are located at the positions (20, 20), (40, 20), (20, 40), (40, 40). In *Border* instances, the scheme T2 designates the case with a single transfer point located in the middle of the area. In *Random* instances, T2 designates the case with four transfer points located at the intersection of parallel horizontal and vertical lines drawn from the centers of the clusters. Finally, for all cluster instances, the scheme T3 considers the transfer points of scheme T2 plus four transfer points located at the center of the clusters. The service time is considered to be five minutes at each pickup and delivery node and zero minutes at the transfer points.

5.2. ALNS Parameter Settings

For most of the parameters of the ALNS algorithm, we validate the values proposed by Ropke and Pisinger (2006). It turns out that for the PDPT, the percentage r of requests removed in each iteration can be reduced to improve the method. We investigated four settings where r is chosen in the intervals [10, 50], [10, 35], [10, 20], and [5, 10], respectively, and run the ALNS for 25,000 iterations. The results obtained for five representative instances are summarized in Table 1.

Unsurprisingly, when the number of requests removed decreases, the run time is reduced. We also observe that better or equivalent costs are obtained when the percentage r is taken in the interval [10, 20]. The counterpart is a slight increase in the standard deviation. Thus, in the following tests, the percentage r is taken in the interval [10, 20].

5.3. Evaluation of the Benefits of Transfers

Each instance is solved five times during 25,000 iterations. We consider that the number of vehicles is not limited and minimize the total distance traveled by the vehicles. The objective is to compare the cost of

Table 1 Impact of the Percentage of Requests Removed

Instances	% Req. removed	Avg.	Min.	Std. dev.	Time (sec.)
Centers-84-2	[10, 50]	778.61	773.15	3.16	3,187
	[10, 35]	775.52	767.20	6.64	1,766
	[10, 20]	**769.83**	**762.15**	6.10	1,167
	[5, 10]	777.30	769.38	5.14	755
Centers-109-2	[10, 50]	620.98	617.82	4.68	18,297
	[10, 35]	615.76	609.98	3.68	10,693
	[10, 20]	**610.74**	**606.85**	3.36	5,418
	[5, 10]	620.97	616.73	4.27	2,328
Centers-193-5	[10, 50]	1,159.10	1,151.90	6.85	123,434
	[10, 35]	1,160.20	1,151.55	6.98	59,309
	[10, 20]	**1,152.44**	**1,134.90**	10.40	37,617
	[5, 10]	1,153.69	1,142.27	7.20	16,085
Schools-66-13	[10, 50]	800.51	800.27	0.33	10,379
	[10, 35]	800.26	799.05	0.74	5,504
	[10, 20]	800.51	800.27	0.33	2,538
	[5, 10]	**798.60**	**792.82**	4.55	1,060
Schools-84-21	[10, 50]	**1,053.64**	1,051.32	2.35	24,690
	[10, 35]	1,054.15	**1,050.42**	2.57	10,839
	[10, 20]	1,055.67	1,052.17	3.43	4,818
	[5, 10]	1,057.41	1,056.03	1.47	2,513

Notes. The first column designates the instance. The second column indicates the interval for r. Columns three to five report the average and minimum values of the objective function and the standard deviation found over five runs, respectively. The last column reports the average run time in seconds.

Table 2 Real-Life Instances: Minimization of the Distance Traveled—ALNS 25,000 Iterations

Instances	Transfer points	Avg.	Min.	Std. dev.	Time (sec.)	Gap (%)
Centers-84-2	0	782.04	782.03	0.02	157	2.54
	2	769.83	762.15	6.10	1,167	
Centers-81-2	0	453.09	452.72	0.56	205	3.23
	2	438.64	438.08	0.58	3,599	
Centers-87-2	0	478.06	477.25	0.77	206	2.96
	2	463.92	463.12	0.48	4,215	
Centers-109-2	0	650.20	647.88	2.22	280	6.33
	2	610.74	606.85	3.36	5,418	
Centers-193-5	0	1,253.63	1,250.30	3.48	750	9.23
	5	1,152.44	1,134.90	10.40	37,617	
Schools-55-16	0	654.72	654.72	0.00	135	0.75
	16	649.78	649.78	0.00	413	
Schools-66-13	0	824.82	824.82	0.00	159	2.98
	13	800.51	800.27	0.33	2,538	
Schools-84-21	0	1,121.37	1,121.37	0.00	215	6.17
	21	1,055.67	1,052.17	3.43	4,818	
Schools-84-33	0	1,239.65	1,239.32	0.30	209	6.23
	33	1,164.48	1,162.08	2.22	6,239	
Schools-106-24	0	978.64	978.15	0.73	285	4.12
	24	940.02	937.85	2.95	9,867	

Notes. The first column contains the instance names. The second column indicates the number of transfer points considered. Columns three to five report the average and best results and the standard deviation over five runs, respectively. The sixth column reports the average run time and the last column shows the gap between the best-known solutions with and without transfer points.

optimized solutions for the PDPTW (using the ALNS algorithm described in §3) with the best costs found for the PDPT. Table 2 summarizes the results on real-life instances.

The table shows that the savings because of transfers can vary a lot from one instance to another. In practice, it is observed that transfers are useful when several people from the same geographical area make transportation requests to distinct destinations.

Table 3 presents the savings achieved on Mitrović-Minić and Laporte (2006) instances. We compare solutions obtained using the ALNS heuristics on instances with various patterns of transfer points.

The gap found by the ALNS between solutions using or not using transfer points is similar to the gap reported by Mitrović-Minić and Laporte (2006) for each class of instances.

To evaluate the minimal savings achieved by transfers, we wish to compare upper bounds of the PDPT found by the ALNS algorithm with optimal solutions or lower bounds for the PDP (without transfer). Given one instance, let us denote UB_T the best upper bound for the PDPT and LB a lower bound for the PDP. The savings because of the use of transfer points are greater than or equal to $gap_T = (LB - UB_T)/UB_T$. We were able to compute LB with a column generation approach on instances with a uniform distribution of the nodes and time windows with the pattern "2–4–8" (see Table 3). For this class of instances, the average value of δ_T is 5.08%. In 60% of the cases, gap_T is greater than 5%. Let us also denote UB an upper bound found for the PDP and $gap^* = (UB - LB)/LB$ the optimality gap. The value of gap^* is 1.1 on average and is below 2% in 90% of the cases. The minimal, average and maximal observed values of these gaps are reported in Table 4.

Note that we also performed some tests on PDPTW instances with known optimal solutions. The instances of Li and Lim (2002) are constructed in such a way that transfers did not produce an interesting

Table 3 Evaluation of Savings Achieved Using Transfers on Mitrović-Minić and Laporte (2006) Instances

Node distribution	TW	Transfer points	Cost	Gap (%)
Uniform	10	T0	800.29	
		T4	738.02	7.78
	2–4–8	T0	1,217.03	
		T4	1,145.47	5.88
Border (6 × 6)	10	T0	439.68	
		T2	366.26	16.70
		T3	364.68	17.06
	2–4–8	T0	956.47	
		T2	696.46	27.18
		T3	686.60	28.22
Border (10 × 10)	10	T0	479.49	
		T2	439.11	8.42
		T3	432.23	9.86
	2–4–8	T0	972.54	
		T2	767.57	21.08
		T3	753.20	22.55
Border (20 × 20)	10	T0	593.28	
		T2	584.61	1.46
		T3	555.75	6.32
	2–4–8	T0	991.22	
		T2	905.53	8.65
		T3	882.51	10.97
Random (6 × 6)	10	T0	323.31	
		T2	297.84	7.88
	2–4–8	T0	665.03	
		T2	568.74	14.48

Notes. The first three columns designate the node distribution, the time windows distribution, and the transfer points configuration, respectively. The fourth column reports the cost found by the ALNS. The fifth column shows the gap with the solution without transfers (T0).

Table 4 Comparison Between Solution With and Without Transfers

	Min (%)	Average (%)	Max (%)
$gap_T = \frac{LB - UB_T}{UB_T}$	2.05	5.08	8.70
$gap^* = \frac{UB - LB}{LB}$	0.00	1.10	2.44

output. The instances of Ropke and Cordeau (2009) have very tight capacity constraints. Therefore only few requests can share vehicles, which does not allow much consolidation of the flows at transfer points. Thus transfers did not yield significant savings on these instances either.

In conclusion, the comparison based on Mitrović-Minić and Laporte (2006) instances shows that introducing transfer points can result in savings of up to 28%. On instances where a lower bound on the cost without transfer has been computed, the average savings achieved thanks to the use of the transfer is larger than 5.08%.

5.4. Run-Time Reduction

The experiments of §5.3 show that the time needed to perform 25,000 iterations is prohibitive for some instances. This is due to the time needed to explore some neighborhoods considering transfer. Thus, we choose to decrease the number of iterations. Four settings with 5,000 and 10,000 iterations and various values for r are compared with the 25,000 iteration runs. Because the cooling schedule of the simulated annealing depends on the number of iterations, the runs with 5,000, 10,000, and 25,000 iterations are performed separately. Table 5 summarizes the results.

When the number of iterations performed is smaller, the results of the algorithm are less stable and it has more difficulty in finding very good solutions. It can also be noted that, even if the running times are equivalent, it is more useful to perform 5,000 iterations with $r \in [10, 20]$ than 10,000 with $r \in [5, 10]$. However, the difference between the 5,000 iteration runs and the 25,000 iteration runs is not dramatic, on average 0.68%. From this perspective, performing 5,000 iterations may provide a good trade-off between the quality of the solution and the running time. Preliminary tests with a faster version of the ALNS running for 30 seconds provide results that compete with those of Mitrović-Minić and Laporte (2006).

5.5. Evaluation on Related Instances

We evaluate the ALNS on 26 representative classes grouping 780 instances from Mitrović-Minić and Laporte (2006). For each class of instances, the results presented in Table 6 are average values over the 30 instances of the class. Concerning the ALNS, each instance is executed five times and the cost of the best solution is reported. The ALNS is run over 25,000 iterations with $r \in [10, 20]$.

These tests show that the ALNS method presented in this article is better overall than the local search on the instances with loose time windows (the TW 10 instances). However, unstable results are observed. We assume that the use of the Manhattan distance makes these instances difficult. In the presence of harder time windows (TW 2–4–8 instances), the ALNS clearly outperforms the local search. As the ALNS is designed to solve problems with quite hard time windows, this result is not too surprising.

Table 5 Impact of the Number of Iterations

Instances	No. of iter.	% Req.	Avg.	Min.	Time (sec.)
Centers-84-2	5,000	[10, 20]	780.63	767.27	205
	10,000	[10, 20]	776.44	765.60	482
	10,000	[5, 10]	784.73	780.83	244
	25,000	[10, 20]	769.83	762.15	1,167
Centers-81-2	5,000	[10, 20]	442.64	437.13	599
	10,000	[10, 20]	441.71	440.78	1,333
	10,000	[5, 10]	442.03	440.12	872
	25,000	[10, 20]	438.64	438.08	3,599
Centers-87-2	5,000	[10, 20]	466.68	463.08	660
	10,000	[10, 20]	467.11	463.77	1,651
	10,000	[5, 10]	466.85	464.33	684
	25,000	[10, 20]	463.92	463.12	4,215
Centers-109-2	5,000	[10, 20]	620.05	614.52	1,055
	10,000	[10, 20]	618.51	615.27	2,047
	10,000	[5, 10]	626.98	617.82	833
	25,000	[10, 20]	615.96	609.27	5,804
Centers-193-5	5,000	[10, 20]	1,163.06	1,145.68	7,447
	10,000	[10, 20]	1,166.82	1,165.53	12,635
	10,000	[5, 10]	1,157.47	1,141.80	6,763
	25,000	[10, 20]	1,152.44	1,134.90	37,617
Schools-55-16	5,000	[10, 20]	649.78	649.78	413
	10,000	[10, 20]	649.78	649.78	864
	10,000	[5, 10]	649.87	649.78	931
	25,000	[10, 20]	649.78	649.78	2,190
Schools-66-13	5,000	[10, 20]	800.39	800.27	472
	10,000	[10, 20]	799.66	798.45	932
	10,000	[5, 10]	801.65	800.87	414
	25,000	[10, 20]	800.51	800.27	2,538
Schools-84-21	5,000	[10, 20]	1,060.36	1,057.78	792
	10,000	[10, 20]	1,055.60	1,052.32	1,754
	10,000	[5, 10]	1,059.10	1,055.27	1,160
	25,000	[10, 20]	1,055.67	1,052.17	4,818
Schools-84-33	5,000	[10, 20]	1,168.61	1,163.35	1,846
	10,000	[10, 20]	1,166.75	1,165.32	3,492
	10,000	[5, 10]	1,168.37	1,164.18	1,291
	25,000	[10, 20]	1,164.48	1,162.08	6,239
Schools-106-24	5,000	[10, 20]	945.41	940.82	2,001
	10,000	[10, 20]	943.01	939.02	4,156
	10,000	[5, 10]	944.21	941.1	1,836
	25,000	[10, 20]	940.02	937.85	9867
Average	5,000	[10, 20]	809.76	803.97	1,549
	10,000	[10, 20]	808.54	805.58	2,935
	10,000	[5, 10]	810.13	805.61	1,503
	25,000	[10, 20]	805.13	800.97	7,805

Notes. The second column indicates the number of iterations performed. The third column is the interval in which r is chosen. The fourth and fifth columns are the average and minimum values found over five runs, respectively. The last column reports the average run time in seconds.

6. Summary and Further Research

In this article, we proposed an adaptive large neighborhood search for solving the pickup and delivery

Table 6 Comparison with the Local Search of Mitrović-Minić and Laporte (2006)

Node distribution	TW	Transfer points	M-M & L	ALNS obj.	ALNS time (sec.)
Uniform	10	T0	845.86	**800.29** (5.39%)	122
		T4	791.53	**738.02** (6.76%)	1,423
	2–4–8	T0	1,324.63	**1,217.03** (8.12%)	34
		T4	1,297.84	**1,145.47** (11.74%)	328
Border (6×6)	10	T0	473.46	**439.68** (7.13%)	592
		T2	**360.47**	366.26 (−1.61%)	1,490
		T3	409.97	**364.68** (13.41%)	2,566
	2–4–8	T0	1,093.38	**956.47** (12.52%)	39
		T2	809.61	**696.46** (13.98%)	869
		T3	811.35	**686.60** (15.38%)	1,098
Border (10×10)	10	T0	526.66	**479.49** (8.96%)	560
		T2	440.86	**439.11** (0.40%)	1,350
		T3	526.18	**432.23** (17.86%)	2,714
	2–4–8	T0	1,120.78	**972.54** (13.23%)	37
		T2	892.38	**767.57** (13.99%)	846
		T3	912.47	**753.20** (17.46%)	1,070
Border (20×20)	10	T0	657.67	**593.28** (9.79%)	389
		T2	633.91	**584.61** (7.78%)	668
		T3	647.07	**555.75** (14.11%)	2,340
	2–4–8	T0	1,137.78	**991.22** (12.88%)	38
		T2	1,053.25	**905.53** (14.03%)	591
		T3	1,073.21	**882.51** (17.77%)	787
Random (6×6)	10	T0	**321.84**	323.31 (−0.46%)	927
		T2	311.39	**297.84** (4.35%)	2,596
	2–4–8	T0	777.98	**665.03** (14.52%)	44
		T2	726.34	**568.74** (21.70%)	789
Average	10	All	534.37	**493.43** (7.04%)	1,364
	2–4–8	All	1,002.38	**862.18** (14.41%)	505

Notes. The first three columns designate the node distribution, the time windows distribution, and the transfer points configuration, respectively. The fourth column indicates the cost found by Mitrović-Minić and Laporte (2006), denoted M-M & L. The fifth column reports the cost found by the ALNS, as well as the gap with column M-M & L. The last column is the average run time of the ALNS in seconds.

problem with transfers. Our approach relies on the ALNS algorithm described by Pisinger and Ropke (2007), which implementation is evaluated on the instances proposed by Li and Lim (2002).

The algorithm is then extended to handle transfer points. This requires introducing new destruction and repairing heuristics dedicated to the use of transfer points. The algorithm improves the results of Mitrović-Minić and Laporte (2006) for the PDPT. We also apply the method to real-life instances concerning the transportation of mentally or physically disabled people. In these cases, the introduction of transfer points generally brings significant improvements (up to 9%), which differ greatly from one geographical area to another.

This work suggests several improvements and further research perspectives. Solving the PDPT clearly shows the potential benefits of using transfer points but requires much more computation time than solving the PDPTW. Most of the time is spent on the calculation of insertion positions. Indeed, every insertion of a request in a route with a transfer point can impact the passengers not only along this route but also those along adjacent routes (routes that include this transfer point). The feasibility check is then more demanding and leads to a long computation time. From a strategic point of view, the decision makers are likely to accept long running times provided the heuristic proposes very good solutions. On the other hand, in an operational context, the users would like to test various scenarios (fleet configurations, special days with various demand profile, etc.). This requires faster computation. As the ALNS relies on an adaptive mechanism, it needs to run during a large enough number of iterations. Providing a method capable of quickly giving good results for the PDPT is then an important practical issue.

An important criterion for measuring the quality of service to passengers is the ride time, i.e., the time spent by each passenger in the vehicles. Associating each request with a maximum ride time transforms the problem into a dial-a-ride-problem (DARP). Applying the ALNS to the DARP with transfers leads to the additional difficulty of checking the ride-time constraints when repairing a destroyed solution. Another challenge is to compute tight lower bounds for the PDPT and to develop exact methods (branch and price) for solving instances of the PDPT of realistic size.

References

Berbeglia G, Cordeau J-F, Laporte G (2010) Dynamic pickup and delivery problems. *Eur. J. Oper. Res.* 202(1):8–15.

Berbeglia G, Cordeau J-F, Gribkovskaia I, Laporte G (2007) Static pickup and delivery problems: A classification scheme and survey. *TOP* 15(1):1–31.

Blasum U, Hochstattler W, Oertel P, Woeginger G (2007) Steiner diagrams and *k*-star hubs. *J. Discrete Algorithms* 5(3):622–634.

Cortés CE, Jayakrishnan R (2002) Design and operational concepts of high-coverage point-to-point transit system. *Transportation Res. Record* 1783:178–187.

Cortés CE, Matamala M, Contardo C (2010) The pickup and delivery problem with transfers: Formulation and a branch-and-cut solution method. *Eur. J. Oper. Res.* 200(3):711–724.

Fugenschuh A (2009) Solving a school bus scheduling problem with integer programming. *Eur. J. Oper. Res.* 193(3):867–884.

Gørtz IL, Nagarajan V, Ravi R (2009) Minimum makespan multi-vehicle dial-a-ride. Fiat A, Sanders P, eds. *Algorithms—ESA 2009: 17th Annual European Symposium*, Lecture Notes in Computer Science, Vol. 5757 (Springer-Verlag, Berlin, Heidelberg), 540–552.

Kerivin HLM, Lacroix M, Mahjoub AR, Quilliot A (2008) The splittable pickup and delivery problem with reloads. *Eur. J. Indust. Engrg.* 2(2):112–133.

Kirkpatrick S, Gelatt CD, Vecchi MP (1983) Optimization by simulated annealing. *Science* 220(4598):671–680.

Lapierre SD, Ruiz AB, Soriano P (2004) Designing distribution networks: Formulations and solution heuristic. *Transportation Sci.* 38(2):174–187.

Laporte G, Musmanno R, Vocaturo F (2010) An adaptive large neighbourhood search heuristic for the capacitated arc-routing problem with stochastic demands. *Transportation Sci.* 44(1):125–135.

Li H, Lim A (2002) A metaheuristic for the pickup and delivery problem with time windows. Bilof R, Palagi L, eds. *Proc. 13th Internat. Conf. Tools with Artificial Intelligence, ICTAI-2001* (IEEE, Dallas), 160–167.

Lin CKY (2008) A cooperative strategy for a vehicle routing problem with pickup and delivery time windows. *Comput. Indust. Engrg.* 55(4):766–782.

Masson R, Lehuédé F, Péton O (2011) A tabu search algorithm for the dial-a-ride problem with transfers. Benyoucef L, Trentesaux D, Artiba A, Rezg N, eds. *Proc. Internat. Conf. Indust. Engrg.* (Systems Management, Metz, France), 1224–1232.

Mitrović-Minić S, Laporte G (2006) The pickup and delivery problem with time windows and transshipment. *INFOR* 44(3):217–228.

Mues C, Pickl S (2005) Transshipment and time windows in vehicle routing. Bein W, Chin FYL, Hsu DF, Palis ML, eds. *ISPAN'05: Proc. 8th Internat. Sympos. Parallel Architectures, Algorithms and Networks* (IEEE Computer Society, Washington, DC), 113–119.

Nakao Y, Nagamochi H (2010) Worst case analysis for pickup and delivery problems with transfer. *IEICE Trans. Fundamentals of Electronics, Comm. Comput. Sci.* E91-A(9):2328–2334.

Oertel P (2000) Routing with reloads. Ph.D. thesis, University of Cologne, Germany.

Parragh SN, Doerner KF, Hartl RF (2008) A survey on pickup and delivery problems, part 2: Transportation between pickup and delivery locations. *J. für Betriebswirtschaft* 58(2):81–117.

Petersen H, Ropke S (2011) The pickup and delivery problem with cross-docking opportunity. Böse J, Hu H, Jahn C, Shi X, Stahlbock R, Voß S, eds. *Computational Logistics*, Lecture Notes in Computer Science, Vol. 6971 (Springer, Berlin/Heidelberg), 101–113.

Pisinger D, Ropke S (2007) A general heuristic for vehicle routing problems. *Comput. Oper. Res.* 34(8):2403–2435.

Potvin J-Y, Rousseau J-M (1993) A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. *Eur. J. Oper. Res.* 66(3):331–340.

Ropke S, Cordeau J-F (2009) Branch and cut and price for the pickup and delivery problem with time windows. *Transportation Sci.* 43(3):267–286.

Ropke S, Pisinger D (2006) An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Sci.* 40(4):455–472.

Ropke S, Pisinger D (2010) Large neighborhood search. Gendreau M, Potvin J-Y, eds. *Handbook of Metaheuristics*, 2nd ed. (Springer, New York), 399–419.

Savelsbergh MWP (1985) Local search in routing problems with time windows. *Ann. Oper. Res.* 4(1):285–305.

Schrimpf G, Schneider J, Stamm-Wilbrandt H, Dueck G (2000) Record breaking optimization results using the ruin and recreate principle. *J. Comput. Physics* 159:139–171.

Shang JS, Cuff CK (1996) Multicriteria pickup and delivery problem with transfer opportunity. *Comput. Indust. Engrg.* 30(4): 631–645.

Shaw P (1998) Using constraint programming and local search methods to solve vehicle routing problems. Maher M, Puget JF, eds. *Proc. 4th Internat. Conf. Principles and Practice of Constraint Programming*, (Springer, Berlin, Heidelberg), 417–431.

Thangiah SR, Fergany A, Awam S (2007) Real-time split-delivery pickup and delivery time window problems with transfers. *Central Eur. J. Oper. Res.* 15(4):329–349.