# Constructing Initial Solutions for the Multiple Vehicle Pickup and Delivery Problem with Time Windows

Manar I. Hosny, Christine L. Mumford
mifawzi@ksu.edu.sa, C.L.Mumford@cs.cardiff.ac.uk

**Abstract**

The Multiple Vehicle Pickup and Delivery Problem with Time Windows (MV-PDPTW) is an important problem in logistics and transportation. However, this problem is characterized by having a large number of constraints that are difficult to deal with in a solution algorithm. Indeed, merely constructing a feasible solution to this hard problem is a challenge in itself. In this research, we compare several construction algorithms that generate initial feasible solutions to the problem. The suggested algorithms all utilize a simple routing heuristic to create individual vehicle routes. The algorithms differ, though, in whether routes are generated sequentially or in parallel. They also have different criteria for selecting requests and the routes in which they will be inserted. Inserting a request in a route is either based on a *first acceptance* criterion, in which a request is inserted in the first route where a feasible insertion is found, or a *best acceptance* criterion, in which a request is inserted in the estimated best route for insertion. Experimental results on several benchmark problem instances indicate that the sequential construction heuristic may be the most suitable construction algorithm for this problem, in terms of simplicity of coding, solution quality as well as processing speed [1].

**Keywords:** Combinatorial Optimization. Pickup and Delivery. Vehicle Routing. Heuristics. Meta-heuristics. Construction Algorithms

## 1 Introduction

The Multiple Vehicle Pickup and Delivery Problem with Time Windows (MV-PDPTW) is a variant of the well-known Vehicle Routing Problem with Time Windows (VRPTW). The problem deals with a number of customer requests that are to be served by a fleet of vehicles, while a number of

[1] This paper is part of the PhD thesis of the first author [8], and it is an expanded version of the *MIC2009* conference paper [6].

constraints must be observed. Each vehicle has a limited capacity (the capacity constraint). A vehicle route usually starts and ends at a central depot. A request must be picked up from a pickup location to be delivered to a corresponding delivery location. Naturally, the pickup and delivery pair must be served by the same vehicle (the coupling constraint) and the pickup must precede the delivery (the precedence constraint). In addition, every request must be served within a predetermined time window interval (the time window constraint). If the vehicle arrives earlier than the allowed service time, it should wait until the beginning of the specified period. A solution to the problem should assign requests to vehicles and find a route for each vehicle, such that the total service cost is minimized and all problem constraints (coupling, precedence, capacity and time windows) are adhered with.

Possible practical applications of the MV-PDPTW include: transportation of raw materials from suppliers to factories, Internet-based pickup from sellers and delivery to buyers, pickup and delivery of charitable donations from homes to different organizations, and the transport of medical samples from medical offices to laboratories. In addition, an important related variant is the dial-a-ride problem, where people instead of goods are transported.

As a generalization of the travelling salesman problem, the MV-PDPTW is known to be *NP-hard* [16], and the presence of many constraints makes the problem particularly complicated. Exact algorithms are too slow for large problem sizes. In addition, generating feasible and good quality solutions to the problem in a reasonable amount of time is often a hard challenge for researchers. The MV-PDPTW is both a ***grouping problem*** (assigning requests to vehicles), and a ***routing problem*** (finding the best route for each vehicle). Thus, an intelligent solution methodology should be able to handle these two aspects efficiently. Researchers in the area usually try to solve the problem in two stages: the first stage constructs one or more initial solutions to the problem, while the second stage tries to improve these solutions using a heuristic or a meta-heuristic approach.

To construct a solution for the MV-PDPTW, each step of the algorithm usually selects an un-assigned request whose insertion is predicted to cause the least increase in the overall cost of the solution. The selected request is then inserted in its best (least cost) feasible insertion position found among all available routes. This kind of insertion may require complicated calculations to estimate the effect of the insertion, in terms of the increase in travel distance and time delay, on all requests already existing in the route who could be affected by the insertion. Additional decisions during the construction of the solution include whether to build routes sequentially or in parallel, and how to order requests prior to the insertion process.

While these considerations also apply to the general VRPTW, where all requests are of the same type (either pickups or deliveries), the pickup and

delivery problem in itself entails additional considerations. This is due to the presence of a pair of related locations for each individual request and the precedence and coupling issues resulting thereof. For example, the decision regarding the best insertion position for a certain request should ideally take both the pickup and the delivery into account. The sorting criteria for requests, prior to insertion, may likewise be based on either the pickup or the delivery location, or perhaps combine both. It is also frequently the case with the MV-PDPTW that the initial solution is drastically changed during the improvement phase. For example [1] and [16] reported very good results using an algorithm that is based on a Large Neighbourhood Search (LNS). The algorithm removes and then relocates a large number of requests (30% - 40%) in each iteration. This could possibly indicate that sophisticated construction algorithms, that are usually time consuming, parameter dependent, and hard to implement, may not actually warrant their cost, as opposed to more straightforward and faster algorithms.

In an attempt to overcome the difficulties inherent in the construction of a feasible solution, which are mainly due to the hard problem constraints and the complex problem-specific decisions, we propose in this paper four different construction heuristics that aim to build initial feasible solutions to the MV-PDPTW. All our algorithms utilize a simple and efficient routing algorithm to generate feasible individual vehicle routes. These algorithms, nevertheless, differ in whether the construction of vehicle routes is performed sequentially or in parallel. They also differ in the criteria according to which the next un-routed request is selected for insertion in a particular route. The aim of the research is to decide which construction algorithm has more potential as a preliminary step towards a complete solution methodology to the problem. A promising construction algorithm should demonstrate a suitable balance between quality of the generated solution, processing speed and simplicity of implementation. In order to evaluate the suggested algorithms, we have tested them on several benchmark problem instances and the experimental results are reported in this paper.

The rest of the paper is organized as follows: Sect. 2 formally defines the MV-PDPTW. Section 3 summarizes some related work. Section 4 explains the routing algorithm embedded within the different construction heuristics used. Section 5 details the construction heuristics suggested in this research. Section 6 reports the experimental results of the algorithms tested. Section 7 sheds light on some implementation issues and complexity analysis of the suggested algorithms Finally, Sect. 8 concludes with a brief summary of the research.

## 2 The Multiple Vehicle Pickup and Delivery Problem with Time Windows

Let $G = (N, A)$ be a digraph. The node set is $N = \{n_i \in N | i = 0, 1, 2, ..., m\}$, such that $m$ is an even index. The node $n_0$ denotes the depot, and each $n_i, i = 1, 2, ...m$ denotes a customer location. Since for each customer request we have a pair of pickup and delivery locations, we can assume, without loss of generality, that the set $N^+ = \{n_i \in N | i = 1, 2, ..., m/2\}$ represents pickup locations, and the set $N^- = \{n_i \in N | i = (m/2) + 1, ..., m\}$ represents delivery locations, such that the pickup location $n_i$ has the corresponding delivery location $n_{i+(m/2)}$. Thus, $N = N^+ \cup N^-$ and $|N^+| = |N^-| = m/2$.

Each location $n_i$ is associated with:

- A customer demand $q_i$, such that $q_i > 0$ for a pickup location, $q_i < 0$ for a delivery location and $q_i + q_j = 0$ for the same customer's pickup and delivery locations ($q_0 = 0$)
- A service time $s_i$ ($s_0 = 0$), which is the time needed to load or unload a customer demand
- A time window $[e_i, l_i]$ during which the location must be served, and $l_i \geq e_i$

For each pair of nodes $< n_i, n_j >$ a travel time $t_{ij}$ and/or a travel distance $d_{ij}$ are specified. Only edges satisfying the time window constraint are allowed. Thus the arc set is $A = \{< n_i, n_j > | n_i, n_j \in N, n_i \neq n_j, t_{0i} + s_i + t_{ij} < l_j\}$.

Each vehicle has a limited capacity $C$. We assume a homogeneous fleet of vehicles, where all vehicles have the same capacity. The capacity constraint ensures that the total load carried by each vehicle at any given time does not exceed its capacity.

A vehicle's journey should start and end at the depot, while each location should be visited exactly once. In addition, a location must be serviced within the specified time window (TW), i.e., if the vehicle reaches the location before the earliest service time $e_i$, it must wait until $e_i$. The precedence constraint requires that each pickup location must precede the corresponding delivery location, while the coupling constraint requires that the same customers's pickup and delivery locations must be served by the same vehicle.

The objective function varies depending on the application. In general, one or more of the following objectives are minimized: the number of vehicles used, the total travelling distance, and the total schedule duration.

## 3 Related Work

Solution construction can be done either sequentially or in parallel. A sequential construction builds routes one after another, while a parallel construction builds a number of routes simultaneously. To construct initial solutions for the MV-PDPTW sequentially, researchers usually adapted Solomon's sequential insertion heuristics of the VRPTW [19]. A weighted sum of the extra travel distance and total time delay resulting from the insertion is often used to estimate the cost of the insertion. This type of construction was used by [10] for the MV-PDPTW, and was followed by a solution improvement phase called a tabu-embedded simulated annealing.

A parallel construction heuristic, on the other hand, was first introduced in [15] for the VRPTW. In a parallel construction, several routes are initialized with seed customers and requests are subsequently inserted into any of the initialized routes. Accordingly, the algorithm needs an initial estimate of the number of vehicles to be used. Routes are later added as needed if the initial estimate does not yield a feasible solution. The authors also introduced an additional complex measure in the cost function, which is a generalized regret value comparing the difference between the cost of an immediate insertion verses a postponed insertion. Customers with a large regret value must be considered first. This regret measure was also used by [16] for the MV-PDPTW, and was embedded within an Adaptive Large Neighbourhood Search (ALNS) technique to improve the solution quality.

The work in [11] presents a sequential construction algorithm for the MV-PDPTW. The algorithm repeats a cycle of three components. The first component is a constructor, which uses a sequential greedy algorithm to add pairs of customers in the order they appear in a priority sequence that is initially random. The analyser afterwards analyses the solution and assigns a certain 'blame' value for each customer based on its contribution to the total solution cost. Finally, the prioritizer reorders the customers, such that customers with a high blame value are moved forward in the priority sequence.

A parallel construction heuristic that solves the MV-PDPTW is presented in [12]. The initial set of routes is created by finding the largest set of customers, where it is impossible to serve any two customers with the same vehicle. Each initial route is then initialized with one customer from this set. To insert the remaining customers afterwards, the algorithm takes into consideration the effect of insertion on both the classical increase in distance measure, and also the remaining time window slack in the route, i.e., priority is given to insertions that do not use much of the available time slack, allowing for more feasible latter insertions. The authors also use a non-standard measure of the visual attractiveness of the route to select the most desired insertions.

An important survey of the general pickup and delivery problem and

approaches developed to handle it was presented in [17]. A more recent surveys is presented in [14]. A survey of the important related dial-a-ride problem is in [3].

We noticed during our literature survey that researchers who adopt a 2-phase approach (i.e, construction of an initial solution, followed by an improvement phase) to solving the problem often pay more attention to the solution improvement phase, such that the results of the initial solution construction phase are seldom reported. This makes it difficult to assess the contribution of the construction method to the success or the failure of the overall algorithm. It is also important to note that the role of the construction algorithm is not only limited to the initialization phase. The construction algorithm is often utilized at various stages during the improvement phase to create or modify new or partial solutions, as done for example in [1] and [13]. In this situation, a good choice of the construction algorithm is vitally important.

To the best of our knowledge, our research is the first attempt to compare different initial solution construction methods for the MV-PDPTW. The research will help identify the construction heuristic(s) that seems to be most appropriate for this problem, and decide whether sophisticated and computationally expensive methods actually perform a better job in constructing good quality initial solutions, as opposed to other simpler and less expensive algorithms. In the following section we explain our simple routing algorithm, which is the core of the different construction algorithms proposed in this research. Section 5 then discusses in detail these construction algorithms.

# 4 The Routing Algorithm

A crucial part of the MV-PDPTW is the routing algorithm that will generate a feasible route for each individual vehicle. A major concern is how to handle all problem constraints efficiently. Our routing algorithm, first introduced in [7], has proven very effective for solving the Single Vehicle PDPTW. This algorithm is based on an *iterative improvement* of individual routes, which is embedded in the overall constructive algorithm that could either be sequential or parallel. The main difference between our routing algorithm and other routing (insertion) heuristics in the literature is that our algorithm does not try to find the best insertion position for each request in the route, but accepts any feasible insertion. As a result, many complex calculations and problem-specific decisions, that are related to the association between the pickup and the delivery, can be avoided. For example, our algorithm eliminates the bias towards either the pickup or the delivery location, which is one of the major drawbacks of 'classical' insertion methods. Clearly, when the best insertion position for one request (pickup or delivery) is chosen first,

the choices available for its partner will be restricted accordingly.

Our routing algorithm adopts a simple route representation. Rather than representing the visiting order of requests by a one-dimensional permutation of all the different locations, we treat both the pickup location and its associated delivery as one unit. In other words, we assign the same code (number) to both the pickup and its delivery. We then rely on a simple decoder to always identify the first occurrence as the pickup and the second as the delivery. An example of a route with 4 requests following this representation is: (**2 1** *1* **3 4** *2 3 4*), where pickups are shown in boldface and deliveries in italics.

Also, to deal with the hard time window constraint, our routing algorithm adopts an intelligent neighbourhood move that uses the time window as a guidance. The idea is to try to improve the current route by creating a new neighbouring route. To avoid the frequent creation and evaluation of infeasible routes, though, our neighbourhood move only swaps locations that are out of order in terms of their late time window bounds, i.e., if the latter location has a deadline that precedes the earlier one. Having dealt with the precedence and the time windows constraints, the capacity constraint is the only remaining issue. However, due to the nature of the problem, the capacity constraint is often easily satisfied, since half of the locations in the route are delivery locations whose loads are removed from the vehicle. This simple representation and neighbourhood move are employed in a classical Hill-Climbing (HC) route-improvement heuristic, which tries to gradually modify the current route until no further improvement is possible. Algorithm 1 describes this simple heuristic.

**Algorithm 1** The HC Routing Algorithm

1: Given a route $r$
2: **repeat**
3: **for** (Each possible pair of locations in $r$) **do**
4: **if** (The latter location is more urgent in its upper time window bound) **then**
5: Swap the current two locations in $r$ to get a new route $r'$
6: $\Delta \leftarrow cost(r') - cost(r)$
7: **if** ($\Delta < 0$) **then**
8: $r \leftarrow r'$
9: **until** (Done){Stop when no improvement achieved in the previous pass}

The cost function used in Step 6 of the HC algorithm to evaluate the quality of each route tries to minimize the total route duration as well as the degree of infeasibility in capacity and time windows constraints. The cost function of a route $r$ is described by the following equation:

$$F(r) = w_1 \times D(r) + w_2 \times TWV(r) + w_3 \times CV(r) \ , \qquad (1)$$

where $D(r)$ is the total route duration, including the waiting time and the service time at each location. $TWV(r)$ is the total number of time window violations in the route, and $CV(r)$ is the total number of capacity violations. The constants $w_1, w_2, and\ w_3$ are weights in the range $[0, 1]$, and $w_1 + w_2 + w_3 = 1.0$ . The choice of appropriate weights depends on the importance of each term in the objective function. We found that in order to get feasible solutions, the largest penalty should be imposed on the time window violations.

# 5 Solution Construction Heuristics

In all our construction heuristics we first start by sorting customers according to the distance from the depot (farthest first). However, since in our approach we deal with customers in pairs, where each pair consists of a pickup location and its associated delivery, the distance measure, in relation to the depot, could either be the distance between the depot and the pickup location, or the distance between the depot and the delivery location. We arbitrarily chose the distance separating the depot and the delivery location for the initial order of requests.

## 5.1 The Sequential Construction Algorithm

The sequential construction heuristic tries to build routes one after another. Requests are taken one by one in order, and each request (pickup and delivery pair) is initially inserted at the end of the current route. Our HC routing heuristic (Algorithm 1) is then called to try to improve the current route. If the HC algorithm returns an improved route that can 'feasibly' accommodate the newly inserted pair, this insertion is accepted and we move on to the next request. However, if the 'improved' route is still infeasible, the newly inserted pair is removed from the current route to wait for another insertion attempt in a new route. Thus, unlike the 'traditional' insertion methods, our algorithm relies on the HC heuristic to improve the quality of the current route, without actually having to calculate the cost of each and every possible insertion position in order to select the best one among them.

Algorithm 2 describes the sequential construction procedure. It is important to note in Step 7 of this algorithm that, besides overcoming the precedence and the coupling issues, inserting a request (a pickup and delivery pair) at the end of the route has the added advantage of speeding up the insertion process, since two locations instead of one are simultaneously inserted.

**Algorithm 2** The Sequential Construction

```
1: Let M ← 0 {M is the number of vehicles used}
2: repeat
3:    Initialize an empty route r
4:    M = M + 1
5:    for (All unassigned requests) do
6:       Get the next unassigned request i
7:       Insert the request i at the end of the current route r
8:       Call the HC routing heuristic (Algorithm 1) to improve r
9:       if (r is a feasible route) then
10:         Mark i as inserted
11:      else
12:         Remove i from r
13: until (All requests have been inserted)
```

### 5.2 The Parallel Construction Algorithms

As mentioned previously, for a parallel construction, several routes are considered simultaneously for inserting a new request, and an initial estimate of the number of vehicles is required. Potvin and Rousseau in their parallel construction algorithm for solving the VRPTW [15], estimate the initial number of vehicles by first running Solomon's sequential construction [19]. The number of vehicles in the resulting solution is then used as an estimate of the initial number of vehicles for the parallel heuristic.

In our research, we adapted the parallel construction heuristic of the VRPTW in [15] to the MV-PDPTW. However, to avoid unnecessary extra processing time, we estimated the initial number of vehicles using a simple formula that divides the total demand of the pickup requests in the problem instance by the capacity of the vehicle, as shown in (2).

$$M = \lfloor (\sum_{i \in N^+} q_i)/C \rfloor \ , \tag{2}$$

where $M$ is the estimated initial number of vehicles, $N^+$ is the set of pickup customers, $q_i$ is the demand (load) of a pickup request, and $C$ is the capacity of the vehicle. However, this estimate seems to be more suitable for instances with a critical (short) schedule horizon. Instances with more flexible time window intervals, on the other hand, may require fewer vehicles to start with. As a result, we introduced a small modification to this formula for some problem instances, as will be explained in Sect. 6.

Similar to the parallel approach for the VRPTW in [15], which initializes each route with a seed *customer*, our parallel algorithms initialize each route with a seed *request* (pickup and delivery pair) from the sorted list of requests. We then take the remaining requests in order and attempt to insert the next

request in one of the partial routes created. If the next request cannot be feasibly inserted in any of the already created routes, a new route is added to accommodate this request. This process is repeated until all requests have been inserted.

As mentioned previously, 'traditional' parallel construction algorithms for both the VRPTW and the PDPTW, usually select the request who has the current minimum insertion cost among all remaining un-routed requests to be inserted next. This cost is often a measure of the extra travel time and distance, which would result from inserting the request in the best possible (feasible and minimum cost) insertion position found in all available routes. Our parallel algorithms, on the other hand, differ among each other in how they select the next request to be inserted, and also the route in which this request will be inserted. Following is an explanation of the different parallel construction algorithms proposed in our research.

### 5.2.1 Parallel Construction - First Route:

In our first parallel construction algorithm, the next request in order is inserted in the first route in which a feasible insertion of this request is found, i.e., no attempt is made to find the best route for the current request. Thus, our first parallel construction uses a fast first acceptance criterion for insertion. Algorithm 3 describes this procedure.

**Algorithm 3** Parallel Construction: First Route

1: Calculate $M$ (the initial estimate of the number of vehicles)
2: Initialize $M$ routes with seed customer pairs from the sorted list of customers
3: **for** (All remaining unassigned requests) **do**
4:  Get the next unassigned request $i$
5:  $r = 0$ {start with the first route}
6:  **while** (($r < M$) and ($i$ not yet inserted) ) **do**
7:  Insert the request $i$ at the end of the current route $r$
8:  Call the HC routing heuristic (Algorithm 1) to improve $r$
9:  **if** ($r$ is a feasible route) **then**
10:  Mark $i$ as inserted
11:  **else**
12:  Remove $i$ from $r$
13:  $r = r + 1$
14:  **if** ($i$ was not inserted) **then**
15:  Initialize a new route $r'$
16:  $M = M + 1$ {increase the number of vehicles}
17:  Insert the request $i$ in the new route $r'$
18:  Mark $i$ as inserted

### 5.2.2 Parallel Construction - Best Route:

In our second parallel construction algorithm, the next request in order is inserted in the best route in which a feasible insertion of this request is found. The best route for each request is the route that causes the least increase in the overall cost of the solution (the routing schedule) due to the insertion process.

To calculate the overall cost of the solution, we used an objective function that is suggested by Bent and Hentenryck in [1]. The objective function consists of 3 components: the first component tries to minimize the number of vehicles used in the solution, the second component tries to minimize the total distance travelled, while the third component is a measure that tries to maximize the square of the number of nodes visited by each vehicle. This last component is intended to favour routes that are rather full and those that are rather empty, as opposed to an even distribution of nodes among routes. The idea is to try to get rid of some vehicles that are under-utilized during subsequent route improvement phases. The objective function of a solution $S$ is described by (3).

$$O(S) = \alpha \times M + \beta \times \sum_{r \in S} Dist(r) - \gamma \times \sum_{r \in S} |r|^2 \ , \tag{3}$$

where $M$ is the number of vehicles used in the current solution, $Dist(r)$ is the total distance travelled by each vehicle, and $|r|$ is the number of nodes visited by each vehicle. The constants $\alpha,\ \beta,\ and\ \gamma$ are weights in the range $[0, 1]$ assigned to each term in the objective function, and $\alpha + \beta + \gamma = 1.0$ . In our research we try to minimizing the number of vehicles as our primary objective followed by the total distance, thus we chose $\alpha > \beta > \gamma$. Algorithm 4 describes the second parallel construction algorithm.

**Algorithm 4** Parallel Construction: Best Route

1: Calculate $M$ (the initial estimate of the number of vehicles)
2: Initialize $M$ routes with seed customer pairs from the sorted list of customers
3: **for** (All remaining unassigned requests) **do**
4: Initialize $LocalMin$ to an arbitrary large value
5: **for** $(r = 0; r < M; r++)$ **do**
6: Get the next unassigned request $i$
7: Insert the request $i$ at the end of the current route $r$
8: Call the HC routing heuristic (Algorithm 1) to improve $r$
9: **if** ($r$ is a feasible route) **then**
10: calculate $\Delta cost$ {$\Delta cost$ is the change in solution cost due to the insertion}
11: **if** $(\Delta cost < LocalMin)$ **then**
12: $LocalMin = \Delta cost$
13: $r* = r$ { $r*$ is the current best vehicle for request $i$}
14: Remove $i$ from $r$ {temporarily remove $i$ until insertion costs of the current request in all routes have been calculated}
15: **if** ($r*$ is found) **then**
16: Insert $i$ in $r*$
17: Mark $i$ as inserted
18: **else**
19: Initialize a new route $r'$ {Since no feasible insertion is found for $i$ in any of the available routes, allocate a new route}
20: $M = M + 1$ {increase the number of vehicles}
21: Insert the request $i$ in the new route $r'$
22: Mark $i$ as inserted

It is important to note, in Step 10 of Algorithm 4, that since the insertion process only affects one route, the calculation of the new solution cost does not require evaluating all routes in the current solution. The calculation is simply done by removing the old cost of the current route (before the insertion), and adding the new cost resulting from the insertion.

### 5.2.3 Parallel Construction - Best Request:

Our next parallel construction heuristic does not only try to find the best route for each request, but also tries to select the best un-routed request to be inserted next. The best un-routed request is the one whose insertion (in its best route) causes the least increase in the overall cost of the solution. To evaluate the cost of the solution, the same objective function used in Algorithm 4, i.e, (3), is used. Algorithm 5 describes this procedure.

**Algorithm 5** Parallel Construction: Best Request

```
1: Calculate M (the initial estimate of the number of vehicles)
2: Initialize M routes with seed customer pairs from the sorted list of customers
3: repeat
4:   Initialize GlobalMin to an arbitrary large value
5:   for (All remaining unassigned requests) do
6:     Initialize LocalMin to an arbitrary large value
7:     for (r = 0; r < M; r++) do
8:       Get the next unassigned request i
9:       Insert the request i at the end of the current route r
10:      Call the HC routing heuristic (Algorithm 1) to improve r
11:      if (r is a feasible route) then
12:        calculate Δcost {Δcost is the change in solution cost due to the insertion}
13:        if (Δcost < LocalMin) then
14:          LocalMin = Δcost
15:          r* = r { r* is the current best route for request i}
16:      Remove i from r {temporarily remove i until insertion costs of all requests in all routes have been calculated}
17:    if (r* is found) then
18:      if (LocalMin < GlobalMin) then
19:        GlobalMin = LocalMin
20:        i* = i {i* is the current best request}
21:        v* = r* {v* is the best vehicle (route) for i*}
22:    else
23:      Initialize a new route r' {because no feasible insertion is found for i in any of the available routes}
24:      M = M + 1
25:      Insert i in the new route r'
26:      Mark i as inserted
27:  if (i* is found) then
28:    Insert i* in v*
29:    Mark i* as inserted
30: until (All requests have been inserted)
```

# 6 Computational Experimentation

## 6.1 Characteristics of the Data Set

To test our algorithms, we used several instances from the benchmark data of the MV-PDPTW created by Li and Lim in [10]. The authors of [10] created this data set based on Solomon's test cases of the VRPTW in [19].

Table 1: Test Files

| Category | 100 customers | 200 customers | 400 customers |
|---|---|---|---|
| **LC1** | LC101 to LC106 | LC1-2-1 to LC1-2-6 | LC1-4-1 to LC1-4-6 |
| **LC2** | LC201 to LC206 | LC2-2-1 to LC2-2-6 | LC2-4-1 to LC2-4-6 |
| **LR1** | LR101 to LR106 | LR1-2-1 to LR1-2-6 | LR1-4-1 to LR1-4-6 |
| **LR2** | LR201 to LR206 | LR2-2-1 to LR2-2-6 | LR2-4-1 to LR2-4-6 |
| **LRC1** | LRC101 to LRC106 | LRC1-2-1 to LRC1-2-6 | LRC1-4-1 to LRC1-4-6 |
| **LRC2** | LRC201 to LRC206 | LRC2-2-1 to LRC2-2-6 | LRC2-4-1 to LRC2-4-6 |
| **Category** | **600 customers** | **800 customers** | **1000 customers** |
| **LC1** | LC1-6-1 to LC1-6-6 | LC1-8-1to LC1-8-6 | LC1-10-1 to LC1-10-6 |
| **LC2** | LC2-6-1 to LC2-6-6 | LC2-8-1to LC2-8-6 | LC2-10-1 to LC2-10-6 |
| **LR1** | LR1-6-1 to LR1-6-6 | LR1-8-1 to LR1-8-6 | LR1-10-1 to LR1-10-6 |
| **LR2** | LR2-6-1 to LR2-6-6 | LR2-8-1 to LR2-8-6 | LR2-10-1 to LR2-10-6 |
| **LRC1** | LRC1-6-1 to LRC1-6-6 | LRC1-8-1 to LRC1-8-6 | LRC1-10-1 to LRC1-10-6 |
| **LRC2** | LRC2-6-1 to LRC2-6-6 | LRC2-8-1 to LRC2-8-6 | LRC2-10-1 to LRC2-10-6 |

There are 6 different categories of problem instances in this data set: LR1, LR2, LC1, LC2, LRC1, and LRC2. Problems in the LR category have randomly distributed customers, problems in the LC category have clustered customers, and problems in the LRC category have partially random and partially clustered customers. On the other hand, problems identified with the number '1' have a short scheduling horizon (tight time window width), while problems identified with the number '2' have a long scheduling horizon (large time window width). Each category has 6 different problem sizes: 100, 200, 400, 600, 800, and 1000 customers[2]. There are between 56-60 files from each problem size. The total number of files in the data set is 354. The data and the best known results can be downloaded from `http://www.sintef.no/Projectweb/TOP/Problems/PDPTW/Li--Lim-benchmark/`. For the purpose of testing our algorithms we selected the first 6 files from each category for each problem size. The total number of files used to test our algorithms is 216. The files used for testing our algorithms are summarized in Table 1.

As mentioned in Sect. 5.2, we used a simple formula (2) to estimate the initial number of vehicles needed for the parallel construction heuristics. However, during our preliminary experimentation, we found that this estimate does not suit all the different types of problem instances. Apparently, problems that have a long schedule horizon allow for a more flexible visiting schedule, and generally require fewer vehicles. We also found during our experimentation that an under-estimate of the initial number of vehicles is usually preferred to an over-estimate, since reducing the total number of

[2]The original data set in [10] contained only 56 100-customers problems. Larger problem sizes were later added to the original data set.

vehicles used is our primary concern. As a result, to estimate the initial number of vehicles for problems with a long schedule horizon (problems of category '2') we reduced our initial estimate by 50%. Thus, (4) was used instead of (2).

$$M = \lfloor (1/2)(\sum_{i \in N^+} q_i)/C \rfloor \ . \tag{4}$$

However, this estimate is to some extent arbitrary and remains under consideration for future reassessment.

### 6.2 Comparing the Construction Heuristics

Throughout the following discussion, we use the following notations to refer to each algorithm

1. Sequential Construction: *SEQ*
2. Parallel Construction - First Route: *PFR*
3. Parallel Construction - Best Route: *PBR*
4. Parallel Construction - Best Request: *PBQ*

The algorithms were implemented using Visual C++ under a Windows XP operating system, on Intel Pentium (R)D CPU 3.40 GHz and 2 GB RAM. Since the construction algorithms are all deterministic, each algorithm was run only once on each test file.

Table 2 shows the average number of vehicles, the average total distance, and the average processing time (in seconds), produced by each algorithm for each problem size separately. Table 3 shows the percentage of time each algorithm produced the minimum number of vehicles and the minimum total distance (as found in the current experiment), over all 216 problem instances[3].

The following observations can be realized from Tables 2 and 3:

- Regarding the number of vehicles generated, *SEQ* and *PBQ* produced the best results, with *SEQ* producing better results than *PBQ* in large size problems, while both *PFR* and *PBR* were slightly inferior in this respect
- Regarding the total distance travelled, *PBQ* was able to beat all other algorithms, followed by *PBR* and *SEQ*
- *PFR* produced the worst average distance in all test cases, but it was slightly better than *PBR* in the average number of vehicles used

[3]Some ties are produced and counted in the results.

Table 2: Average Results for all Algorithms

| Problem Size | SEQ | | | PFR | | |
|---|---|---|---|---|---|---|
| | **Vehic** | **Dist** | **Time** | **Vehic** | **Dist** | **Time** |
| **100-customers** | 11.78 | 2662.92 | 0.02 | 11.83 | 2767.19 | 0.02 |
| **200-customers** | 17.33 | 8887.08 | 0.08 | 17.69 | 8954.06 | 0.08 |
| **400-customers** | 33.56 | 22215.14 | 0.32 | 34.64 | 23010.53 | 0.3 |
| **600-customers** | 48.22 | 44949.4 | 0.72 | 49.89 | 46644.49 | 0.69 |
| **800-customers** | 63.53 | 74650.07 | 1.24 | 65.94 | 77895.32 | 1.23 |
| **1000-customers** | 77.25 | 108513.19 | 1.88 | 81.97 | 115106.93 | 1.93 |
| **Average** | **41.95** | **43646.30** | **0.71** | **43.66** | **45729.75** | **0.71** |
| **Problem Size** | **PBR** | | | **PBQ** | | |
| | **Vehic** | **Dist** | **Time** | **Vehic** | **Dist** | **Time** |
| **100-customers** | 11.83 | 2711.89 | 0.03 | 11.69 | 2564.09 | 0.34 |
| **200-customers** | 18.17 | 8816.33 | 0.1 | 17.14 | 8132.84 | 3.62 |
| **400-customers** | 34.69 | 21898.96 | 0.38 | 33.72 | 19758.38 | 26.93 |
| **600-customers** | 50.69 | 45234.96 | 0.86 | 49.53 | 41791.82 | 147.57 |
| **800-customers** | 66.44 | 74056.45 | 1.65 | 64.89 | 68713.31 | 438.97 |
| **1000-customers** | 81.75 | 108662.01 | 2.54 | 81.58 | 103751.31 | 952.34 |
| **Average** | **43.93** | **43563.43** | **0.92** | **43.09** | **40785.29** | **261.62** |

Table 3: Frequency of Generated Best Solutions

| Algorithm | Min-Vehic | Min-Dist |
|---|---|---|
| **SEQ** | 48% | 31% |
| **PFR** | 24% | 6% |
| **PBR** | 19% | 8% |
| **PBQ** | 49% | 56% |

- *SEQ* and *PFR* have compatible average processing time in all test cases. Their processing time on average is faster than the other two algorithms, with *PBQ* being the slowest among all

In summary, the results in Tables 2 and 3 suggest that *PFR* and *PBR* are inferior to *SEQ* and *PBQ*, both in terms of the number of vehicles used and the total distance travelled. However, *PBR* was able to slightly improve upon *PFR* with respect to the total distance travelled, while *PFR* was slightly better than *PBR* in the number of vehicles used. As a result, *PFR* and *PBR* can be eliminated from further consideration, and we can focus our attention on *SEQ* and *PBQ*.

As can be noticed from the overall average results shown in Table 2, *SEQ* produced better results than *PBQ* in the number of vehicles used. The *PBQ* algorithm, however, was able to beat the *SEQ* algorithm in minimizing the total distance travelled. This was obviously due to the fact that the *SEQ* algorithm was more concerned with fitting the largest possible number of requests in each vehicle before allocating a new one, while the *PBQ* algorithm relied on a cost function that has the total travel distance among its components. The *PBQ* algorithm was, nevertheless, much slower than the *SEQ* algorithm. The average processing time of the *SEQ* algorithm ranged from 0.02 seconds for 100-customers problems to 1.88 seconds for 1000-customers problems. The *PBQ* algorithm, on the other hand, had a processing time ranging from 0.34 seconds to 952.34 seconds for the same problem types, which indicates beyond doubt the huge difference in the computational effort needed for both algorithms.

We also performed a one-way analysis of variance of the average results produced by both the *SEQ* and the *PBQ* algorithms. The analysis showed that there is no statistically significant difference in the average results produced by the two algorithms, both in terms of the number of vehicles and the total distance. This further indicates that the *SEQ* algorithm, despite its simplicity and its exceptional speed, produced comparable results to the results of the *PBQ* algorithm. It should also be noted that the *SEQ* algorithm neither requires an initial estimate of the number of vehicles, nor does it need a solution evaluation mechanism during the construction process. The only advantage that the *PBQ* algorithm offers, which is a slight reduction in the total travel distance, does not seem to justify its added cost in terms of the complexity of the algorithm and the increase in processing time. Another advantage of the *SEQ* algorithm is that it can be easily adapted to population-based heuristics or meta-heuristics by randomizing the initial order of requests to generate different diverse solutions. The *PBQ* algorithm, on the other hand, is expected to produce a limited diversity, even if the initial order of requests is randomized, because of the selection criteria and the cost function it relies on during the insertion process. Most likely, requests that are hard to insert, and thus cause a large increase in the solution cost,

Table 4: Average Relative Distance to Best Known

| Problem Size | Vehic-Gap | | Dist-Gap | |
|---|---|---|---|---|
| | **SEQ** | **PBQ** | **SEQ** | **PBQ** |
| **100-customers** | 58% | 57% | 146% | 137% |
| **200-customers** | 63% | 61% | 187% | 163% |
| **400-customers** | 66% | 67% | 207% | 173% |
| **600-customers** | 64% | 69% | 206% | 185% |
| **800-customers** | 66% | 70% | 205% | 181% |
| **1000-customers** | 65% | 74% | 191% | 178% |
| **Average** | **64%** | **66%** | **191%** | **170%** |

will always remain the same, despite the change in the insertion order.

### 6.3 Comparing with Previous Best Known

Although our algorithms are not intended to provide final good quality solutions to the MV-PDPTW, it would still be useful to compare our results with the best known solutions. This would give us a general idea about the expected effort needed in the solution improvement phase. We tried to analyse the relative gap (difference) to best known results, produced by the *SEQ* algorithm for each benchmark category separately. The best known results are reported in [18].

Table 4 shows the relative distance (in percentage) between the average results produced by both the *SEQ* algorithm and the *PBQ* algorithm and the average best known results. The relative distance (gap) is measured with respect to both the number of vehicles and the distance[4]. The table shows that the *SEQ* algorithm produced, on average, a slightly smaller gap with respect to the number of vehicles, and a slightly larger gap with respect to the total distance. Together with the fact that the *SEQ* algorithm is quite simple and fast compared to the *PBQ* algorithm, the results in Table 4 would again seem to justify its preference as a solution construction method over the *PBQ* algorithm.

It would also be beneficial to try to analyse the results produced by the construction heuristics for each benchmark category separately. This may give an insight into what problem types would require more effort in the solution improvement phase. Figure 1 shows the average gap produced by the *SEQ* algorithm for all tasks, organized by problem categories. Figure 2 shows the average gap produced by the same algorithm with respect to the distance travelled.

[4]For example, a gap of 50% in the average number of vehicles means that the result of the construction heuristic produced 50% more vehicles than the best known result.

Both figures show that the *SEQ* construction heuristic seems to be more 'successful' in instances with a short schedule horizon, i.e., instances identified with '1' in the data set, since these instances always have a smaller gap than instances of type '2'. Regarding the primary objective, which is the number of vehicles used, the algorithm seems to do a better job for instances that have clustered customers, as opposed to instances that have random or partially random customers. It is clear that instances in the *LC* category always have the smallest gap compared to the other problem types. Problems with random customers and a long time window interval appear to be the most challenging for the *SEQ* algorithm, and possibly all solution algorithms. The reason could be that the solution space for these problems seems to be larger, due to the randomness of locations and the large width of time windows involved in this case. It also appears from both graphs that the gap in the number of vehicles is inversely proportional to the gap in the total travel distance, in most test cases. This indicates that a solution that uses more vehicles may result in an overall shorter travel distance compared to a solution that uses less number of vehicles.

It is also worth mentioning that the results in Table 4 and Figs. 1 and 2 clearly indicate that there is a lot of work still to be done in the improvement phase, in order to reach the anticipated standard for the final problem solutions. This is evident by the relatively large gap between the current initial solutions and the final best known results. Designing an 'intelligent' improvement phase seems to be inevitable, in order to cope with the difficult problem constraints and the various types of problem instances.

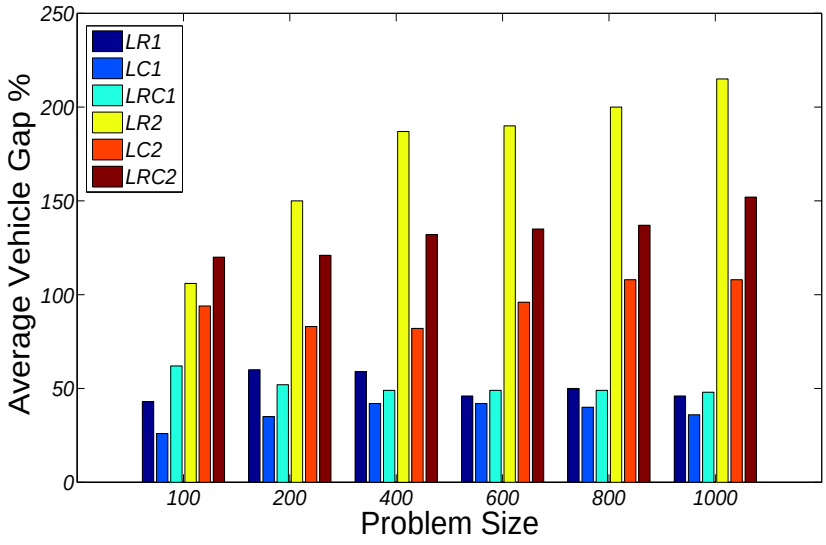


Figure 1: *SEQ* algorithm - average vehicle gap for all problem categories

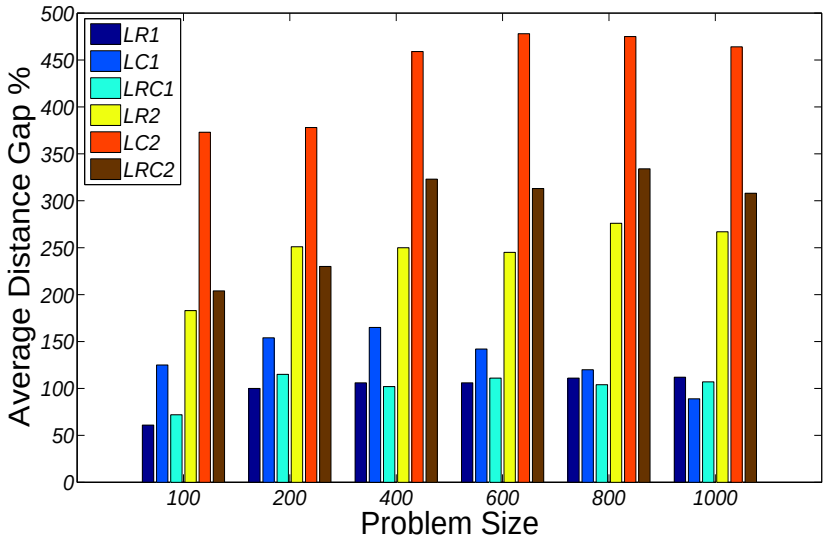


Figure 2: *SEQ* algorithm - average distance gap for all problem categories

Finally, Table 5 shows the average processing time of the *SEQ* algorithm, for each problem size in each benchmark category. It appears in this table that problems involving random customers with a long schedule horizon, *LR2* and *LRC2*, generally require a longer processing time than the other problem categories, which sustains our previous observation regarding the large solution space for these problems.

Table 5: Average Processing Time (seconds) of the SEQ Algorithm for all Tasks

| **Category** | **Problem Size** | | | | | |
|---|---|---|---|---|---|---|
| | **100** | **200** | **400** | **600** | **800** | **1000** |
| **LR1** | 0.02 | 0.06 | 0.22 | 0.53 | 0.95 | 1.54 |
| **LC1** | 0.02 | 0.05 | 0.22 | 0.48 | 0.92 | 1.43 |
| **LRC1** | 0.02 | 0.06 | 0.21 | 0.50 | 0.93 | 1.44 |
| **LR2** | 0.03 | 0.13 | 0.43 | 1.04 | 1.71 | 2.72 |
| **LC2** | 0.03 | 0.06 | 0.39 | 0.61 | 1.06 | 1.57 |
| **LRC2** | 0.02 | 0.12 | 0.47 | 1.12 | 1.84 | 2.53 |

# 7 The SEQ Algorithm: Complexity Analysis and Implementation Issues

We present in this section some remarks concerning the complexity and feasibility checking of the SEQ algorithm in relation to the common construction methods. Analysing our SEQ algorithm we find that: the routing heuristic in Algorithm 1 needs $O(n^2)$ time for accessing each pair of locations in the route, where $n$ is the number of requests in the problem instance. Also, the cost function (1), which checks the feasibility of the whole route as well, needs $O(n)$ time. The SEQ algorithm (Algorithm 2) needs $O(n)$ since its major iteration processes all requests in order. This will make the run-time complexity of the whole algorithm $O(n^4)$.

For the sake of comparison, Algorithm 6 describes a basic construction method for vehicle routing problems in general. The algorithm appears in [2].

**Algorithm 6** Basic Construction Algorithm for the VRP [2]

```
1:  N = set of unassigned customers
2:  R = set of routes {initially contains one route}
3:  while N ≠ ∅ do
4:     p* = −∞
5:     for j ∈ N do
6:        for r ∈ R do
7:           for (i − 1, i) ∈ r do
8:              if (Feasible(i, j) and Profit(i, j) > p*) then
9:                 r* = r
10:                i* = i
11:                j* = j
12:                p* = Profit(i, j)
13:    Insert(i*, j*) {insert j* between (i* − 1) and i*}
14:    N = N \ j*
15:    Update(r*)
```

According to [2], Algorithm 6 is of $O(n^3)$, provided that the *feasibility* and *profitability* can be performed in constant time. *Feasibility* makes sure that the current position adheres to all problem constraints, while *Profitability* is usually measured as a weighted combination of extra travel distance and time delay resulting from the insertion. When it is more profitable to insert a customer in a new route, a new route will be allocated. As explained in [2], a special algorithm can be applied to reduce the TW feasibility test of the VRPTW from a linear time to a constant time. The same algorithm can also be applied to the PDPTW. Nevertheless, this algorithm requires that extra information is kept for each customer already existing in the route. In general, two quantities have to be maintained: the earliest and the latest possible times the service can take place, relative to the customer's current

location in the route. This information is not fixed and subject to change after each insertion, which accounts for the existence of an *update* function to maintain the desired quantities (Step 15 of Algorithm 6).

Unlike the VRPTW, however, checking the capacity feasibility for the PDPTW can only be done in linear time [9], due to the presence of two different types of customer services in the route. As a result, the basic construction algorithm when applied to the PDPTW also results in $O(n^4)$ complexity. The algorithm can also include a selection of seed customers for route initialization, which usually does not change the complexity of the algorithm.

As mentioned above, it is possible to reduce the TW feasibility check for the PDPTW from a linear time to a constant time, by maintaining and frequently updating extra route information, as done for example in [4]. However, since our route cost function (1) tests the feasibility of both the TW and the capacity concurrently, it would be redundant to calculate and store additional service timing information to accelerate the TW feasibility test, since an $O(n)$ testing would still be needed for the capacity feasibility.

In addition, since our SEQ algorithm accepts any feasible insertion, it does not have to check the feasibility nor estimate the profitability of each and every possible insertion position, as done in Algorithm 6. In our algorithm, the cost of the route as a whole will be calculated, if at all, only if the route has been changed. This is due to the restriction imposed by the TW condition in Step 4 of Algorithm 1.

Finally, during the insertion process, i.e., Step 7 of the SEQ algorithm (Algorithm 2), two locations (a pickup and delivery pair) are simultaneously inserted, then Algorithm 1 handles the feasibility checking and the improvement of the underlying route altogether. Besides overcoming the precedence and the coupling issues, this insertion has the added advantage of accelerating the solution construction process, since only half the number of locations is processed in the main iteration of Algorithm 2.

On the other hand, the parallel construction algorithms implemented in this research seem to be one order of magnitude higher than the SEQ algorithm, due to the presence of an extra loop that passes through all available vehicles, although the number of vehicles is always less than $n$ (the number of nodes in the data set).

# 8 Summary and Conclusions

In this research we investigated several initial solution construction heuristics for the MV-PDPTW, aiming to identify the best heuristic that can be used as part of a comprehensive solution methodology. In our opinion, existing approaches in the literature often overlook and perhaps underestimate this vital component of the overall solution algorithm.

The experimental results on a large number of benchmark instances indicate that the sequential construction heuristic (SEQ) seems to be the most favourable solution construction method, which can be by easily embedded in a heuristic or a meta-heuristic technique to reach final good quality solutions. With just a few simple lines of code, and without a pre-determined number of vehicles or a solution evaluation mechanism, this algorithm produced good quality results, that are sometimes even better than the results obtained by the most sophisticated parallel algorithm tested in our research (the PBQ algorithm). The SEQ algorithm also had an impressive speed, with a processing time that is at most 6% of the time needed by the PBQ algorithm, making it even more suitable for population-based solution algorithms.

The construction algorithms developed in this research are distinguished by their simplicity and ease in coding and replication, compared to many construction methods that are adopted from the VRPTW literature. All of our algorithms are general portable frameworks that can be used within other heuristics and meta-heuristics that solve the PDPTW and its related variants [5].

# References


[1] R. Bent and P. Van Hentenryck. A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows. *Computers & Operations Research*, 33(4):875–893, 2006.

[2] A. M. Campbell and M. Savelsbergh. Efficient insertion heuristics for vehicle routing and scheduling problems. *Transportation Science*, 38(3):369–378, 2004.

[3] J.-F. Cordeau and G. Laporte. The dial-a-ride problem (DARP): Variants, modeling issues and algorithms. *4OR: A Quarterly Journal of Operations Research*, 1(2):89–101, 2003.

[4] U. Derigs and T. Döhmer. Indirect search for the vehicle routing problem with pickup and delivery and time windows. *OR Spectrum*, 30(1):149–165, 2008.

[5] M. I. Hosny. Comparing genetic algorithms and simulated annealing for solving the pickup and delivery problem with time windows. In H. R. Arabnia, D. de la Fuente, E. B. Kozerenko, and J. A. Olivas, editors, *Proceedings of the 2011 International Conference on Artificial*


[5] The interested reader may follow up this research in a recently published paper [5], where an example of using the SEQ algorithm within two meta-heuristic approaches (a GA and an SA) for solving the MV-PDPTW is presented.

*Intelligence, ICAI'11, Las Vegas, Nevada, USA*, volume II, pages 513–519. CSREA Press, July 2011.

[6] M. I. Hosny and C. L. Mumford. New solution construction heuristics for the multiple vehicle pickup and delivery problem with time windows. In *MIC2009, Metaheuristic International Conference*, July 2009.

[7] M. I. Hosny and C. L. Mumford. The single vehicle pickup and delivery problem with time windows: Intelligent operators for heuristic and metaheuristic algorithms. *Journal of Heuristics, Special Issue on Advances in Metaheuristics*, 16(3):417–439, June 2010.

[8] Manar I. Hosny. *Investigating Heuristic and Meta-heuristic Algorithms for Solving Pickup and Delivery Problems*. PhD thesis, Cardiff School of Computer Science & Informatics, Cardiff University, UK, March 2010. `http://faculty.ksu.edu.sa/M_F_Hosny/Documents/Thesis.pdf`.

[9] B. Hunsaker and M. Savelsbergh. Efficient feasibility testing for dial-a-ride problems. *Operations Research Letters*, 30(3):169–173, 2002.

[10] H. Li and A. Lim. A metaheuristic for the pickup and delivery problem with time windows. In *Proceedings of the 13th IEEE International Conference on Tools with Artificial Intelligence*, pages 160–167, November 2001. Dallas, TX, USA.

[11] H. Lim, A. Lim, and B. Rodrigues. Solving the pickup and delivery problem with time windows using iSSQUEAKY WHEELi optimization with local search. In *AMCIS 2002 Proceedings*, 2002.

[12] Q. Lu and M. Dessouky. A new insertion-based construction heuristic for solving the pickup and delivery problem with time windows. *European Journal of Operational Research*, 175(2):672–687, December 2006.

[13] G. Pankratz. A grouping genetic algorithm for the pickup and delivery problem with time windows. *OR Spectrum*, 27:21–24, 2005.

[14] S. Parragh, K. Doerner, and R. Hartl. A survey on pickup and delivery problems. *Journal für Betriebswirtschaft*, 58(2):81–117, 2008.

[15] J.-Y. Potvin and J.-M. Rousseau. A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. *European Journal of Operational Research*, 66(3):331–340, May 1993.

[16] S. Ropke and D. Pisinger. An adaptive large neighbourhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(4):455–472, November 2006.

[17] M. Savelsbergh and M. Sol. The general pickup and delivery problem. *Transportation Science*, 29(1):17–29, 1995.

[18] SINTEF. Vehicle routing and travelling salesperson problems. http://www.top.sintef.no/vrp/benchmarks.html.

[19] M. M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2):254–265, 1987.