PICKUP AND DELIVERY WITH TIME WINDOWS: ALGORITHMS AND TEST CASE GENERATION

HOONG CHUIN LAU ZHE LIANG
School of Computing, National University of Singapore, Singapore 117543.

In the pickup and delivery problem with time windows (PDPTW), vehicles have to transport loads from origins to destinations respecting capacity and time constraints. In this paper, we present a two-phase method to solve the PDPTW. In the first phase, we apply a novel construction heuristics to generate an initial solution. In the second phase, a tabu search method is proposed to improve the solution. Another contribution of this paper is a strategy to generate good problem instances and benchmarking solutions for PDPTW, based on Solomon's benchmark test cases for VRPTW. Experimental results show that our approach yields very good solutions when compared with the benchmarking solutions.

Keywords: Pickup and delivery problem, vehicle routing problem, tabu search, test case generation.

1. Introduction

The *Pickup and Delivery Problem with Time Windows* (PDPTW) models the situation in which a fleet of vehicles must service a collection of transportation requests. Each request specifies a pickup and delivery location. Vehicles must be routed to service all requests, satisfying time windows and vehicle capacity constraints while optimizing a certain objective function such as total distance traveled. PDPTW can be used to model many core problems arising in logistics and public transit. Finding good solutions to these problems is important because it enables planners to utilize the existing fleet in the most cost-effective fashion to meet customer demands.

PDPTW is a generalization of well-known Vehicle Routing Problem with Time Window (VRPTW). PDPTW is an NP-hard problem, since VRP is a well-known NP-hard problem [1]. While VRPTW is well-studied (for a comprehensive survey, see [2], there is relatively less literature on PDPTW. Moreover, no one has developed comprehensive benchmark PDPTW instances that facilitate experimentation of new approaches.

In this paper, we propose a new approach for PDPTW, extending and improving the results of [3]. We also propose a method to generate good problem instances and solutions for PDPTW, based on Solomon's benchmark data sets for VRPTW. This turns out to be an interesting exercise, because good solutions for VRPTW do not imply good solutions for PDPTW, mainly due to the fact that while the vehicle load cumulatively increases along each route for VRPTW, it is not the case for PDPTW as pickups and deliveries occur in juxtaposition.

2. Problem Formulation

In our model, we assume there are an unlimited number of vehicles and all vehicles have the same capacity. Let N be the set of transportation requests. For each request $i \in N$, a load of size q_i is to be transported from an origin N_i^+ to a destination N_i^- (positive qi for pickup and negative q_i for delivery). Each pickup or delivery is also referred to as a *job*. Define $N^+ \equiv \bigcup_{i \in N} N_i^+$ and $N^- \equiv \bigcup_{i \in N} N_i^-$ as the sets of all origins and destinations respectively. For simplicity, assume Ni^+ and N_i^- to be disjoint. Let $V \equiv N^+ \bigcup N^-$ and $n = |V|$. Let M and m denote the set and number of vehicles. Each vehicle has a capacity Q, starting from and ending with the depot O with no cargo. For all $i, j \in V \bigcup O$, let d_{ij} denote the travel distance and t_{ij} the travel time. Let $[e_i, l_i]$ denote the time window, i.e. time interval in which service at location i must take place. Note that the service durations at the origins and destinations can be easily incorporated in the travel times and hence will not be considered explicitly in this paper.

Definition 1 *A pickup and delivery route R_k for vehicle k is a directed route through a subset $V_k \subset V$ such that:*

1. *R_k starts and ends in O.*
2. *Both or neither N_i^+ and N_i^- belongs to V_k for all $i \in N$.*
3. *If both N_i^+ and N_i^- belong to V_k, N_i^+ is visited before N_i^-.*
4. *Vehicle k visits each location in V_k exactly once.*
5. *The vehicle load at any one time never exceeds Q.*
6. *The arrival time A_i and departure time D_i of any location i satisfy $D_i \in [e_i, l_i]$, where $D_i = \max\{A_i, e_i\}$* (i.e. if $A_i < e_i$, the vehicle has to wait at location i).

Definition 2 *A pickup and delivery plan is a set of routes $R \equiv \{R_k \mid k \in M\}$ such that*

1. *R_k is a pickup and delivery route for vehicle k, for all $k \in M$.*
2. *$\{V_k \mid k \in M\}$ is a partition of V.*

Define $f(R)$ as the cost of plan R corresponding to a certain objective function f. PDPTW is defined as the optimization problem of finding a pickup and delivery plan R that minimizes the function $f(R)$.

There are a wide variety of objective functions for PDPTW. In this paper, we consider the following:

1. Minimize the number of vehicles, which is almost always the most dominant part of the cost.
2. Minimize travel distance. That is, the sum of lengths of all the routes in the plan.

To model PDPTW as an integer linear program (ILP), two types of binary variables are introduced. Let $z_{ik}\ (i \in V, k \in M)$ become true iff request i is assigned to vehicle k, $x_{ijk}\ (i \in V, j \in V, k \in M)$ is true iff vehicle k is traveling from node i to node j. Let y_j

denote an intermediate variable that stores the total load of the vehicle visiting job *j*. PDPTW is to minimize $f(x)$ subject to the following constraints:

$$\forall i \in N, \sum_{k \in M} z_{ik} = 1 \quad (1)$$

$$\forall i \in V, \sum_{k \in M} \sum_{j \in V} x_{ijk} = 1 \quad (2)$$

$$\forall k \in M, \sum_{i \in V} x_{iOk} = 1 \quad (3)$$

$$\forall k \in M, \sum_{j \in V} x_{Ojk} = 1 \quad (4)$$

$$(\forall h \in V)(\forall k \in M), \sum_{i \in V} x_{ihk} - \sum_{j \in V} x_{hjk} = 0 \quad (5)$$

$$(\forall j \in V)(\forall k \in M), \sum_{i \in V} x_{ijk} Q \geq y_j \quad (6)$$

$$(\forall i, j \in V \cup O)(\forall k \in M), x_{ijk} = 1 \Rightarrow y_i + q_i = y_j \quad (7)$$

$$y_O = 0 \quad (8)$$

$$\forall i \in V, y_i \geq 0 \quad (9)$$

$$(\forall i, j \in V)(\forall k \in M), x_{ijk} = 1 \Rightarrow D_i + t_{ij} \leq D_j \quad (10)$$

$$\forall i \in N, p = N_i^+, q = N_i^-, D_p \leq D_q \quad (11)$$

$$D_O = 0 \quad (12)$$

Constraint (1) ensures that each request is assigned to exactly one vehicle. Constraint (2) ensures that each job is visited exactly once. Constraints (3) and (4) ensure that each vehicle departs from and arrives at the depot. (5) ensures that if a vehicle arrives at a node then it must also depart from that node. (6)-(9) together form the capacity constraints. The time windows and precedence constraints are ensured by (10)-(12).

To model the duo-objective of minimizing (a) the total number of vehicles and (b) total travel distance as a linear function, we multiply a coefficient for each objective and then add them together. Since the number of vehicles is more important than the total distance of a plan, the cost of each vehicle (route) is penalized with a coefficient P, which is set to be greater than the maximum possible total travel distance. Hence, the objective function of the problem is:

$$\text{minimize } P \times m + \sum_{k \in M} \sum_{i \in V} \sum_{j \in V} d_{ij} x_{ijk} \quad (*)$$

This formulation of the problem has $O(n^2 m)$ constraints and $O(n^2 m)$ variables. For large-scale problems, an ILP solver almost always experiences combinatorial explosion.

3. Literature Review

Most previous work focused on the single vehicle dial-a-ride problem with time windows (1-PDPTW). For the objective to minimize the total customer inconvenience, Psarafis ([4],[5]) developed a dynamic programming algorithm with a $O(n^2 3^n)$ time complexity, which could only solve small-sized problems with 10 or fewer requests. In Sexton and Bodin ([6],[7]), the problem was de-coupled into a coordinating routing master problem formulated as an integer program, and a scheduling subproblem for a fixed route, which

was formulated as linear program. By using a heuristic version of Benders' decomposition, the routing master problem and the scheduling subproblem were solved individually. Real problems with sizes from 7 to 20 could be solved in an average of 18 seconds of UNIVAC 1100/81A CPU time. Sexton and Choi [8] used a similar approach to minimize a linear combination of total vehicle operating time and total customer penalty due to missing any of the time windows. For minimizing the schedule duration, Van der Bruggen *et al.* [9] developed a two-phase heuristic algorithm based on arc-exchange procedures and an alternative algorithm based on simulated annealing. Their approaches produced high quality solutions on real-life problems in reasonable computational time. Finally, for minimizing the total travel cost, a forward dynamic programming approach was developed by Dumas *et al.* [10]. The efficiency of the algorithm is improved by eliminating states that are incompatible with vehicle capacity, precedence and time window constraints.

The multiple vehicle pickup and delivery problem with time windows has received few attention until recently. The only optimal algorithm to our knowledge developed by Dumas *et al.* [11] who employed a column generation scheme with a shortest path subproblem with capacity, time window, precedence and coupling constraints. Their algorithm can solve 1-PDPTW problems up to 55 paired requests and multiple- vehicle PDPTW with a small size of paired requests per vehicle.

In [12], Savelsbergh and Sol divided the General Pickup and Delivery Problem into four categories, which are Static Single-Vehicle PDP, Static Multi-Vehicle PDP, Dynamic Single-Vehicle PDP and Dynamic Multi-Vehicle PDP. He presented a general model that can handle the practical constraints. The paper aimed to isolate and discuss some of the characteristics that differentiate pickup and delivery problems from traditional vehicle routing problem,

Recently, Nanry and Barnes [3] proposed a reactive tabu search approach to minimizing the travel cost by using a penalty objective function in terms of travel time, penalty for violation of overload and time window constraints. The approach was tested on 25-customer instances, 50-customer instances and 9 100-customer instances constructed from Solomon's C1 VRPTW benchmark instances. They first used a greedy insertion method to construct a feasible PDPTW plan. Then, a reactive tabu search method was used to improve the plan. They proposed three neighborhoods, namely, *Single paired insertion (SPI)*, *Swapping pairs between routes (SBR)* and *Within route insertion (WRI)*. In their work, the data sets were built based on Solomon test cases for VRPTW. The vehicle capacity, the spatial information and time window of each location is the same as the original Solomon test cases. Jobs are paired randomly based on the optimal solution provided by [13], while assuring that feasibility was maintained. However, it is not clear from their work how the load of each pickup-delivery pair is set. This is an important consideration because it would determine whether the given solutions (from [13] in this case) are still the optimal solutions for the corresponding PDPTW test cases. In this paper, we will address this issue by proposing a more rigorous test case generation strategy.

4. Two Phase Method

In this section, we propose our two-phase method for solving PDPTW. This two-phase method comprises the **Construction heuristic** and the **Tabu Search**.

4.1 *Construction heuristics*

Our construction heuristic, which we name as *partitioned insertion heuristic,* is a hybrid heuristic combining the advantages of the standard *insertion heuristic* and *sweep heuristic*.

Insertion Heuristic

Insertion Heuristic is one of the most commonly used construction heuristics for VRP. To solve PDPTW, the Insertion heuristic can be adapted as shown below:

1. Let all vehicles have empty routes.
2. Let *L* be the list of unassigned requests.
3. Take a job pair v in *L*.
4. Insert *v* in a route at a feasible position where there is the least increase in cost.
5. Remove *v* from *L*.
6. If *L* is not empty, go to 3.

Figure 1 shows a PDPTW instance. Using the above insertion heuristic, the solution generated is shown in Figure 2. Observe that, in this instance, all the jobs close to depot (i.e. A^+, A^-, B^+, B^-) are *also* close to one another; hence, they will be served by the same vehicle. Consequently, all the jobs further away are far from one another, to the extent that each vehicle can only serve one pair of jobs at a time. In order to avoid such kind of imbalance (some very good routes and some very bad routes), another construction heuristic, the Sweep Heuristic, is often used.

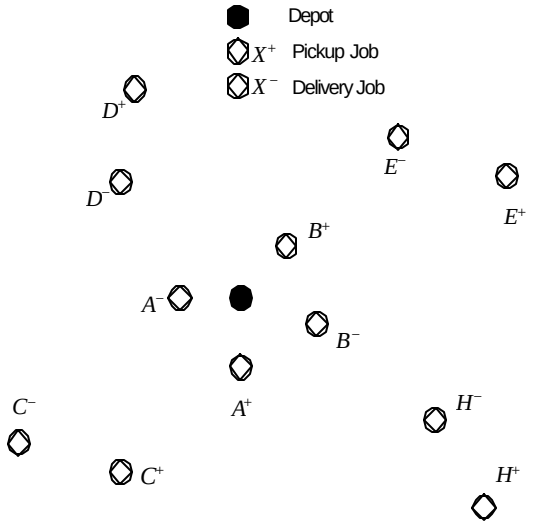

Figure 1. Example PDPTW instance

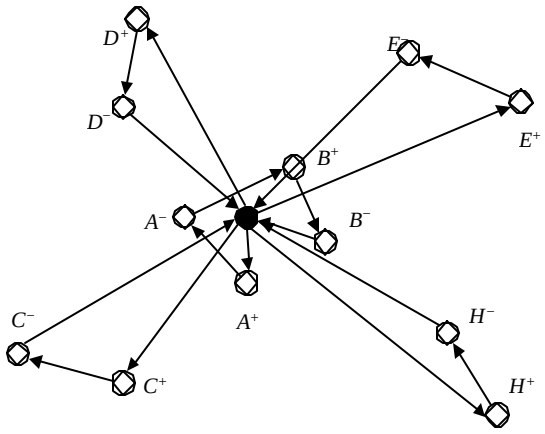

Figure 2. Solution Using Insertion Heuristic

Sweep Heuristic

Sweep heuristic is the other well-known construction heuristic method for VRP. It builds routes by a sweep technique around the depot. The Sweep heuristic for VRP is shown below:

1. Let O be a site from which vehicles leave (usually the depot), and let A (different from O) be another location, which serves as a reference.
2. Sort jobs by increasing angle $\angle AOS$ where S is the job location. Put the result in a list L.
3. The jobs in L will be allocated to the vehicles in that order as long as constraints are respected.

The advantage of sweep heuristic is that near and far jobs are mixed in the same route. This makes the solution more balanced, i.e. there are no extremely good routes and extremely bad routes. Notice that in PDPTW, geographically close destinations may have origins that are far away; consequently, a pickup and delivery pair may not be served by the same vehicle using the above algorithm! The following modifications are done to adapt the sweep heuristic for PDPTW:

1. Let O be a site from which vehicles leave, and let A (different from O) be another location, which serves as a reference.
2. Sort pickup jobs by increasing angle $\angle AOS$ where S is the job location. Put result in a list L.
3. Pick a pickup job in L with location I and its delivery job with location J and create a new route with this job pair.
4. Until no more jobs can be added the route do:
   a. If there are uninserted pickup jobs located in the sector $\angle IOJ$, insert the pair that is best feasible. Otherwise, insert an uninserted pickup-delivery job pair, in which the pickup job is at location K where $\angle JOK$ is smallest and all the constraints are respected.
   b. Remove this pickup job from L.
5. If L is not empty, go to 3.

Figure 3 shows the solution generated using the modified Sweep Heuristic.

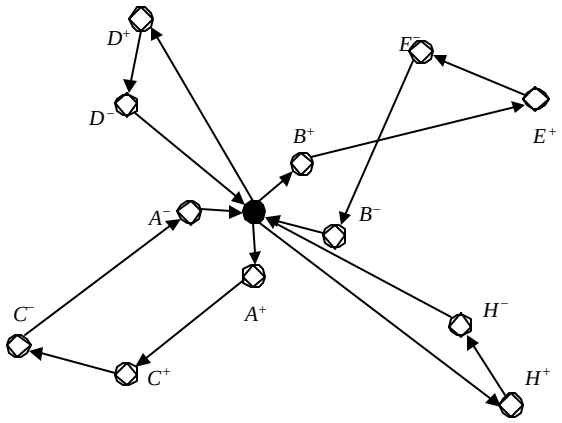

Figure 3. Solution Using Sweep Heuristic

Partitioned Insertion Heuristic

We now present our construction heuristic, the *Partitioned Insertion Heuristic*:

1. Set all vehicles to empty routes.
2. Let L be the list of unassigned visits.
3. Sort jobs by increasing angle $\angle AOS$ where S is the job location. Put the result in a list L.
4. Divide L into K sub-lists such that $\forall i \in [1, K]$, all the jobs in the ith sub-list satisfy $\angle AOS \in [i/p, (i+1)/p)$.
5. Randomly find a partition and insert the farthest job v in L.
6. Insert v in a route at a feasible position where there will be the least increase in cost.
7. Use the Insertion Heuristic to form a route.
8. If L is not empty, go to 5.

In our algorithm, both advantages of the insertion heuristic and modified sweep heuristic are merged. The furthest job within a sub-list is always selected as the first job to be inserted in a new route. This will ensure that the "bad" jobs (since they are far) are taken care of at the onset, thus avoiding the formation of imbalance routes. The number of the partition is set to the number of the established routes needed. In Section 6, we will illustrate the proposed algorithm in terms of both speed and quality of solutions.

4.2 *Tabu Search*

We introduce three different neighborhood moves, namely, **Single Pair Insertion (SPI)**, **Swap Pairs between Routes (SBR)** and **Within Routes Insertion (WRI)**. These moves are adapted from [3].

The Notion of Cluster

In [14], multiple consecutive jobs exchange was presented as a local move. This is because often a segment with consecutive jobs is a good component to forming a good route. This move is extended to PDPTW as follows. Consider a situation that pickup jobs A^+, B^+ and delivery jobs A^-, B^- are in the same route (Fig. 4). If we move both pairs together as done in [4], they must be consecutive. We define such a consecutive segment as a *cluster*. In other words, the vehicle can enter and leave the cluster with no other jobs

involved. Another property of cluster is that vehicle will enter and leave the cluster with the same load.

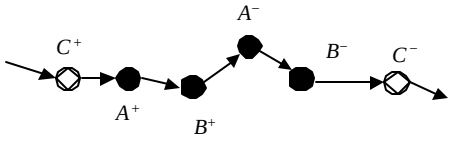

Figure 4. Example of Cluster

Single Pair Insertion (SPI)

The first move neighborhood attempts to move a pickup-delivery pair or a cluster from its current vehicle route to another vehicle route in the solution. SPI performs the following process for all *n*/2 pickup-delivery job pairs in the current solution. Once a pickup-delivery job pair or a cluster is identified, the method attempts to place it on another route. An admissible placement is one where both jobs (pickup and delivery) satisfy both time window and capacity constraints. There are *n*/2 ways of choosing pickup-delivery job pair. There are $O(n)$ positions to place the pickup and delivery jobs respectively. Hence, SPI has an $O(n^3)$ search neighborhood.

To reduce the number of routes, the search process should be biased such that it tries to remove the job pairs from the shorter routes and insert them into longer routes. Assume that a pickup-delivery job pair is selected from route r_1, and are inserted into route r_2. The routes after this move are denoted as r_1', r_2'. Originally, there are n_1 jobs in r_1 and n_2 jobs in r_2. The cost of a route *r* is denoted as $f(r)$ and the *pure saving cost* (PSC) is defined as $PSC(r_1, r_2) = f(r_1) + f(r_2) - f(r_1') - f(r_2')$. To bias the search, we introduce a new saving cost known as the *bias route saving cost (BRSC)*, defined as:

$$BRSC(r_1, r_2) = PSC(r_1, r_2) + \frac{P}{n_1/2} - \frac{P}{(n_2+2)/2}.$$

Clearly, if $n_1 < n_2$, $BRSC(r_1, r_2)$ will likely become positive; if $n_1 = n_2$, $BRSC(r_1, r_2)$ and $BRSC(r_2, r_1)$ will only depend on $PSC(r_1, r_2)$ and $PSC(r_2, r_1)$; if $n_1 > n_2$, $BRSC(r_2, r_1)$ will likely become positive.

Swapping Pair Between Routes (SBR)

The second move neighborhood involves exchanges of pickup-delivery pairs and/or clusters between two different routes. Assume that n jobs are evenly distributed in m routes, the computational time complexity of swap neighborhood is $O\left(\frac{n^4}{m^2}\right)$

Within Route Insertion (WRI)

WRI is used to improve routes by moving individual nodes forward or backward within their respective routes. Note however that since there are $(n/m)!$ possible ways to sequence the jobs, local search is used again. For each route, do the following:

1. Move one pickup and delivery pair in the route.
2. If the cost is reduced and all constraints are satisfied, goto Step 1.
3. When all such moves have been tried, move clusters consisting of two pairs.
4. Eventually, move clusters consisting of three pairs.

Composite Neighborhood

Of the three move neighborhoods, SPI has the greatest potential for improvement in the objective function, and it is the only move that can reduce the number of routes. When SPI reaches a barrier where no more admissible SPI exists, SBR is used to overcome the barrier. Finally, WRI is applied, which is especially helpful when large time windows are prevalent. The application of three neighborhood moves in our tabu search is shown below:

1. Find SPI move with highest PSC and implement the move.
2. If no more SPI move with positive PSC exists, find the best SPI move with BRSC and implement the move. Goto 1.
3. If no more SPI move with positive BRSC, find the SBR move with the greatest saving and implement the move. Goto 1.
4. If no more SBR move with positive saving, find the best WRI move and implement the move. Goto 1.
5. If no WRI move found, stop.

5. Test Case Generation

We performed a careful literature survey, and to our knowledge, no comprehensive benchmark test cases for PDPTW are available. Fortunately, from the VRPTW literature, there are well-established benchmark test cases for VRPTW by Solomon [15], as well as good solutions to those instances. In this section, we present how we adapt Solomon instances and the best-published solutions to generate good PDPTW instances and their corresponding benchmark solutions.

Our strategy is to reuse the best-published VRPTW solutions as benchmark solutions for PDPTW instances. In essence, two issues need to be resolved. First, how we ensure that a VRPTW solution remains *feasible* for the PDPTW instance, given that the latter is more constrained than a VRPTW instance. Second, given that pickup and dropoff occur throughout the route, how to ensure that the VRPTW solution remains to be *good* (in the sense of its optimality) for the PDPTW instance.

5.1 *Preserving Feasibility*

Unlike VRPTW in which jobs have no precedence constraints, a PDPTW instance does. Hence, any feasible solution y for a given VRPTW instance X may *not* be feasible on a PDPTW instance resulting from randomly or arbitrarily designating the jobs in X as pickup or delivery jobs. Hence, rather than pairing jobs on X, we pair jobs based on y. We do so in such a way that y *remains* feasible under the generated PDPTW instance, as shown below:

Algorithm GENERATE:

For each route r in y do

a. Randomly select two jobs (j_1, j_2) in r to be paired

b. Randomly select either j_1 or j_2's load as pickup and delivery load for both j_1 and j_2

c. If there are still jobs not paired

i. If the number of jobs is more than 1, go to step a.

ii. If number of jobs is 1, set it as a pickup job; create a dummy delivery job, whose time window is set to the largest possible time window, service time is set to 0, and load is equal to the load of the remaining pickup job.

5.2 *Preserving Optimality*

Unlike VRPTW where the total load (sum of job loads) on each route remains static, each route in PDPTW will have different cumulative loads as the vehicle picks up and drops off the loads throughout the route. Hence, if we keep the vehicle capacity as it is (as done in [3]), the vehicle capacity constraint is no longer as tight as intended for the given VRPTW instance. This makes it unclear whether an optimal solution for a VRPTW instance is still optimal, or there exists even better solutions for the corresponding PDPTW test case generated from the VRPTW instance. On the other hand, however, if the vehicle capacity were changed to become too tight, then the neighborhood space will be naturally limited. Hence, the key issue is to adjust the vehicle capacity so that a good (i.e. near optimal) solution for a given VRPTW instance remains to be good for the corresponding PDPTW instance.

Given a PDPTW instance and its solution, we first compute the maximum load of each route on the solution (which is the maximum possible load that the vehicle is carrying at any one point throughout the route). The vehicle capacity for *that* instance is then set as the highest maximum load over all routes.

From Algorithm GENERATE presented in Section 5.1, we see that numerous PDPTW instances can be generated from a given VRPTW instance. Hence, to ensure that the vehicle capacity is sufficiently tight over the many possible PDPTW instances derived from a given VRPTW instance, we apply the following procedure:

1. Apply Algorithm GENERATE to generate 100 different PDPTW instances and compute their corresponding vehicle capacities based on the max load argument above.
2. Compute the *average vehicle capacity* by averaging over the vehicle capacities for all instances.

Using the above approach, we present statistics on the average vehicle capacities of several R1 type test cases (rounded to the nearest integer). These figures were presented using the solutions presented in the [16] and [17]:

Test case	R103	R104	R107	R108	R109	R110	R111
Avg Vehicle Capacity	83	91	86	91	85	89	85

Table 1. Average Vehicle Capacity for PDPTW R1 instances

Notice that for Solomon's VRPTW test cases, all problem instances belonging to the same type category (R1, for example) have the *same* vehicle capacity. Likewise, to be consistent with this standard, we compute the average of all R1 test instances, which turns out to be 86.13. Hence, we set the R1 type vehicle capacity as 85 (rounded to the

nearest 5 or 10, like Solomon test cases). Likewise, the vehicle capacities for RC1, R2 and RC2 are computed, as shown in Table 2.

Test case type	R1	RC1	R2	RC2
Avg Vehicle Capacity	85 (86.13)	95 (96.07)	205 (205.55)	210 (211.38)
Min Veh Cap	46	60	109	116
Max Veh Cap	137	137	353	333

Table 2. Average Vehicle Capacities for all PDPTW Test Cases

In this table, we also list the minimum and maximum vehicle capacity computed over different PDPTW test cases within each category. We observe that even within each type category, there is a *large* gap of between the minimum and maximum vehicle capacities over all generated PDPTW instances. This implies that we should set a vehicle capacity for each test case type (R1, RC1, R2, RC2) according to its respective average vehicle capacity, in order to ensure that the problem instances generated under each type is consistently tight.

5.3 *Illustration*

We perform the following experiments to demonstrate that the hardness of PDPTW instances are sensitive to vehicle capacities. Specifically, we show that PDPTW instances with the right vehicle capacity setting are hard, in the sense that it makes an algorithm work hard.

We impose a maximum of 30 minutes on the CPU run time on our algorithm to demonstrate that, within that timing:

1. the algorithm (and conceivably any such similar algorithm) produces good solutions *very easily* when the test cases are easy; and
2. the algorithm (and conceivably any such similar algorithm) does *not* yield good performance when the test cases are hard.

In these experiments, we consider the number of vehicles as the performance metric.

For each test case, we subject it to 4 different vehicle capacity values:

a) the average capacity computed above,
b) (upper bound) the original Solomon test case capacity,
c) (lower bound) the maximum load of all jobs, and
d) an intermediate value between c) and a).

We will demonstrate that, if the value is too high (case (b)), the algorithm easily produces solutions that match best-published results, and hence says nothing about the effectiveness of the algorithm. On the contrary, if the value is set too low (case (c)), then the solution obtained is too poor relative to the best-published results for any meaningful comparison to be made.

Tables 3 and 4 give the results of the experiments. Row 1 is best-published results for the purpose of benchmarking. Rows 2 and below show the behavior of our algorithm under

the different vehicle capacities (whose specific values are given in brackets). The test instances chosen are those R-instances for which best-published solutions have been published in [16] and [17].

From the tables, we observe that the following:

1. When the capacity is the upper bound, our tabu search produces good solutions much more readily, compared with setting the vehicle capacity by other values. Particularly, when capacity is the upper bound, the algorithm yields number of vehicles that are equal to the best-published results in 8 out of 13 instances, compared with 4 out of 13, when the capacity is average. The average capacity is tight from above, in the sense that the number of vehicles obtained by the algorithm is no more than 1 greater than the best-published results (with the exception of R210).

2. On the other extreme, when the vehicle capacity is the lower bound, we observe that our tabu search algorithm produces solutions where the number of vehicles are too far from the best-published results for any meaningful benchmarking to occur. Again, the average capacity is tight from below, in the sense that for intermediate values that are below the average, the number of vehicles obtained is still poor.

Vehicle used	R103	R104	R107	R108	R109	R110	R111
Best	13	9	10	9	11	11	10
Upper bound (200)	13	10	10	10	11	12	11
Average (85)	14	11	11	10	11	12	11
Intermediate (60)	16	13	15	14	14	15	13
Lower bound (36)	22	19	20	19	21	21	17

Table 3. Testing different capacities on R1 -instances

Vehicle used	R201	R204	R205	R207	R208	R210
Best	4	2	3	2	2	3
Upper bound (1000)	4	2	3	3	2	3
Average (205)	4	3	3	3	2	5
Intermediate 1 (155)	4	3	5	4	3	5
Intermediate 2 (95)	7	5	6	4	4	5
Lower bound (36)	13	11	12	10	11	10

Table 4. Testing different capacities on R2 -instances

6 Experimental Results

In this section, we present an experimental analysis on the effectiveness of the three construction heuristics proposed, and a comparison of our approach against published PDPTW algorithms.

In our implementation, we set the penalty value for each route to be 1000, which is approximately the total distance traveled for all solutions.

6.1 Analysis of Construction Heuristics Results

Using the above test generation algorithm, we generate 4 types of test cases (R1, R2, RC1, RC2) for PDPTW, which consists of 27 test cases. We did not generate C type test cases. Instead, we used those provided by [3][1]. Each of the test cases was run 100 times against each construction heuristic and the best solutions returned were picked. The detailed result is listed in Table 5. The four columns respectively represent the results obtained by the Insertion Heuristic, Partitioned Insertion Heuristic, Sweep Heuristic and the best-published results. These are the objective values of the VRPTW solutions obtained from [16] and [17], based on the objective function (*)defined in Section 2.

Data	Best IH	Best PIH	Best SH	Best Pub.
C101	**10846.9**	10860.6	13325.7	10827.3
C104	10166.3	***10128.5***	10341	10822.9
C105	10888.7	***10860.6***	12377.6	10827.3
C106	***10867.4***	10879.2	12558.9	10822.9
C107	10934.2	***10854.4***	11226.8	10827.3
C108	10985.8	***10854.4***	12286.6	10826.1
C109	***11007***	11021	11689.6	10827.3
R103	16643	**16583.3**	18149.4	14292.67
R104	13249.8	**13239.9**	15791.5	10007.31
R107	**13304.9**	13306.4	16942.2	11104.65
R108	**12223.8**	12249.6	15814.4	9963.99
R109	15530.3	**15482.3**	18948.5	12197.41
R110	14413.3	**13377.7**	16987.4	11135.07
R111	14429.4	**14389**	17912.7	11096.72
RC103	15562.2	**14604.2**	18214.8	12261.67
RC104	13425.8	**12292.9**	14821.1	11135.48
RC107	15578	**14512.6**	17055	12230.48
R201	5874.04	***5718.64***	7483.85	5253.23

[1] During our experiments, we found some bugs for the C102 and C103 test cases reported in [NB2000]. In particular, the pickup and delivery jobs in these cases do not match. We have done some patching to ensure the correctness of the test cases.

R204	**4080.61**	4207.16	4671.28	2856.36
R205	5744.09	**5486.67**	5953.3	3998.72
R207	4321.78	**4249.99**	4654.09	2894.89
R208	4139.85	**3970.63**	4466.46	2726.82
R210	**5595.67**	5658.27	5925.34	3958.24
RC201	**6927.4**	7039.28	7693.85	5406.94
RC202	6115.65	**6031.86**	7775.59	4377.09
RC203	**5633.1**	5704.28	6196.7	4062.3
RC207	5716.38	***4514.17***	7445.28	4068.86

Table 5. Construction heuristics Results

In this table, the bold figures represent the best among the three heuristics. The Italic fonts represent that the results use the same number of vehicles in best result and the underline fonts represent that the solution use less number of vehicles than other construction heuristics. Several observations can be made.

First, it shows that while the Partitioned Insertion Heuristic yield best results in 18 out of 27 instances, the Sweep Heuristic has no good effect on the PDPTW instances. This is attributed to the time windows constraints. In fact, the tighter the time windows are, the worse the solutions obtained by Sweep.

Second, we observe that both Insertion Heuristic and Partitioned Insertion Heuristic behave quite well in C type of the test cases. Both of them can achieve the best number of vehicles. Notice also that the Partitioned Insertion Heuristic gives even better result in terms of distance traveled.

Another interesting observation is that the solution for C104 is even better than the optimal solution. This shows that the solution of test cases given by [3] is no longer optimal under the corresponding PDPTW instance. The optimality is destroyed because they did not pay attention to setting the capacity constraints appropriately.

6.2 Tabu Search Results

In this section, we will present the results produced by our tabu search approach. The construction heuristic we used is the Partitioned Insertion Heuristic. The tabu length was set to 50. Columns **Veh** and **Dist** respectively denotes the number of vehicles used and the total distance traveled. Our experimental results are listed below:

Data	**Tabu**		**NB2000**		**Best Pub.**	
	Veh	**Dist**	**Veh**	**Dist**	**Veh**	**Dist**
C101	**10**	**828.9**	10 (0 Iter.)	827.3	10	827.3
C104	**9**	**989.9**	10 (300 Iter)	834.7	10	822.9
C105	**10**	**829.8**	10 (0 Iter)	827.3	10	827.3
C106	**10**	**828.9**	10 (0 Iter.)	827.3	10	822.9
C107	**10**	**828.9**	10 (75 Iter)	826.1	10	827.3

C108	**10**	**826.1**	10 (83 Iter)	826.1	10	826.1
C109	**10**	**828.9**	10 (291 Iter)	827.3	10	827.3

Table 6. Tabu search results comparing with [3] and best-published results

Data	Tabu		Best Pub.	
	Veh	Dist	Veh	Dist
R103	**13**	**4325.2**	13	1292.67
R104	**9**	**1037.1**	9	1007.31
R107	**10**	**1214.6**	10	1104.65
R108	**9**	**964.9**	9	963.99
R109	12	1260.3	11	1197.41
R110	**10**	**1390.5**	11	1135.07
R111	11	1169.0	10	1096.72
RC103	**11**	**1540.8**	11	1261.67
RC104	**10**	**1140.3**	10	1135.48
RC107	**11**	**1300.6**	11	1230.48
R201	**4**	**1306**	4	1253.23
R204	**2**	**1004.7**	2	856.36
R205	**3**	**1267.9**	3	998.72
R207	**2**	**996.3**	2	894.89
R208	**2**	**865.1**	2	726.82
R210	4	1127.8	3	958.24
RC201	**4**	**1613.7**	4	1406.94
RC202	**3**	**1659.3**	3	1377.09
RC203	4	1092.4	3	1062.3
RC207	**3**	**1268.3**	3	1068.86

Table 7. Tabu Search results comparing with best-published results[2]

From Tables 6 and 7 above, we can see that our proposed tabu search yields solutions that are very close to the best solution for most of the cases. Considering the fact that we have paid careful attention in setting the vehicle capacities, we believe that the best-published results remain to be near optimal solutions for the PDPTW instances. This implies that our results are capable of generating near optimal solutions for PDPTW. In fact, there are 17 out of 21 non-C-type cases that attain the number of vehicles given in the best solution (as shown in bold figures). The others require exactly one vehicle more. For the C type of the cases, our approach yields solutions that are almost equal to the best-published solutions.

An observation is the results provided by [3] (NB2000 column in Table 6). In brackets are the number of the iterations that their approach required to obtain their results. There, 4 out of 8 test cases yielded the best results without *any* iteration. In other words, these

[2] No results were reported for these instances in [3].

results were obtained simply by the construction heuristic. In their paper, the authors claimed that the insertion algorithm is used as a construction heuristic. Hence, we suspect that there could be something wrong with their test cases. After a careful comparison between their test cases and the optimal solutions of VRPTW, we found that in their test cases, most pickup and delivery jobs were adjacent with each other! They were not well randomly paired and hence the problem instances became very easy to solve.

6. Conclusion

In this paper, we presented a two-phase approach to solve the Pickup and Delivery Problem with Time Windows (PDPTW). We designed a set of good (i.e. reasonably hard) benchmark test cases and solutions for PDPTW based on the full suite of Solomon test cases, thus paving the way for future PDPTW research. We conducted experimental comparisons over different construction heuristics on these data sets. Our experimental results show that our tabu search approach yields solutions that are very close to the benchmark solutions.

Reference

[1] M.W.P. Savelsbergh, "Local Search for Routing Problems with Time Windows". Annals of Operations Research, 4, 285-305, (1985).

[2] J. Desrosiers *et al*., "Time Constrained Routing and Scheduling". In Handbooks in Operations Research and Management Science: Network Routing. Elsevier Science Publ., 35-139, (1995).

[3] W.P. Nanry, J.W.Barnes, "Solving the pickup and delivery problem with time windows using tabu search". Transportation Research Part B 34, 107-121, (2000).

[4] H. Psarafis, "A dyanmic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem". Transportation Science 14, 130-154, (1980).

[5] H. Psarafis, "An exact algorithm for the single vehicle many-to-many immediate request dial-a-ride problem". Transportation Science 17 (4), 351-361, (1983).

[6] T.R. Sexton, L. D. Bodin, "Optimizing single vehicle many-to-many dial-a-ride problem with desired delivery time: I Scheduling". Transportation Science 19, 378-410, (1985).

[7] T.R. Sexton, L. D. Bodin, "Optimizing single vehicle many-to-many dial-a-ride problem with desired delivery time: II Routing". Transportation Science 19, 411-435, (1985).

[8] T.R. Sexton, Y.Y. Choi, "Pickup and delivery partial loads with soft time windows". American Journal of Mathematical and Management Science 6, 369-398, (1986).

[9] L.J.J. Van der Bruggen, J.K. Lenstra, P.C. Schuur, "Variable-depth search for the single vehicle pickup and delivery problem with time windows". Transportation Science 27, 298-311, (1993).

[10] Y. Dumas, J. Desrosiers, F. Soumis, "A dynamic programming solution of the large-scale single vehicle dial-a-ride problem with time windows". American Journal of Mathematical and Management Science 16, 301-325, (1986).

[11] Y. Dumas, J. Desrosiers, F. Soumis, "The pick-up and delivery problem with time windows". European Journal of Operational Research 54, 7-22, (1991).

[12] M.W.P. Savelsbergh, M. Sol, "The General Pickup and Delivery Problem", Transportation Science, 29, 17-29, (1995).

[13] N. Kohl, "Exact Methods for Time Constrained Routing and Related Scheduling Problems", Ph.D. Dissertation, Institute of Mathematical Modeling, Technical University of Denmark.

[14] E. Taillard *et al*., "A Tabu Search Heuristic for the Vehicle Routing Problem with Soft Time Windows", Transportation Science, 31, 170-186, (1996).

[15] M.M. Solomon, "Algorithms for the vehicle Routing and Scheduling Problem with Time Window Constraints", Operations Research, 35, 254-265, (1987).

[16] http://www.cs.strath.ac.uk/~ps/GreenTrip/NewBest/all.solns

[17] http://www.idsia.ch/~luca/macs-vrptw/solutions/welcome.htm