*Proceedings of the 9th International Conference on*
*Applied Operational Research (ICAOR 2017)*
*Chung Yuan Christian University*
*Taoyuan, Taiwan*
*18-20 December 2017*

# Job Insertion for the Pickup and Delivery Problem with Time Windows

Yi Qu [1,*], Timothy Curtois
[1] Newcastle Business School, Northumbria University, Newcastle, UK

**Abstract**

Two heuristic algorithms are proposed for a practical but relatively under-studied vehicle routing scenario. It requires the insertion of jobs into already planned routes. It occurs when new jobs arrive throughout a day but the current plans are already being performed. The benefit of solving such a problem is providing a better service for collection and delivery jobs whilst also providing better vehicle fill rates and increased revenue for delivery companies. Solutions must be generated quickly because of the dynamic nature of the problem. Two iterative heuristic algorithms are presented and tested on a well-known benchmark set. The algorithms are able to insert new jobs quickly and efficiently and even found some new best known solutions for the benchmark instances.

*Keywords*: job insertion; PDPTW; vehicle routing; heuristics

## Introduction

This paper investigates and proposes algorithms for solving a scenario arising within vehicle routing problems. The problem being solved occurs after a vehicle routing plan has already been prepared and the routes and schedules have already been assigned to vehicles and drivers. The vehicles may already have started on their routes and completed jobs already. Customers could also have been notified of arrival times and these arrival times must be met or not allowed to change significantly. The scenario being examined occurs when new collections and deliveries become available after the routes have already been prepared or while the routes are being executed. There may be an opportunity of inserting these new jobs within existing routes while minimising or avoiding disruption to existing plans. The benefit of inserting new jobs is an increase vehicle utilisation and company revenues. Increased utilisation also has environmental benefits. Solving this problem provides companies with options of developing new services or improving existing services for customers.

The vehicle routing problem being investigated in this scenario is the pickup and delivery problem with time windows (PDPTW). This is a suitable problem to use as the basis for the job insertion problem being investigated here. The reason is that collections and deliveries often become available throughout a day and many organisations would like to have these jobs completed as soon as possible. PDPTW is described in more detail in the next section. First we describe relevant previous research from the literature.

As far as we are aware very little research has previously been conducted on the job insertion problem for PDPTW. There are many heuristic and exact algorithms for solving PDPTW (e.g. Bent & Hentenryck, 2006; Kammarti, Hammadi, Borne, & Ksouri, 2004; Lu & Dessouky, 2004; Masson, Lehuédé, & Péton, 2012; Nagata & Kobayashi, 2010; Nanry & Wesley Barnes, 2000; Ropke & Cordeau, 2009; Ruland & Rodin, 1997; Venkateshan & Mathur, 2011; Xu, Chen, Rajagopal, & Arunapuram, 2003). These algorithms are unsuitable for the job insertion problem however because of

[*] *Correspondence*: Yi Qu
Newcastle Business School, Northumbria University, Newcastle, UK
yi.qu@northumbria.ac.uk

the route non-disruption requirements. These requirements mean that from the initial solution, jobs cannot be moved from one route to another and the order of existing jobs within routes cannot be changed. There are however two papers which indirectly address this problem. This problem indirectly occurs as a sub-problem within some large neighbourhood search (LNS) algorithms. The LNS algorithms for PDPTW operate by iteratively un-assigning a set of jobs from a solution and then attempting to re-assign them in a new configuration to produce an improved solution. This sub-problem is very similar to the problem being addressed here. Hence not only can the algorithm proposed here be used to solve job insertion scenarios it could conceivably also be used as a sub-problem algorithm within an LNS algorithm. The two previously published LNS algorithms for PDPTW use two very different approaches for solving the job insertion problem. Bent and Van Hentenryck (2006) use an exact based algorithm which involves bounding by finding the cost of a cost of a minimum spanning k-tree. An alternative heuristic approach is used by Ropke and Pisinger (2006) which uses a regret assignment heuristic. The regret heuristic is a method for deciding which job to insert next and where within a route. It operates in a similar manner to a greedy heuristic. The difference is that it also includes a scoring mechanism to try and avoid the short-sighted weakness of greedy heuristics. It will be explained further in the following sections because it is also used in the methods proposed in this paper.

**Problem Definition**

The pickup and delivery problem with time windows (PDPTW) will now be defined. The model is based on the model given in Curtois et al. (2017).
Parameters

$M$ set of jobs 1...$m$

$L$ set of locations 0,1…$m$ where 0 is the vehicle start location and 1…$m$ are the jobs

$P$ set of pickup jobs

$D$ set of delivery jobs

$P \cap D = \emptyset$ and $P \cup D = M$

Each pickup $p_i \in$ P is associated with a corresponding delivery $d_i \in$ D. Let:

$t_{ij}$ the travel time between locations $i$ and $j$

$d_{ij}$ the distance between locations $i$ and $j$

$s_i$ the service duration for location $i$

$e_i$ the earliest time at which the service at location $i$ can start

$l_i$ the latest time at which the service at location $i$ must start

Constraints

A route of length $n$ is $\langle v_0, v_1 \ldots v_n, v_{n+1} \rangle$ where $v_0$ and $v_{n+1}$ are the start and end location and visits $v_1 \ldots v_n$ are jobs. The pairing and precedence constraints mean that if a route contains a pickup $p_i$ then it must also contain its corresponding delivery $d_i$ (and vice-versa) and $p_i$ must precede $d_i$

Each pickup $p_i$ has a nonnegative demand $q_i$ and the corresponding delivery $d_i$ has the demand $-q_i$.

The load $c_{v_i}$ carried by a vehicle $v$ at a visit $i$ is

$$c_{v_i} = \sum_{j=1}^{i} q_{v_j} \qquad (1)$$

All vehicles have the same capacity $Q$ and the load must never exceed the vehicle capacity

$$c_{v_i} \leq Q \quad \forall v_i \in \{1 \dots n\} \tag{2}$$

The start time $b_v$ for each visit in the route is calculated as

$$b_{v_0} = 0$$
$$b_{v_i} = \max\{b_{v_{i-1}} + s_{v_{i-1}} + t_{v_{i-1}v_i}, e_{v_i}\} \quad \forall i \in \{1 \dots n\} \tag{3}$$

The services within a route must start before or at a location's service start time

$$b_{v_i} \leq l_{v_i} \quad \forall\, v_i \in \{0 \dots n\} \tag{4}$$

Objectives

The objective is to minimise the sum of the distances of all routes where the distance of a route of length *n* is given by

$$\sum_{i=0}^{n} d_{v_i,v_{i+1}} \tag{5}$$

In the problem we are solving there are existing routes with jobs allocated to each route. The problem is to insert new jobs into the existing routes according to this objective and respecting constraints 1. to 4. In other words, for each new job we must decide which route to insert it into and at which position.

**Methods**

Two algorithms are proposed for the problem. These two heuristics were chosen because they are fast, relatively simple and yet shown in other publications to be successful for other vehicle routing problems. The main difference between the two heuristics is the way in which they select the next job to insert. The first one uses a greedy heuristic and the second uses a regret heuristic. The heuristics are used within an iterative procedure to assign available, un-assigned jobs. Each time the heuristics are applied random bias is also introduced. Without the bias in selecting which jobs to insert, exactly the same solution would be produced at each iteration because the heuristics are not stochastic. Instead of simply choosing the best option according to the heuristic at each decision point, the second, third or fourth best options are occasionally chosen instead. The probabilities used for selecting which next insertion to make are set as: 1st best: 0.5, 2nd best: 0.25, 3rd best: 0.15, 4th best: 0.1.
When deciding which job to insert next the greedy heuristic ranks all the feasible insertions for each job into each route and selects the job insertion with the lowest cost. The regret heuristic attempts to overcome this short-sighted approach by also taking into account the second, third, fourth up to the kth best insertions for a job. If the second kth best insertions are significantly worse than the first best insertion for a job, the heuristic encourages inserting that job into its first best possible position while it is still available. The regret heuristic does this by calculating a score for each job's insertion. The score is the total of the differences between the cost of the first best insertion and the second best insertion, the difference between the best and third best, up to, the difference between the best and kth best. Where k is an input parameter. These scores are then ranked and the insertion with the highest score is made next. The advantage of these heuristics is that they are very fast. This is because at each insertion iteration it is not necessary to re-calculate all possible insertions because if a route has not changed (i.e. the last inserted job was not in this route) then the insertion cost will be the same as at the previous iteration. The iterated regret heuristic algorithm tries all k values 2,3,4,5. Note that the greedy heuristic is the same as the regret heuristic but with k=1. Both algorithms are run for a maximum number of iterations or a fixed time limit, whichever is reached first. In our tests we set the maximum time as one or ten seconds and the maximum iterations as 100000.
The pseudocode for the algorithms are given in Figure 1 and Figure 2.

```
1.  SET BestSolution := null
2.  FOR 1..MAX ITERS OR TIME LIMIT
3.      Create NewSolution containing initial jobs
4.      UNTIL no feasible job insertions in NewSolution
5.          FOR each job (j1) available for insertion in NewSolution
```

```
6.              FOR each existing route (r1) in NewSolution
7.                  Calculate and record best position for inserting j1 into r1
   and record
                    change in objective function
8.              END FOR
9.              From all possible insertions do best job insertion according to
   greedy
                heuristic with random bias
10.         END FOR
11.     END UNTIL
12.     IF NewSolution is better than BestSolution
13.         SET BestSolution := NewSolution
14.     END IF
15. END FOR
16. RETURN BestSolution
```

Figure 1. Iterated Greedy Heuristic

```
1.  SET BestSolution := null
2.  FOR 1..MAX ITERS OR TIME LIMIT
3.      Create empty NewSolution containing initial jobs
4.      UNTIL no feasible job insertions in NewSolution
5.          FOR each job (j1) available for insertion in NewSolution
6.              FOR each existing route (r1) in NewSolution
7.                  Calculate and record best position for inserting j1 into r1
   and record
                    change in objective function
8.              END FOR
9.              From all possible insertions do best job insertion according to
   regret
                heuristic with random bias
10.         END FOR
11.     END UNTIL
12.     IF NewSolution is better than BestSolution
13.         SET BestSolution := NewSolution
14.     END IF
15. END FOR
16. RETURN BestSolution
```

Figure 2. Iterated Regret Heuristic

**Results**

To test the algorithms the commonly used benchmark dataset[1] of Li and Lim (Li & Lim, 2003) were used. The 354 instances are categorised into six groups of different sizes. The groups range from 50 jobs up to 500 jobs. Each group is sub-grouped into clustered location, randomly distributed location and randomly clustered location instances. The sub groups are then further categorized into instances with short planning horizons and long planning horizons. This provides a varied and challenging test suite

The following method was used replicate the scenarios that the algorithms are designed for. First the instances were solved using an existing algorithm. To solve the instances the hybrid LNS+GES method of Curtois et al. (2017) was used. A fixed number of jobs was then randomly selected and removed from each solution. The jobs were then attempted to be re-inserted using the insertion algorithms. The solution with the re-inserted jobs were then compared with the original solution. When comparing against the original solution there are three possible results. 1. The algorithm was not able to re-insert the jobs back into the solution as efficiently as they were originally allocated. This means that either not all the jobs could legally be inserted back again or the total distance is higher than the original solution. 2. The insertion algorithm was able to re-insert the jobs in the same configuration as they were originally assigned. 3. The insertion algorithm was able to insert the jobs back in a better configuration than in the original solution. Although option 3. is possible it would be unexpected because the original solutions are already good solutions.

[1] Available from http://www.sintef.no/projectweb/top/pdptw/li-lim-benchmark/

Five tests were performed on each instance for each algorithm. In each test a different number of jobs were removed. The percentage of jobs removed for each test were selected as [5%,10%,20%,30%,40%]. The same set of jobs was removed for both algorithms on each instance at each percentage level. Each algorithm was allowed to run for a maximum of 100,000 iterations or 1 second (whichever was reached first). The experiments were then repeated but allowing 10 seconds instead (again the same set of jobs removed). The short run time was selected to allow very fast decisions to be made. In rapidly changing situations where for example the schedule is already being executed longer computation times are not practical. This is also the feedback we had received from industry.
All experiments were performed on an Intel Xeon CPU E5-1620 @ 3.5 GHz using a single core for each test. The results are summarised in Table 1 and Table 2. Table 1 lists the number of instances on which the algorithm inserted the jobs to equal the original solution objective function value (S*ame*), the number of times the solution was worse than the original (*Worse*) and the number of times it was an improvement on the original solution (*Better*). It also lists how many times all the jobs were inserted, how many times they were not all inserted and finally the percentage of the jobs that were not assigned when not all were assigned.

| | % jobs removed for re-insertion | Same | Worse | Better | Instances all jobs assigned | Instances not all jobs assigned | Avg % of available jobs not assigned |
|---|---|---|---|---|---|---|---|
| Greedy (1s) | 5 | 312 | 19 | 23 | 347 | 7 | 5.1 |
| | 10 | 256 | 76 | 22 | 322 | 32 | 3.4 |
| | 20 | 130 | 213 | 11 | 227 | 127 | 3.2 |
| | 30 | 74 | 277 | 3 | 166 | 188 | 5.4 |
| | 40 | 40 | 314 | 0 | 118 | 236 | 8.8 |
| Regret (1s) | 5 | 312 | 17 | 25 | 347 | 7 | 5.1 |
| | 10 | 255 | 78 | 21 | 320 | 34 | 3.5 |
| | 20 | 129 | 216 | 9 | 225 | 129 | 3.3 |
| | 30 | 75 | 277 | 2 | 162 | 192 | 5.4 |
| | 40 | 41 | 313 | 0 | 113 | 241 | 8.7 |
| Greedy (10s) | 5 | 314 | 14 | 26 | 348 | 6 | 5.3 |
| | 10 | 261 | 64 | 29 | 327 | 27 | 2.9 |
| | 20 | 144 | 199 | 11 | 247 | 107 | 2.9 |
| | 30 | 83 | 267 | 4 | 177 | 177 | 4.9 |
| | 40 | 48 | 306 | 0 | 129 | 225 | 8.3 |
| Regret (10s) | 5 | 314 | 14 | 26 | 348 | 6 | 5.3 |
| | 10 | 261 | 65 | 28 | 328 | 26 | 3.4 |
| | 20 | 137 | 205 | 12 | 246 | 108 | 3.0 |
| | 30 | 80 | 272 | 2 | 173 | 181 | 4.9 |
| | 40 | 46 | 308 | 0 | 129 | 225 | 8.3 |

Table 1 Experiment results

The results show only a small difference between the two algorithms. When there are less jobs available to insert, both algorithms are usually able to insert all jobs within a short computation time. When there are a large number of jobs to insert, for example, 40% of the jobs within the total instance, then the algorithms are not able always insert all the jobs again. When all the jobs are inserted the solution is often equal to the original solution. For the smaller number of jobs to insert the original solution is often matched or sometimes even improved upon. For the larger number of jobs available to insert, the original solution is not always matched. When all the jobs cannot be re-inserted, it is on average a small number percentage of the un-assigned jobs that cannot be re-inserted (less than 10% even on the largest instances). Nine new best known solutions were also found. These were actual new best solutions found by the insertion algorithms because the initial solutions were not new best knowns. They are listed in Table 3.

| | 1s | 10s |
|---|---|---|

| | | |
|---|---|---|
| Regret better | 346 | 283 |
| Greedy better | 324 | 344 |
| Equal | 1100 | 1143 |

Table 2 Algorithm Comparison

| Instance | Vehicles | Distance |
|---|---|---|
| LR1_6_8 | 18 | 12254.62 |
| LR2_6_3 | 7 | 19181.63 |
| LR2_6_7 | 6 | 15525.57 |
| LRC2_8_3 | 14 | 21622.66 |
| LRC2_8_5 | 16 | 24404.65 |
| LR1_10_4 | 28 | 31616.74 |
| LR2_10_6 | 11 | 53050.39 |
| LRC1_10_2 | 72 | 45282.42 |
| LRC2_10_10 | 11 | 31324.94 |
| LRC2_10_2 | 20 | 33574 |
| LRC2_10_4 | 11 | 26236.79 |

Table 3 New best knowns

## Conclusion

Two fast heuristic algorithms have been proposed for job insertion for the pickup and delivery problem with time windows. The algorithms are designed for solving problem scenarios that occur when new jobs become available after an initial plan has been created and assigned. For operational reasons the initial plan cannot change. This means that jobs cannot be moved to different routes and the ordering of jobs within a route cannot change. New jobs can be inserted within an existing plan though. The scenario could also occur as the plan is being executed. The algorithms were tested on benchmark data sets. The results showed that the two algorithms perform similarly. The greedy heuristic may be preferable because of its simpler implementation but both algorithms were able to produce satisfactory solutions. This was emphasised by the algorithms finding nine new best known solutions for the benchmark instances. For future work there are possible options. For example, there are still instances where the results are worse. It would be interesting to evaluate the differences between the solutions to try and improve the proposed algorithms.

## References

Bent, R., & Hentenryck, P. V. (2006). A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows. *Computers & Operations Research, 33*(4), 875-893. doi:http://dx.doi.org/10.1016/j.cor.2004.08.001

Curtois, T., Landa-Silva, D., Qu, Y., & Laesanklang, W. (2017). Large neighbourhood search with adaptive guided ejection search for the pickup and delivery problem with time windows. *Under review*.

Kammarti, R., Hammadi, S., Borne, P., & Ksouri, M. (2004, 10-13 Oct. 2004). *A new hybrid evolutionary approach for the pickup and delivery problem with time windows.* Paper presented at the Systems, Man and Cybernetics, 2004 IEEE International Conference on.

Li, H., & Lim, A. (2003). A Metaheuristic for the Pickup and Delivery Problem with Time Windows. *International Journal on Artificial Intelligence Tools, 12*(02), 173-186. doi:10.1142/S0218213003001186

Lu, Q., & Dessouky, M. (2004). An Exact Algorithm for the Multiple Vehicle Pickup and Delivery Problem. *Transportation Science, 38*(4), 503-514. doi:10.1287/trsc.1030.0040

Masson, R., Lehuédé, F., & Péton, O. (2012). An Adaptive Large Neighborhood Search for the Pickup and Delivery Problem with Transfers. *Transportation Science, 47*(3), 344-355. doi:10.1287/trsc.1120.0432

Nagata, Y., & Kobayashi, S. (2010). Guided Ejection Search for the Pickup and Delivery Problem with Time Windows. In P. Cowling & P. Merz (Eds.), *Evolutionary Computation in Combinatorial Optimization: 10th European Conference, EvoCOP 2010, Istanbul, Turkey, April 7-9, 2010. Proceedings* (pp. 202-213). Berlin, Heidelberg: Springer Berlin Heidelberg.

Nanry, W. P., & Wesley Barnes, J. (2000). Solving the pickup and delivery problem with time windows using reactive tabu search. *Transportation Research Part B: Methodological, 34*(2), 107-121. doi:http://dx.doi.org/10.1016/S0191-2615(99)00016-8

Ropke, S., & Cordeau, J.-F. (2009). Branch and Cut and Price for the Pickup and Delivery Problem with Time Windows. *Transportation Science, 43*(3), 267-286. doi:10.1287/trsc.1090.0272

Ropke, S., & Pisinger, D. (2006). An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows. *Transportation Science, 40*(4), 455-472. doi:10.1287/trsc.1050.0135

Ruland, K. S., & Rodin, E. Y. (1997). The pickup and delivery problem: Faces and branch-and-cut algorithm. *Computers & Mathematics with Applications, 33*(12), 1-13. doi:http://dx.doi.org/10.1016/S0898-1221(97)00090-4

Venkateshan, P., & Mathur, K. (2011). An efficient column-generation-based algorithm for solving a pickup-and-delivery problem. *Computers & Operations Research, 38*(12), 1647-1655. doi:http://dx.doi.org/10.1016/j.cor.2011.02.009

Xu, H., Chen, Z.-L., Rajagopal, S., & Arunapuram, S. (2003). Solving a Practical Pickup and Delivery Problem. *Transportation Science, 37*(3), 347-364. doi:10.1287/trsc.37.3.347.16044