On Open Vehicle Routing Problem with Soft Time Windows and Tabu Search

DUAN Fenghua1, HE Xiaonian2
1. School of the traffic & transport engineering, Central South University
2. School of Electric & Information, hunan International Economics University, P. R. China, 410000
Duanfenghua.2020@yahoo.com.cn

Abstract: As a typical NP-hard problem, open vehicle routing problem with soft time windows (OVRPSTW) is a kind of basic open vehicle routing problem with time windows. By setting fit neighbourhood structure, evaluation solutions, tabu list, and stopping criterion, author constructed a tabu search (TS) to solve the OVRPSTW. TS shows its good computational performance by testing benchmark problems.

Keywords: Vehicle Routing Problem; Open Vehicle Routing Problem; Soft Time Windows; Tabu Search

1 Introduction

OVRP is a kind of vehicle routing problem that does not require the vehicle to return the depot after its completing delivery, or if return depot then run along the original travel route to return[1]. OVRP can be further divided into different types in according to different restrictive conditions of the problem such as Capacitated OVRP, OVRP with time windows, Distance-constrained and Capacitated OVRP, OVRP with pikup & delivery, OVRP with pikup & delivery and time windows.

2 Literature Review

Because OVRP is a NP-hard problem such as VRP, the emphasis and difficulty of the OVRP study still focused on the algorithm. For OVRP, although some relevant cases study and description has been taken in the past, theoretically systematic study only just begin. Sariklis and Powell took the lead on studying COVRP by proposing a minimum spanning tree which includes two-stage heuristics. Then, Fu Z put up a TS. The Tarantilis proposed threshold accepting algorithm. Letchford constructed a precision algorithm based on branch-and-cut. For DCOVRP, Fu Z et al and Brandão et al constructed a TS respectively. [2] For OVRP with hard time windows, Repoussis constructed a forward greedy method.[3] Zhong Shiquan constructed a GA.[4] This paper constructs an advanced TS to solve the OVRPSTW.

3 Open Vehicle Routing Problem with Soft Time Windows

OVRPSTW means that a group of vehicles depart from depot to visit clients whose demands and service time windows are known, and allow the beginning service time is not in clients’ service time windows but some penalty must be paid. The vehicle doesn’t return depot after completing its work. The question is how to design vehicles’ travel routes to minimize the number of vehicles, the travel cost and the deviation from the service time windows of clients on condition that meeting vehicle’s capacity and travel distance constraints, and so on. Generally speaking, minimizing number of vehicles is the goal above of all, then minimizing total cost and deviation from the service time windows is second goal.

Set depot for the No.0, the number of clients respectively is 1,2,…, N, and variables are defined as follows:

\[
    x_{ij} = \begin{cases} 
    1 & \text{vehicle K travel to client j from client i} \\
    0 & \text{otherwise} 
    \end{cases}
\]
4 Notation

$K$--the number of vehicles in demand; $g_i$--goods demand of client $i$; $q$--the maximum capacity of vehicle; $L$--the maximum travel distance of vehicle; $a_i$--the earliest time window of client $i$; $b_i$--the latest time window of client $i$; $t_{ij}$--the travel time from client $i$ to client $j$; $t_i$--the time of arrive at client $i$ and begin to service earlier than $a_i$; $d$--the punishment coefficient that vehicle arrive at client $i$ and beggin to service late than $b_i$; $t_{ij}=t_i+s_i+t_{ij}$ if client $i$ and client $j$ in the same route, and the service to client $j$ is just after client $i$.

The minimal number of required vehicles could be estimated according to the client demand and vehicle capacity, of which $[*]$ means largest integer values that is not more than the values in the bracket.

$$\left\lceil \sum_{i=1}^{N} \frac{g_i}{q} \right\rceil + 1$$

Therefore, the OVRPSTW can be modelled as follows:

$$\min \{K\}, \quad \min \left\{ Z = \sum_{i=0}^{N} \sum_{j=1}^{K} t_{ij} x_{ijk} + d \sum_{i=1}^{N} \max(a_i-t_i,0) + e \sum_{i=1}^{N} \max(t_i-b_i,0) \right\}$$

s.t. \hspace{1cm} \sum_{j=0}^{N} \sum_{k=1}^{K} x_{ijk} \leq L, i=0, k \in \{1,2,\ldots,K\} \hspace{1cm} \text{(3)}

\hspace{1cm} \sum_{j=0}^{N} \sum_{k=1}^{K} x_{ijk} = 1, i \in \{1,2,\ldots,N\} \hspace{1cm} \text{(4)}

\hspace{1cm} \sum_{i=0}^{N} \sum_{j=1}^{N} x_{ijk} = 1, j \in \{1,2,\ldots,N\} \hspace{1cm} \text{(5)}

\hspace{1cm} \sum_{i=0}^{N} \sum_{j=1}^{N} g_i x_{ijk} \leq q, k \in \{1,2,\ldots,K\} \hspace{1cm} \text{(6)}

where $x_{ijk}=0\text{ or }1$, $i=0,1,\ldots,N$, $j=1,2,\ldots,N$.

There, constraints (1) presents that minimizing the number of vehicles is first optimal objective; constraints (2) presents that minimizing cost and deviation from the time windows is second optimal objective; constraints (3) presents constraints to run distance of route; constraints (4) & (5) presents visiting any clients only once; constraints (6) presents the constraints of vehicle’s capacity.

5 The tabu search (TS) heuristic

5.1 Introduction to tabu search heuristic

As a heuristic method designed to guide other methods, including local search algorithms, to escape local optima, TS was first introduced by Hred Glover in 1986 and since has been used to solve many practical applications. TS introduces a tabu list to forbid certain moves which would allow the search to return to a previous solution and become trapped in a local optimum. The basic idea of TS conclude: beginning with an initial solution; creating a neighbourhood of the current solution through different classes of moves at each iteration; then, selecting the best admissible solution in the neighbourhood as the new current solution, and repeating the procedure until a stopping criterion being satisfied; admitting a move if it not being tabu, and if it being tabu, then still admitting it if the move produces a solution strictly better than the best solution so far. In this paper, an advanced TS heuristicis is proposed for the OVRPSTW.

5.2 Initial solution

An initial solution is required for any TS algorithm to start the local search process. In this paper,
for the initial solution, besides generating randomly, by exploiting the problem structure, we propose an optimization-based farthest first heuristic (FFH) for it. In the OVRPSTW, because the vehicle routes are open ones, it is often the case that in the best solutions, the farthest customer from the depot is at the end of a route, so our proposed new routes are always formed from the farthest unrouted customer from the depot. A new route starts from the farthest unrouted customer \( i \) from the depot. Along the shortest route, back from \( i \) to the depot, add customers to this new route until the vehicle is full enough. If the vehicle is not sufficiently full, the route is rejected and the next shortest route is considered instead. The process is repeated until a route is accepted.

**Step 0:** Initialisation. Find the shortest distance from the depot to every customer.

**Step 1:** Starting new route from the farthest unrouted customer \( i \). Choose the farthest customer \( i \) whose demand is not assigned to any vehicle.

**Step 2:** Adding other customers to this new route until the vehicle is full enough.

Set \( k = 0 \)

Repeat

Increment \( k \)

Testing: along the \( k \)th shortest route from \( i \) back to the depot, add those customers \( j \) on this route one by one to this vehicle so that both \((d + \sum d_j \leq C)\) and \((\text{total length of the route} \leq L)\) hold, until no more can be added;

If \([\text{(vehicle capacity left } < \min \{(\text{any unassigned } d, (C-C_{\text{vag}})}) \text{ or (no demands left)})\] Then accept the route;

Until a route is accepted.

**Step 3:** Accept the new route. Update the data accordingly; if all customers are routed then end of the FFH else go to step 1.

The strategy of not starting a new route until the vehicle currently considered is full enough aims for a solution that keeps the number of vehicles needed to a minimum.

### 5.3 Neighbourhood structure

In our implementation, different classes of neighbourhood moves are applied to the current solution. These moves are based on: (a) **Vertex Reassignment** Remove the first selected vertex from its current position on the route and insert it into the position before (after-for pick up problem), the second selected vertex, that is \( X_1 = (013560479028) \rightarrow X_2 = (015604790238) \); (b) **Vertex Swap**. Exchange the positions of two selected vertices, that is, \( X_1 = (013560479028) \rightarrow X_2 = (013546079028) \); (c) **2-opt**. Reverse the order of all elements between two selected vertices like the standard 2-opt move in TSP, if two selected vertices are within the same route, that is, \( X_1 = (013564079028) \rightarrow X_2 = (013546079028) \); (d) **Tails Swap**. Exchange the ‘tails’ after two selected vertices (from the selected vertex to the end of the route; both vertices must be customers), if two selected vertices are in different routes, that is, \( X_1 = (013560479028) \rightarrow X_2 = (013790456028) \). This neighbourhood structure is able to allow moving infeasible solutions in terms of the vehicle capacity or route length, can enhance the TS mechanism and may result in better improvement on both travel costs saving and eliminating an existing route.

### 5.4 Evaluation of solutions

A feasible solution with a certain number of vehicles always dominates over any other feasible solutions requiring more vehicles. For those solutions with the same number of vehicles required, the one with minimum total travelling cost is selected. To facilitate the exploration of the search space, a move is allowed even if it results in an infeasible solution. The extent of the infeasibility can be measured by incorporating the vehicle capacity and maximum route length constraints into the objective function by adding a penalty if the constraints are broken. If the time windows is deviated, then set punishment coefficient \( d = e = 100 \); If the deviation is in the time windows \([a, b]\), then set punishment coefficient \( d = e = 0 \). 297
5.5 Tabu list

The tabu list contains the move attributes of solutions during the last 5-10 (selected randomly) iterations. A set of \((n+1) \times (n+1)\) matrices can be constructed for there cord of tabu status vertex \(i\) is selected for the move (a), Vertex Reassignment, the tabu status is saved in the element \((i, i)\) of a matrix. If vertices \(i\) and \(j\) are selected for the moves (b)-(d), the tabu status is saved in the element \((i, j)\) of a matrix. At each iteration, the tabu status of the last move performe disadded to the list while the others are decreased by one until equal to zero.

5.6 Stopping criterion

The search is terminated if either a specified number of iterations has elapsed in total or since the last best solution was found. The following variables are used in the description of the TS heuristic: \(\text{iter}\) current number of iterations; \(\text{max}\_\text{iter}\) maximum number of iterations; \(\text{cons}\_\text{iter}\) current number of consecutive iterations without any improvement to the best solution so far; \(\text{max}\_\text{cons}\_\text{iter}\) maximum number of consecutive iterations without any improvement to the best solution so far; \(\text{cand}\_\text{list}\) current number of candidate moves on the list; \(\text{max}\_\text{cand}\_\text{list}\) maximum number of candidate moves on the list.

5.7 TS heuristic

Generate an initial feasible solution randomly or by the FFH and set this solution as the current solution and the best solution so far;
Set and \(\text{cons}\_\text{iter}\) to 0;
While(\(\text{iter} \leq \text{max}\_\text{iter}\)) and (\(\text{cons}\_\text{iter} \leq \text{max}\_\text{cons}\_\text{iter}\)) do

Begin

While (\(\text{cand}\_\text{list} \leq \text{max}\_\text{cand}\_\text{list}\)) do

Begin

Select two vertices randomly;
Select one of the four types of neighbourhood move randomly;
Add the solution produced by the selected move to the candidate list;
End.

Select the best solution in the candidate list if it is not tabu, or it produces a solution strictly better than the best solution so far;
Set the new solution as the current solution, update the tabu list and increment \(\text{iter}\);
If the new solution improves the best solution so far, update the best solution so far, and set \(\text{cons}\_\text{iter}\) to 0; Otherwise, increment \(\text{cons}\_\text{iter}\).
End.

The distinctive feature of this TS heuristic is the use of a simple but powerful neighbourhood structure. The penalty function approach allows the search process to examine solutions that may be infeasible with respect to the capacity and duration constraints.

6 Benchmark Problems Computational Results and Comparison

C101…C105 of the Benchmark Problems have been computed to realize the OVRPSTW on Pentium IV 1.80GHZ by delphi programming language. The test result only is compared with Repoussis et al OVRPHTW.[3] The comparison is shown in table 1:
<table>
<thead>
<tr>
<th>Problems</th>
<th>Repoussis /OVRPHTW[12] K(min) Run Distance</th>
<th>OVRSTW K(min) Run Distance</th>
<th>Saving</th>
</tr>
</thead>
<tbody>
<tr>
<td>C101</td>
<td>10 709.71</td>
<td>9 860.57</td>
<td>-21.27%</td>
</tr>
<tr>
<td>C102</td>
<td>10 1036.98</td>
<td>10 900.46</td>
<td>13.16%</td>
</tr>
<tr>
<td>C103</td>
<td>10 1146.89</td>
<td>10 892.75</td>
<td>22.15%</td>
</tr>
<tr>
<td>C104</td>
<td>10 907.08</td>
<td>10 907.01</td>
<td>0%</td>
</tr>
<tr>
<td>C105</td>
<td>10 695.08</td>
<td>10 913.08</td>
<td>-31.36%</td>
</tr>
</tbody>
</table>

Note: Thanks for Repoussis’ providing C101, C102, C103, C104, C105 results.

7 Conclusions

Compared with Repoussis’ results on solving OVRPHTW, conclusion is available that OVRPSTW can save number of vehicles (C101), or route length at large, and the average route length reduced 0.99 percent (C102-C105).

References