Effective Solutions for the Split Delivery Vehicle Routing Problem with Time Windows

Nathalie Helal and Haidar Harmanani
Department of Computer Science and Mathematics
Lebanese American University
Byblos, 1401 2010, Lebanon

Abstract

The Split Delivery Vehicle Routing Problem with Time Windows (SDVRPTW) is a variation of the *vehicle routing problem* (VRP) that incorporates *time windows* and *split delivery constraints*. The VRP is a generalization of the traveling salesman problem, and was initially proposed in order to find the optimum routing of a fleet of gasoline delivery trucks between a bulk terminal and a large number of service stations supplied by the terminal. In this paper, we present an efficient method for solving the SDVRPTW using simulated annealing. Experimental results are presented and favorable results comparisons are reported.

1 Introduction

The Split Delivery Vehicle Routing Problem with Time Windows (SDVRPTW) is a variation of the vehicle routing problem that determines a set of least-cost routes in order to service the delivery or collection of goods. A customer is served by one or several vehicles while respecting the vehicle capacity and the customers requested service time windows. The demand of each customer maybe split and consequently fulfilled by several vehicles leading to a decrease in transportation cost [1]. Various approaches have been proposed to solve the vehicle routing problem, and a comprehensive survey can be found in [2]. Other researchers proposed metaheuristic solutions such as simulated annealing [3], tabu search [4, 5], honey bees mating [6], genetic algorithms [7, 8], and ant colony optimization [9].

The SDVRPTW has not received a lot of attention in the literature although both split deliveries and time windows have been tackled in isolation. For example, Archetti *at al.* [10, 11] solved the split delivery vehicle routing problem (SDVRP) using tabu search. Campos *et al.* [13] proposed a scatter search algorithm while Desaulniers *et al.* [12] tackled the problem using a branch-and-price-and-cut method based on a column generation method. The approach generates feasible results based on the implicit objective of minimizing the number of vehicles. Chang *et al.* [14] proposed a solution for the VRPTW using a hybrid genetic algorithm. Gambardella *et al.* [15] proposed an ant colony optimization approach for the VRPTW. The algorithm was validated using the traveling salesman problem. Belfiore *et al.* [16] presented a scatter search algorithm for the SDVRPTW. The procedure generates first a set of diverse solutions by means of a random procedure intended for diversification. Hoet *et al.* [17] proposed a tabu search-based heuristic in order to solve the vehicle routing problem with time windows and split deliveries. Frizzell *et al.* [18] proposed a construction and improvement heuristic to solve the SDVRPTW.

This paper presents a simulated annealing algorithm for solving the SDVRPTW, a combinatorial optimization problem with no known efficient exact solution [20]. The proposed algorithm combines the global search ability of simulated annealing with problem knowledge information in order to provide competitive solutions. The remainder of the paper is organized as follows. Section 2 describes the SDVRPTW problem while Section 3 formulates the *annealing SDVRPTW* problem and describes the *configuration, cooling schedule, the neighborhood transformations*, and the cost function. Section 4 presents the *annealing SDVRPTW* algorithm. Finally, *experimental results* are presented in section 5.

2 Problem Description

Given an undirected graph $G = (V, E)$ where vertex v_0 denotes the *depot* and each vertex v_i has a non-negative demand of goods q_i. Each edge is associated with a travel time c_{ij} between customers v_i and v_j and a service time s_i which is the time needed to drop the goods at the customer. An edge is also associated with a time window interval $[r_i, d_i]$ where r_i characterizes the ready time for v_i and d_i defines the due time for v_i. Vertex v_i cannot be serviced before time r_i and after time d_i. At v_0, m identical vehicles are placed; each vehicle of capacity C must serve all the

978-1-880843-95-6 / copyright ISCA, CATA 2014
March 24-26, 2014, Las Vegas, Nevada, USA

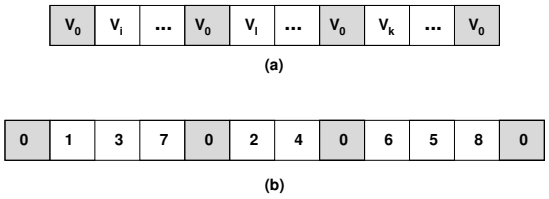

Figure 1: (a) Configuration Representation, (b) Sample Configuration

cities or customers, represented by the set of n vertices $\{v_1, \ldots, v_n\}$. The SDVRPTW [19] aims at determining a set of least cost vehicles routing solutions such that: (i) each route visits the depot; (ii) each customer can be visited by more than one vehicle; (iii) demand can be greater than a vehicle capacity and can be split; (iv) all customer demands are met; (v) for each customer i, the service starts within the time window $[a_i, b_i]$ and the vehicle stops for s_i time instants.

2.1 Simulated Annealing

Simulated annealing is a global stochastic method that is used to generate approximate solutions to combinatorial problems. The annealing algorithm begins with an initial feasible configuration and proceeds to generate a neighboring solution by perturbing the current solution. If the cost of the neighboring solution is less than that of the current solution, the neighboring solution is accepted; otherwise, it is accepted or rejected with probability $p = e^{-\frac{\Delta_C}{T}}$ where T is the *temperature* and Δ_C is the change in cost between the neighboring solution. The set of parameters controlling the initial temperature, stopping criterion, temperature decrement between successive stages, and the number of iterations for each temperature is called the *cooling schedule*. Typically, at the beginning of the algorithm, T is large and an inferior solution has a high probability of being accepted. During this period, the algorithm acts as a random search to find a promising region in the solution space. As the optimization process progresses, the temperature decreases and there is a lower probability of accepting an inferior solution.

3 Problem Formulation

The proposed method starts with a graph of demands and generates, through a sequence of transformations, a set of routes. The key elements in implementing the annealing SDVRPTW algorithm are: 1) the definition of the initial configuration, 2) the definition of a neighborhood on the configuration space and perturbation operators exploring it; 3) the cost function; and 4) the cooling schedule. In what follows, we describe our annealing algorithm with reference to Figure 1.

```
GreedyCustomerSelector()
{
   i ← 1
   start ← i
   route.addElement(newInteger(0))
   fillcustsplit()
   while(not all customers picked)
   {
    Create a new route r_i
    i ← start
    while (i < customer_k && route capacity < capacity constraint)
    {
     if (demand_k is not satisfied && ( (route is empty) || (route not empty &&
     time_ready of customer_k >= time_due of last customer in route))) then
       {
       addcustomertoroute
       if ((routecapacity+customer's remaining demand) > capacity constraint)
         split this customer's demand
       else
           add the whole demand for this customer
       }
     i++
     }
    start++
    }
}
```

Figure 2: Initial Configuration

3.1 Configuration Representation

In order to solve the SDVRPTW problem, we propose the configuration shown in Figure 1(a). The representation is based on a vector where each cell $m \in v_1, v_2, \ldots, v_n$ corresponds to a customer and cell 0 denotes the depot v_0. In order to speed-up the operation, we maintain hash tables that represent the generated routes as well as the corresponding routes capacity and the splits. For example, given routes $(r_0, 15, 3, 16)$, a hash capacity entry of $(r_0, 20, 7, 3)$, a hash customer split of $(C_{15}, 0, 20)$, and a vehicle capacity of 30, one can deduce that C_{15} is the first visited customer serviced with a demand equal to 20. On the other hand, a route $(r_5, 12, 9, 4, 21)$ and a hash split entry of $(C_5, 8, 11, 9, 8)$ indicate that the demand for customer C_5 which is equal to 19 was split into two routes. Thus, C_5 is first visited in route r_8 and served the amount of 11 then visited in route r_9 and served its remaining demand which is equal to 8.

3.2 Initial Configuration

The initial configuration is generated using a local recursive greedy algorithm that finds the best local route (Figure 2). The algorithm schedules customers by maximizing the space that is available to other customers. Thus, customers are sorted based on their finish time, and customers who finish earlier are selected first and assigned to the route, until reaching the maximal route capacity constraint. Demands that violate the route capacity are split. Subsequent routes are built using the customer with the earliest finish time and not visited yet or not fully served. This

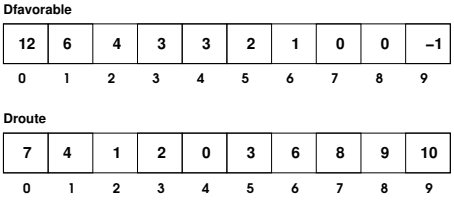

Figure 3: (a) Dfavorable, (b) droute

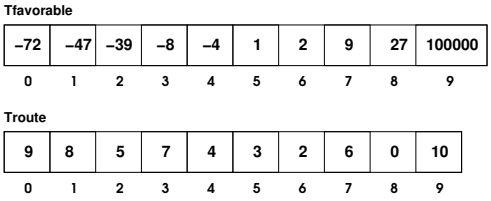

Figure 4: (a) tfavorable, (b) troute

procedure is repeated until all customers have been served. The overall number of splits is less than the number of routes and no two routes have more than one customer in common. During the implementation of the initial configuration, all hash tables are initialized. In specific, the *routes* hash table that represents, the *capacity* hash table that represents the customers' demand served, and the *split* hash table that maintains for each customer the route location and the demand served on that route.

3.3 Neighborhood Transformation

The annealing algorithm explores the design space using two transformations, *progress* and *replace highest first*. In what follows we present both transformations.

3.3.1 Progress

The *progress* transformation improves the results by perturbing the selected route through the removal of the *last* customers in every route and reallocating them. The transformation starts by sorting the removed customers in increasing order of their due time and reinserts them in three rounds. The first round minimizes the "demand splits" by inserting customers with the least due time and whose demands have been split into one of the routes they are already assigned to by adjusting the demand to the highest the route can hold. Thus reducing the number of splits. The second round iterates through the remaining customers and reinserts the customer with the lowest due time at the end of the most favorable route. We define a route to be favorable if the insertion of a customer into this route results with the smallest time gap along with the highest demand of that customer. The performance of the *progress* transformation is improved through the use of two hash tables. The first hash table, *tfavorable*, points to the gaps between the ready time of the customer under consideration and the due time of the last customer in each route. The second hash table, *dfavorable*, refers to the available demand that every route can provide the customer. The algorithm also maintains two additional lists. In specific, *troute* maps for each customer c_i the route number corresponding

```
RouteSearch()
{
   flag ← 0
   for(L<= favorable.size() && tfavorable.get(L) not related to an empty route)
   {
    if (L/2 >= avg && flag ==1)
       break
    if (tfavorable.get(L) >= 0.0)
    {
     for (M <== dfavorable.size() && dfavorable.get(M) > 0.0
     {
      if(route in common found)
      {
       oldavg ← avg
       avg ← (L + M)/2
       if(oldavg <= avg && flag==1)
          avg ← oldavg
       else
          routechosen ← routeincommon
       flag ← 1
       break
      }
     }
    }
   }
   if (flag = 1)
     insert customer at the end of route chosen
}
```

Figure 5: Efficient Route Search

to *tfavorable* while *droute* maps for each customer c_i the route number corresponding to *dfavorable*. Figure 3 and 4 show respectively *dfavorable* and *tfavorable* and the equivalent mapping for their *droute* and *troute*. The algorithm next chooses the first common route between the two *troute* and *droute*, and the most favorable route is picked in order to insert customer c_i. The first route in common is calculated throughout a *efficient route search technique* described in Figure 5 that goes through both vectors in *tfavorable* and *dfavorable*. The method ignores negative values in *tfavorable* and then picks the nearest route in common. Calculations for the nearest route is done by means of an average calculation for the positions in the vectors. Finally, the transformation allocates new routes for the remaining customers that could not be inserted in any route using the greedy technique that we described in section 3.2. The increase in routes is mainly due to conflicts with time windows or remaining capacity.

3.4 Replace Highest Average

The *replace highest average* procedure calculates the average distance between every two customers. So for customer c_i, the average distance d_i is $\frac{d_{i-1,i}+d_{i,i+1}}{2}$, then the five highest average distances are selected, and customers c_{i+1} equivalent to each of the five highest

```
Annealing_SDVRPTW()
{
   S0 = Initial solution
   α = 0.90            //Temperature reduction multiplier
   β = 1.05            // Iteration multiplier
   M0 = 5              //Time until next parameter update
   BestS = Best solution
   T = 6000
   CurrentS = S0
   CurrentCost = Cost(CurrentS)
   BestCost = Cost(BestS)
   Time = 0
   do {
      M = M0
      do {
         NewS = Neighbor(CurrentS);
         NewCost=Cost(NewS)
         Δ_Cost = NewCost − CurrentCost
         If (Δ_Cost < 0)
            CurrentS=NewS
            CurrentCost=Cost(CurrentS);
            If (NewCost < BestCost) then
               BestS=NewS
               BestCost = Cost(BestS)
         else if (Random < e^(−Δ_Cost/T) then
            CurrentS=NewS
            CurrentCost = Cost(CurrentS);
         M = M − 1
      } while (M ≥ 0)
      Time = Time + M0;
      T = α ∗T;
      M0 = β ∗ M0;
   } while (Time > MaxTime and T > 0.001);
   Return(BestS);
   checkIfValidRoute(BestS);
   checkIfValidDemand(BestS);
   checkIfValidDemandSplit(BestS);
   checkTimeConstraint(BestS);
}
```

Figure 6: Annealing SDVVRPTW Algorithm

average distances are removed from their route. After removing the 5 vertices they are inserted in a random route at the only position they could fit in, due to the time constraint. However if that customer already exists in that route then its demand served in that route is only increased.

3.4.1 Cost Function

Let R_i be a permutation of the customers in V_i specifying the order of visiting them, starting and finishing at the depot v_0. The cost of a given route $R_i = v_{i0}, v_{i1}, ..., v_{ik+1}$, where $v_{ij} \in V$ and $v_{i0} = v_{ik+1} = 0$ (0 denotes the depot), is given by:

$$Cost(R_i) = \sum_{j=0}^{k} c_{j,j+1} + \sum_{j=0}^{k} \delta_j \quad (1)$$

and the cost of the problem solution (S) is:

$$F(S)_{CVRP} = \sum_{i=1}^{m} Cost(R_i). \quad (2)$$

3.5 Cooling Schedule

The cooling schedule is the set of parameters controlling *the initial temperature*, *the stopping criterion*, *the temperature decrement* between successive stages, and *the number of iterations*. The cooling schedule was empirically determined as follows. The initial temperature, T_{init}, was set to 6000 while the temperature reduction multiplier, α, was set to 0.82. The number of iterations, M, was determined to be 5 while the iteration multiplier, β was set to 1.05. The algorithm stops when the temperature, T_f, is below 0.001.

4 SDVRPTW Annealing Algorithm

Each configuration represents an intermediate route that has a different cost. During every annealing iteration, the neighborhood of the configuration is explored. The algorithm must ensure that: (i) all routes start and end at the depot; (ii) a customer can be visited more than once; however its total demand must be fully served in the complete route; (iii) a route's capacity must not exceed the capacity constraint; (iv) customers in a given route must respect the time windows constraint. A vehicle v delivering goods to customer i can arrive before time a_i, but then it has to wait until this time to make the delivery; (v) all customers must be visited.

The algorithm starts with the initial greedy selective method, generates an initial solution then initiates the annealing process. The process generates neighborhood solutions through two steps the *progress* alteration then the *Replace Highest Average* modification. The *progress* transformation is applied for a probability of 82%, whereas the *Replace Highest Average* transformation is always applied. The solution is always feasible as the neighborhood transformations use a constructive approach that generates feasible routes. The variation in the cost functions, Δ_C, is computed and if negative then the transition from C_i to C_{i+1} is accepted. If the cost function increases, the transition is accepted with a probability based on the Boltzmann distribution. The temperature is gradually decreased from a high starting value, $T_{init} = 6000$, where almost every proposed transition, positive or negative, is accepted to a freezing temperature, $T_f = 0.001$, where no changes occur.

5 Experimental Results

The proposed algorithm was implemented using Java, on an Intel Core i3, 4GB RAM, CPU 2.53 GHZ. We verify the algorithm using the Solomon instances.[1]

5.1 R1 Benchmarks

The customers' geographical positions in sets R1 are randomly created. However customers have a

[1] http://w.cba.neu.edu/~msolomon/problems.htm

Benchmark	Best Known Optimal	Simulated Annealing			Std Deviation σ	Difference from Known Optimal
		Best	Worst	Average		
R101-25-30	772.5	833.53	864.02	846.91	11.71	7.90%
R101-25-50	631.5	635.37	658.27	645.39	7.66	0.61%
R101-25-100	617.1	560.15	574.33	568.92	4.80	-9.22%
R101-50-50	1191.1	1478.57	1567.48	1523.21	36.5	24.13%
R102-25-30	750.4	853.4	888.04	881.77	10.42	13.73%
R102-25-50	580.7	752.31	763.31	756.34	3.64	29.55%
R102-50-50	1114.2	1613.5	1642.72	1633.84	10.75	44.81%

Table 1: Results for the R1-type instances

Benchmark	Best Known Optimal	Simulated Annealing			Std Deviation σ	Difference from Known Optimal
		Best	Worst	Average		
R201-25-30	750.7	817.54	864.27	849.68	13.32	8.9%
R201-25-50	564.7	650.83	661.57	656.01	3.52	15.25%
R201-25-100	463.3	628.79	639.19	633.86	4.03	35.71%

Table 2: Results for the R2-type instances

short scheduling prospect therefore allocating only few customers per route (just about 5 to 10 customers per route). Each benchmark in set R1 was solved for 10 times and the best, worst and average results are reported in Table 1 in addition to the standard deviation and the difference from the optimal solutions reported in the literature. The average running time for the R1 set is 34 minutes for the problem with 25 customers, 43 minutes for 50 customers and 55 minutes for the 100 customers. However the running time decreases when the capacity is increased and turns into 18 minutes for the 25 customers with a capacity of 50, and 13 minutes for the 25 customers with a capacity equals to 100.

5.2 R2 Benchmarks

In problem sets R2, the geographical positions of the customers are also randomly created. Unlike to R1 they have a long scheduling perspective thus many customers are assigned to the same vehicle (more than 30). In the R2 set, each benchmark was resolved for 10 running times. Also the best, worst, average results and the standard deviation and the difference from the optimal solutions are reported in Table 2. The standard running time for the R2 set is 30 minutes for the problem with 25 customers. However for the 25 customers with capacity equals to 50, the running time is 19 minutes, and 17 minutes for a capacity equals to 100.

5.3 C1 Benchmarks

In problem sets C1, customers are clustered based on their geographical position. Nevertheless customers have a short scheduling prospect therefore allocating only few customers per route (just about 5 to 10 customers per route). Table 3 reports the best, worst, average results and the standard deviation and the difference from the optimal solutions. The average running time for the first instance of class C1 is 17 minutes for a capacity equals to 50. However for the second instance of class C1 the running time is 33 minutes for the capacity of 50 , and 25 minutes for the capacity of 100.

6 Conclusion

We have presented a meta-heuristic solution for the *the Split Delivery Vehicle Routing Problem with Time Windows*. The problem is solved based on simulated annealing using a combination of *random* and *deterministic operators* that are based on problem knowledge information. The algorithm was implemented and various benchmarks were attempted. The reported results are promising and future directions will focus on parallelization in order to further improve the results.

References

[1] M. Dror and P. Trudeau, "Savings by Split Delivery Routing," *Transportation Science*, Vol.

Benchmark	Best Known Optimal	Simulated Annealing			Std Deviation σ	Difference from Known Optimal
		Best	Worst	Average		
C101-25-50	516.8	462.66	472.03	466.45	2.75	-10.48%
C101-25-100	291.9	266.29	289.54	271.96	6.72	-8.77%
C102-25-50	516.5	648.06	663.23	655.38	4.64	25.47%
C102-25-100	291.9	484.52	492.15	486.2	2.12	65.98%

Table 3: Results for the C1-type instances

23, pp. 141-145, 1989.

[2] P. Toth and D. Vigo, *The Vehicle Routing Problem.* Bologna: SIAM, 2002.

[3] H. Harmanani, D. Azar, N. Helal, W. Keirouz, "A Simulated Annealing Algorithm for the Capacitated Vehicle Routing Problem," In *Proc. of the ISCA 26th CATA*, pp. 96-101, 2011

[4] J. Brandao and R.A. Eglese, "A Deterministic Tabu Search Algorithm for the Capacitated Arc Routing Problem," *Computers and Operations Research,* Vol. 35, pp. 1112–1126, 2008.

[5] E.D. Taillard, "Parallel Iterative Search Methods for Vehicle Routing Problems," *Networks,* Vol. 23, pp. 661–672, 1993.

[6] Y. Marinakis, M. Marinaki, and G. Dounias, "Honey Bees Mating Optimization Algorithm for Large Scale Vehicle Routing Problems," *Natural Computing*, Vol. 9, pp. 5-27, 2010.

[7] B.M. Baker and M.A. Ayechew, "A Genetic Algorithm for the Vehicle Routing Problem," *Computers and Operations Research,* vol. 30, pp. 787–800, 2003.

[8] C. Prins, "A Simple and Effective Evolutionary Algorithm for the Vehicle Routing Problem," *Computers and Operations Research,* Vol. 31, pp. 1985–2002, 2004.

[9] S. Mazzeo and I. Loiseau, "An Ant Colony Algorithm for the Capacitated Vehicle Routing," *Electron. Notes in Discrete Math.,* Vol. 18, pp. 181–186, 2004.

[10] C. Archetti, A. Hertz, and G. Speranza, "An Optimization-Based Heuristic for the Split Delivery Vehicle Routing Problem," *Transportation Science,* vol. 42, pp. 22–31, 2008.

[11] C. Archetti, M.G. Speranza, and A. Hertz, "A Tabu Search Algorithm for the Split Delivery Vehicle Routing Problem," *Transportation Science,* Vol. 40, pp. 64–73, 2006.

[12] G. Desaulniers, "Branch-And-Price-And-Cut For the Split Delivery Vehicle Routing Problem With Time Windows," *Operations Research,* vol. 58, no. 1, pp. 179–192, 2010.

[13] V. Campos, A. Corberan and E. Motta, "A Scatter Search Algorithm for The Split Delivery Vehicle Routing Problem," in *Studies in Computational Intelligence*, A. Fink and F. Rothaulf, Eds. Vienna, Austria: Springer, 2008, pp. 137–152.

[14] Y. Chang, and L. Chen, "Solve the Vehicle Routing Problem with Time Windows Via a Genetic Algorithm," *Discrete and Continuous Dynamical Syst.,* pp. 240–249, 2007.

[15] L.M. Gambardella, E. Taillard and G. Agazzi, "A Multiple Ant Colony System for Vehicle Routing Problems With Time Windows," in *New Ideas in Optimization,* D.W. Corne, M. Dorigo, and F. Glover, Eds. Peradenyia, Sri Lanka: McGraw Hill, 1999, pp. 63–76.

[16] Patrcia Belfiore, Hugo Tsugunobu and Yoshida Yoshizaki, "Scatter Search for Vehicle Routing Problem With Time Windows and Split Deliveries," *Vehicle Routing Problem*, Tonci Caric and Hrvoje Gold (Ed.), InTech, 2008.

[17] S.C. Ho and D. Haugland, "A Tabu Search Heuristic For The Vehicle Routing Problem With Time Windows and Split Deliveries," *Computers and Operations Research*, Vol. 31, pp. 1947–1964, 2004.

[18] P. W. Frizzell and J. W. Giffin, "The Split Delivery Vehicle Scheduling Problem With Time Windows and Grid Network Distances," *Computers and Operations Research*, Vol. 22, pp. 655-667, 1995.

[19] M. Dror, G. Laporte and P. Trudeau, "Vehicle Routing with Split Deliveries," *Discrete Applied Mathematics*, Vol. 50, pp. 239-254, 1994.

[20] J. M. Belenguer, M. C. Martinez, and E. Mota, "A Lower Bound for the Split Delivery Vehicle Routing Problem," *Operations Research*, Vol. 48, pp. 801-810, 2000.