EAI Endorsed Transactions
on Context-aware Systems and Applications
Research Article EAI.EU

Ant Colony-based Tabu List Optimization for Minimizing the Number of Vehicles in Vehicle Routing Problem with Time Window Constraints

J.L.E.K Fendji[1,*], M.V.K. Yakam[1], M.D. Fendji[2]

[1]Computer Engineering, University Institute of Technology, The University of Ngaoundere – Cameroon
[2]Faculty of Engineering and Technology, University of Buea, Cameroon

Abstract

The Vehicle Routing Problem consists in finding a routing plan for vehicles of identical capacity to satisfy the demands of a set of customers. Time window constraints mean that customers can only be served within a pre-defined time window. Researchers have intensively studied this problem because of its wide range of applications in logistics. In this paper, we tackle the problem on an economical point of view with a focus on capital expenditure (CAPEX), where the minimization of the number of vehicles is more important than the total traveling distance. This customization finds its applications in scenarios with limited CAPEX or seasonal/temporary operations. In these cases, the CAPEX should be minimized as much as possible to reduce the overall cost of the operation, while satisfying time window constraints. We provide an Ant Colony Optimization-based Tabu List (ACOTL). We test the proposed approach on the well-known Solomon's benchmarks. We compare experiments results to Dynamic Programming on small size instances and later to the best-known results in the literature on large size instances. ACOTL allows to reduce the number of vehicles used sometimes up to three units, compared to the best-known results, especially for instances where customers are geographically in clusters randomly distributed with vehicles of low or medium charges.

Keywords: Vehicle Routing Problem with Time Window, Ant Colony-based Tabu List, number of vehicles, CAPEX minimisation.

Received on 06 August 2020, accepted on 30 August 2020, published on 31 August 2020

doi: 10.4108/eai.12-5-2020.166041

1. Introduction

Every year, many logistic companies for delivery receive huge incomes from distribution processes. Distribution processes play an important role in supply chains, since almost half of the total supply chain cost comes from transportation processes [1]. For this reason, the management of distribution processes is critical in minimizing total supply chain cost. According to Toth and Vigo, transportation cost represents between 10 and 20 percent of goods' prices on the market, and computerized procedures based on optimization techniques permit make savings of around 5 to 20 percent on this transportation cost [2]. Transportation cost includes capital expenditures (CAPEX), that depends mainly on the cost and the number of vehicles, and operational expenditures (OPEX), that depends mainly on the total distance traveled by the set of vehicles. The determination of the number of vehicles and the route for each vehicle is known as the Vehicle Routing Problem.

Vehicle routing problem (VRP) is a common name associated to a class of combinatorial problems involving sets of customers that should be served by several vehicles [3]. The VRP can model various real-life problems, linked in supply chain management in the physical delivery of goods and services, such as postal deliveries, school bus routing, recycling routing and so on [4]. There are several variants of this problem. These are formulated based on the nature of the transported goods, the quality of service

*Corresponding author. Email:lfendji@gmail.com

required and the characteristics of the vehicles and the customers.

simulation results and the comparison with the best-known solutions in the literature, before ending with conclusions and future directions.

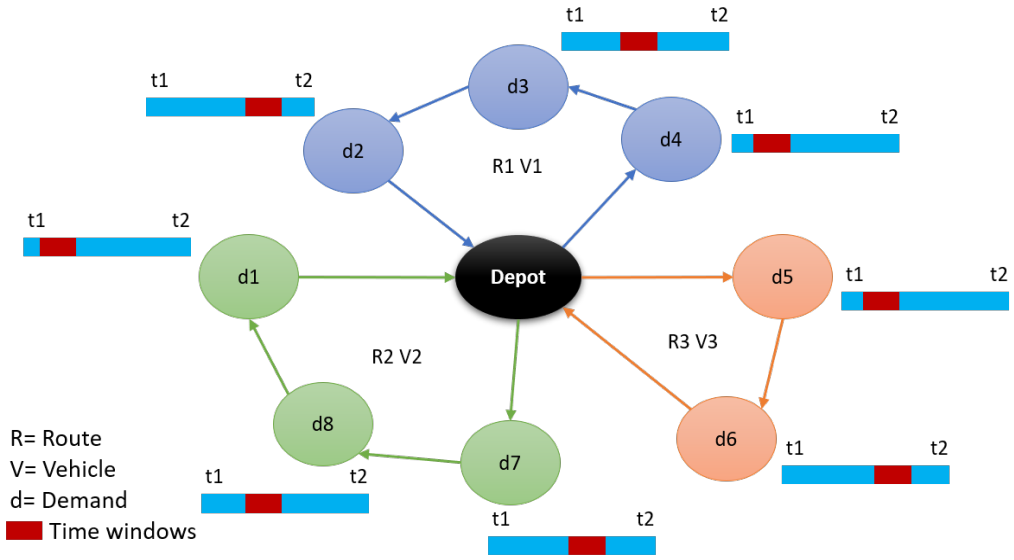

Figure 1. Illustration of a VRPTW solution.

In real-life scenarios, an important characteristic of customers is the time window during which a customer can be served. This characteristic extends the classic VRP problem to the well-known Vehicle Routing Problem with Time Window constraints (VRPTW). In VRPTW, routes must contain all the points (customer locations). Each point is visited within its time window by a single vehicle. Each route is associated to a vehicle and starts and ends at the depot. In addition, the total demands of all points on a route must be less than or equal to the capacity of the vehicle. Figure 1 presents an example of a VRPTW solution involving a depot and eight customers (nodes 1 to 8).

Approaches proposed to solve VRPTW usually try to optimize both CAPEX (number of vehicles) and OPEX (total traveled distance). But in some scenarios, the CAPEX is limited and the reduction of the number of vehicles of just one unit can make the set of vehicles affordable for the company. Moreover, some distribution processes can only be performed during a period or season because of the environmental conditions or the availability of products. This means reducing the CAPEX can drastically reduce the overall cost of the operation. This paper focuses on such scenarios, which are usually observed during agricultural campaigns, mainly in sub-Saharan Africa. To tackle this issue, an Ant Colony Optimization-based Tabu List approach is proposed, which is a combination of two well-known optimization approaches.

The rest of the paper is organized as follows. Section 2 briefly presents related works on VRPTW. The problem formulation is defined in Section 3; followed by the presentation of the proposed Ant Colony Optimization-based Tabu List approach in Section 4. Section 5 presents

2. Related Works

The Vehicle Routing Problem with Time Window constraints is classified as a NP-hard combinatorial optimization problem [5]. Consequently, approaches based on meta-heuristics are habitually used for larger instances of the VRPTW.

Most researchers model VRPTW as a multi-objective optimization problem with the aim of minimizing both the number of vehicles and the total travelled distance [6]; while others consider minimizing the number of vehicles as the primary objective like in [7]. In general, a two-phase approach is proposed starting by the minimization of the number of vehicles and ending by the minimization of the total traveled distance with a fixed number of routes.

Other works proposed new objective functions in VRPTW, including the minimization of the total waiting time [8].

Meta-heuristics are usually developed for solving the multi-objective VRPTW since the problem is NP hard. They work on a set of candidate solutions which require a high computation cost, depending on the size of inputs, to achieve high performance in VRPTW. More details are available in [9].

Several population-based approaches have been developed to VRPTW, such as Genetic Algorithms [2] and Artificial Bee Colony [10]. Ant Colony Optimization has been applied to Long-Distance VRP [11] and an Improved Ant Colony Optimization for Multi-Depot Vehicle Routing Problem is found in [12].

Authors in [13] proposed an incremental route building and an enhanced algorithm to tackle the VRP with soft time windows.

Some researchers tried to provide exact approaches such as restricted dynamic programming to solve VRPTW [14]. But this approach solves only small VRP instances because of the NP-hard property.

Many local search approaches have also been proposed to solve VRPTW, namely Tabu Search [15], Simulated Annealing [16], Variable Neighborhood Search (VNS) [17], Large Neighborhood Search [18], and Guided Local Search [19].

3. Problem Formulation

3.1. Notations

We will use the following list of notations to represent the problem formulation.

Parameters

N: number of customers {1,…,n}
K: Number of vehicles {1,…,k}
Q: capacity of vehicle K
q_i : customer i demand
d_{ij}: cost incurred on arc from node i to j
t_{ij}: travel time between node i and j
e_i: earliest arrival time at node i
e_0 : exit time from depot
f_i: latest arrival time at node i or j
f_0 : maximum route time allowed for vehicle k or return route time at depot
b_i : service time at node i
a_i : arrival time at node i
w_i: waiting time
r_k: total time of route allocated to each vehicle k

Decision Variables

x_{ij} = 1 if node i is visited immediately before node j. x_{ij} = 0 otherwise.

y_{ik}= 1 if the node request i is satisfied by the vehicle k. y_{ik} = 0 otherwise.

$b_{ik} = b_i$ if the vehicle k arrive at node i at time $ai \epsilon \max\{e_i; a_i\}$ for service $b_{ik} = 0$ if arrival time is $a_i > f_i$

We assume $a_0 = 0$ and $b_{0i} = 0$, for all k.

Let $G = (V; E)$ an undirected graph where $V = \{v_i;\ i = 0, \dots, n\}$ denoting a depot (v_0) and n customers $(v_i;\ i = 1, \dots, n)$. A non-negative demand q_i and service time s_i are associated with v_i, with $q_0 = 0$ and $s_0 = 0$. E is a set of arcs with non-negative weights d_{ij} (which often represents distance) between v_i and $v_j, (v_i, v_j) \epsilon V,\ i < j$. It is often assumed that it is symmetrical and satisfies the triangular inequality i.e., $d_{ij} = d_{ji}$.

All customer demands are served by a set of K vehicles. At each customer v_i, the starting of service time b_i must be in the time window $[e_i;\ f_i]$, where e_i and f_i are the earliest and latest time to serve v_i. If a vehicle arrives at v_i at time $a_i < e_i$, a waiting time $w_i = \max\{0;\ e_i - a_i\}$ is observed. Consequently, the starting of service time $b_i = \max\{e_i;\ a_i\}$. Each vehicle of a capacity Q travels on a route connecting a subset of customers starting from v_0 and ending within a schedule horizon $[e_0;\ f_0]$, corresponding to the earliest time of exit from the depot and the latest time of return to the depot.

3.2. Model

The objective function of the model is:

$minimize$ K

Subject to the following constraints:

$$\sum_{j=1}^{n} x_{ijk} = 1 \quad for\ i = 0\ and\ k \epsilon \{1, \dots, K\} \tag{1}$$

$$\sum_{j=1}^{n} x_{jik} = 1 \quad for\ i = 0\ and\ k \epsilon \{1, \dots, K\} \tag{2}$$

$$\sum_{k=1}^{K} \sum_{\substack{j=1 \\ j \neq i}}^{n} x_{ijk} = 1, \quad for\ i \epsilon \{0,1, \dots, n\} \tag{3}$$

$$\sum_{k=1}^{K} \sum_{\substack{i=1, \\ i \neq j}}^{n} x_{ijk} = 1, \quad for\ j \epsilon \{0, \dots, n\} \tag{4}$$

$$\sum_{i=0}^{n} \sum_{\substack{j=1 \\ i \neq j}}^{n} x_{ijk} y_{ik} = n \quad for\ k \epsilon \{1, \dots, k\} \tag{5}$$

$$\sum_{i=0}^{n} \sum_{\substack{j=1 \\ i \neq j}}^{n} q_i x_{ijk} y_{ik} \leq Q \quad for\ k \epsilon \{1, \dots, k\} \tag{6}$$

$$a_0 = b_0 = w_0 = 0 \tag{7}$$

$$e_i \leq b_i \leq f_i \qquad e_i = a_i + w_i \tag{8}$$

$$\sum_{k=1}^{K} \sum_{\substack{i=0 \\ i \neq j}}^{n} x_{ij} \left(a_i + b_i + w_i + t_{ij}\right) \leq a_j\ for\ j \epsilon \{1, \dots, K\}$$

(9)

$$\sum_{i=1}^{n} \left(t_{ij} + w_i + b_i\right) \leq r_k,\ for\ j = 0,\ k \epsilon \{1, \dots, k\} \tag{10}$$

$$x_{ijk} \epsilon \{0; 1\} \quad \forall 0 \leq i;\ j \leq n;\ 1 \leq K \leq k \tag{11}$$

As defined in [20], constraint (1) ensures that for each vehicle starting its tour from the depot. There is exactly one outgoing arc from this node. Similarly, the constraint set (2) guarantees that for each vehicle k, ending its tour to the depot (I=0), there is exactly one entering arc into the node. Both constraints (1) and (2) together guarantee a complete tour for each vehicle. Constraint (3) ensures that from each node i only one arc is outgoing for each vehicle. Constraint (4) makes sure that for each node j, only one arc is incoming for each vehicle. Constraints (3) and (4) ensure that each vehicle visits each node only once. Constraint (5) makes sure that for each vehicle starting its trip from depot, n nodes are visited. Constraint (6) guarantees that for each

vehicle, the total demand of customers assigned to it does not exceed its capacity. The constraint (7) sets the arrival, waiting and service times at the depot to zero for each vehicle. The constraint (8) ensures that the sum of the arrival and waiting times at each node i and for each vehicle is within the time window (between the earliest arrival time at that node and latest arrival time), $i = 1,2,3,\cdots,n$. The constraint (9) ensures that the arrival time of each vehicle to each node j is not greater than the specified arrival time at that node. Constraint (10) ensures that the total traveling time of each vehicle is not greater than the maximum route time allocated to that vehicle. This is done to avoid any uncompleted tour.

4. Ant Colony Optimization-based Tabu List (ACOTL)

4.1. Basic Ant Colony Optimization algorithm

Ants can solve complex problems collectively, such as finding the shortest path between two points in a rugged environment. For this, they communicate with each other locally and indirectly, thanks to a volatile hormone called pheromone. In fact, during its progression, an ant leaves behind a trace of pheromone which increases the probability that other ants passing nearby choose the same path using the receivers in their antennas [19, 21]. This collective problem-solving mechanism is at the origin of algorithms based on artificial ants.

The first ant-based algorithm, called Ant System, was proposed by Marco Dorigo in 1992 [22], and its performances were initially illustrated on Traveling Salesman Problem. Thus, various improvements have been made to the initial algorithm, giving rise to different variants of Ant System, such as ACS (Ant Colony System) and MMAS (MAX - MIN Ant System) [23, 24] which get in practice competitive results.

Many works on ant colony optimization have been inspired by MMAS algorithmic scheme. According to ACO meta-heuristic, at each cycle of the algorithm, each ant builds a solution. These solutions can be improved by applying a local search procedure. The pheromone traces are then updated. Each trace is "evaporated" by multiplying it by a persistence factor ρ between 0 and 1. A certain quantity of pheromone proportional to the quality of the solution, is then added to the components of the best solutions (the best solutions built during the last cycle or best solutions built since the start of the execution).

Among the problems strictly related to the one considered in this paper, the first one to which this method has been applied is the Traveling Salesman Problem (TSP) [25]. Then several other algorithms have been proposed for VRP [26] and VRPTW [27].

4.2. Ant Colony Optimization-based Tabu List

The proposed approach enhances the basic Ant Colony Optimization with an additional feature: the Tabu List. The main idea behind the approach is the following: Each time an ant m needs to move to the next city, a random search function is called to select a new city. Then the total traveling time is computed to check whether it is possible to move to that city and come back to the depot.

- If the move is possible then the city is visited, and the Tabu List is reset.
- Otherwise, it is considered as a prohibited city and stored in the Tabu List to avoid being selected once again at the next call by the search function during the same iteration. After multiple unsuccessful tries (the time window constraint is not satisfied), ant m returns to depot (node 0).

The number of tries is defined by the parameter try which is reset at the beginning of an iteration and each time a selected node can be visited.

Algorithms 1 to 5 detail the proposed approach. The following parameters are used by the different algorithms.

M: the number of ants;

α: Pheromone parameter;

β: Visibility parameter;

vol: Evaporation Rate of pheromone ϵ] 0,1[;

Q_{pher}: Pheromone quantity deposed by each ant on track after constructing of a solution;

τau: remaining quantity of pheromone after evaporation at the end of each iteration, added to total sum of pheromone deposed on track by ants at each iteration;

deltaτau: total sum of pheromone deposed on track by ants at each iteration;

Heu_F : Heuristic function (1/D);

velocity: Vehicle velocity;

$TabuList$: list of current nodes that do not satisfy time window constraints;

try: number of time that function of search is called to choose the next node to visit.

NV_{set}: Matrix containing number of vehicles of a set of solution built by m ants at each iteration.

$AllowIndex$: logical row matrix containing 0 for visited nodes and 1 for not visited.

$TargetIndexes$: indexes of target nodes randomly selected.

4.3. Algorithm Explanation

The flowchart of the proposed approach is provided in Figure 2. It can be decomposed into five main steps. The complete algorithm is provided in Algorithm1.

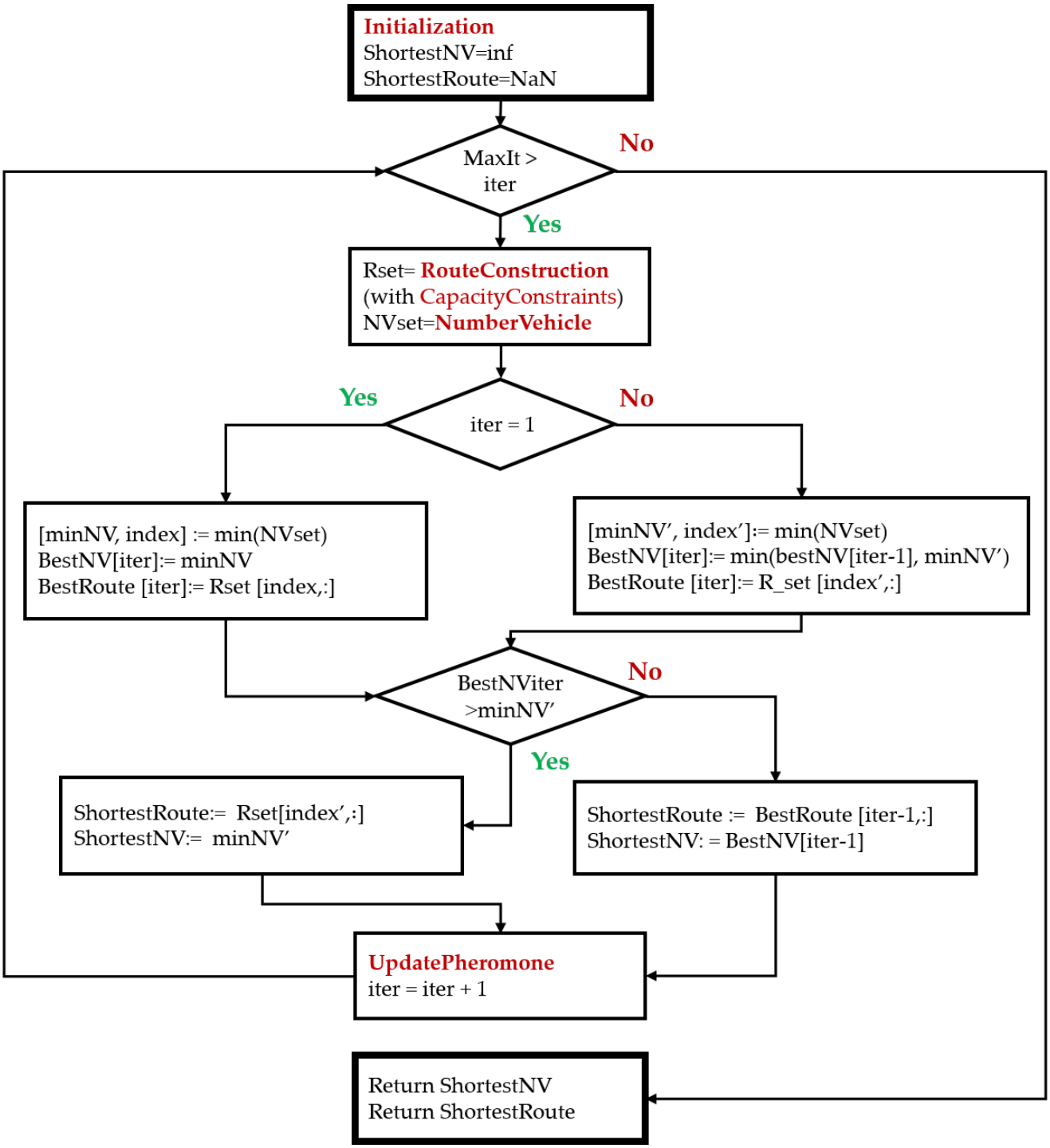

Figure 2. Flowchart of the proposed approach

Algorithm1: ACOTL for VRPTW

Input: VRPTW instance
Output: ShortestNV, ShortestRoute

```
1  Begin
2    Parameters initialization (α, β, vol, Q_pher, MaxIt, m, n,
3                               τau, Heu_F, velocity, iter)
4    bestNV MaxIt, 1 :=0, addition:= n-1,
5    R_set[1, m][1, n+addition]:=0, BestRoute[MaxIt,1]:=0,
6    CityIndex :=[1, n], a[1, n] :=0
7    while MaxIt > iter do
8      R_set =RouteConstruction(VRPTW instance, m, velocity, CityIndex, a)
9      NV_set= NumberVehicle(R_set, m)
10     if iter == 1 then
11       [minNV, index] : = min ( NV_set )
12       BestNV[iter]:= minNV
13       BestRoute [iter]:= R_set [index, : ]
14     else
15       [minNV', index'] := min ( NV_set )
16       BestNV[iter]:= min(bestNV[iter-1], minNV')
17       BestRoute [iter]:= R_set [index', : ]
18     if BestNV iter >= minNV' then
19       ShortestRoute:= R_set [index', :]
20       ShortestNV: = minNV'
21     else
22       ShortestRoute := BestRoute [iter - 1, : ]
23       ShortestNV: = BestNV[iter-1]
24     updatePheromone ( R_set , NV_set , Q_pher , vol, τau)
25     iter := iter + 1
26   return ShortestNV, ShortestRoute
27 End
```

Step 1: Initialization (Algorithm 1)

1. Lines 2 to 6: Parameters values are defined in section 4.2 with D the matrix distance between nodes and N the number of customers as presented in section 3.

- ACO is divided into two main phases, which are ant's route construction and the pheromone update [28].

2. Before route construction (Algorithm 2), at each iteration, all ants are located at the depot. The set of demands q_i of cities is known beforehand. All cities are set as unvisited.

Step 2: route construction (Algorithm 2)

3. Lines 6 to 15: At each construction step of the 'route', each ant m at node $j-1$, applies a probabilistic rule to the next node to visit. The choice of moving to a node j depends on two values: heuristic function Heu_F_{ij} and the level or the rate of pheromone τau_{ij} on the arc (i, j) according to (11).

$$P_{ij}^{m} = \frac{(\tau au_{ij})^{\alpha}(Heu\_F_{ij})^{\beta}}{\sum_{j\epsilon J_i^m}(\tau au_{ij})^{\alpha}(Heu\_F_{ij})^{\beta}} \qquad (11)$$

P_{ij}^{m} is a moving rule called "probabilistic random proportional rule". It is the probability that an ant m moves from node i to node j, which belongs to a set of nodes J_i^m that are not yet visited by the ant m.

4. Lines 16 to 19: when a city j is chosen according to the moving rule, some computations are performed: the travel time of the ant m from node i to a randomly chosen node j (t_{ij}), the waiting and services times, and the travel time of the ant m from node j to the depot 1 (t_{j1}).

5. Lines 20 to 28: If the values t_{ij} and t_{j1} do not satisfy time window constraints $[e_j; f_j]$ and $[e_0; f_0]$, then the movement to j is considered as unfeasible. The node is therefore inserted in the Tabu List (short-term memory). The choice to move to the next node by ant m is repeated up to try times until time window constraints are satisfied. In unsuccessful cases, ant m returns to the depot. Otherwise, the node is appended to the route

and the arrival and waiting time (a_j and w_i) at node j are computed for the next iteration (Lines 29 to 32).

Step 3: vehicle capacity constraints (Algorithm 3)

6. Algorithm 3 takes as input the set of demands q_i of cities; a route constructed by each ant at each step of the construction, containing visited nodes; indexes of nodes CityIndex; binary matrix AllowIndex containing 1 for unvisited nodes.
7. Lines 3 to 5: If ant *m* is at the depot, vehicle load is set to the maximum value, all cities receive logical number 1, excepted starting node.
8. Lines 6 to 12: Else, if the need of a node is less than or equal to current vehicle load then this node is served, and vehicle load is decreased consequently.

Algorithm 2: RouteConstruction

Input: VRPTW instance, m, velocity, CityIndex, a
Output: R_{set}: set of m route constructed by each ant

```
1  Begin
2    R_set [1:m,1]:=1
3    for i = 1: m do
4      q_set [1,n] := [ q_1, q_n ]
5      AllowIndex[1:n]=1,  AllowIndex[1]:=0
6      for j: =  2: n +  addition do
7        route :=  R_set [i][1,j-1]
8        AllowIndex:=CapacityConstraint(q_set , route, CityIndex, AllowIndex)
9        Try:=5, TabuList :=[], stop:=0
10       while Stop==0 && Try>0 do
11         AllowIndex [TabuList] :=0
12         Allow:=CityIndex[AllowIndex]
13         for k := 1:length(Allow) do
14           p[k]:=  (τau[route[end], Allow[k]])^α ( Heu_F[route[end], Allow[k]])^β
15         P_sum := sum (p)
16         TargetIndexes :=find (cumsum (P_sum )>=rand)
17         Target := Allow[TargetIndexes[1]]
18         t[route[end],Target]:=a[route[end]]+(d[route[end],target])/velocity
19         t[Target,1]=t[route[end],Target]+b[Target]+w[Target] +(d[target,1])/velocity
20         if t route end , Target  <=  f[Target] &&  t Target, 0  <= f[0] then
21           Stop:=1
22         else
23           TabuList:=[index, Target]
24         Try:= Try-  1
25       if Stop==0 then
26         R_set  i [j] :=1
27       else
28         R_set  i [j] := Target
29     a[Target]:= a[route[end]]+b[Target]+w[Target]+(d(route[end],target))/velocity
30      w[Target]  :=  w[Target] - t[route[end], Target]
31     if  w[Target] < 0 then
32       w[Target]:=0
33   return  R_set
34 End
```

Algorithm 3 : CapacityConstraints

Input: q_{set}, route, CityIndex, AllowIndex
Output: AllowIndex: city indexes to explore

```
1  Begin
2    load:=Q
3    if route[end] ==1
4      load:=Q
5      AllowIndex := ismember (CityIndex, find(q_set ! = 0))
6    else
7      if load >= q[route[end]] then
8        load := load-  q[route[end]]
9        q_set [route[end]] := 0
10       AllowIndex:= ismember (CityIndex, find(q_set ! = 0))
11       if load == q[route[end]] then
12         AllowIndex [1: n] :=0, AllowIndex[1]:=1
13     else
14       AllowIndex [1: n] :=0, AllowIndex [1]:=1
15       q[route[end]] := q[route[end]] - load
16   return AllowIndex
17 End
```

Algorithm 4: NumberVehicle

Input: R_{set} : set of m route, m: number of ants
Output: NV_{set} : set of number vehicle of m route

```
1  Begin
2    NV_set  1, m  : = 0, count: =1
3    for i = 1: length ( R_set ) do
4      for j  =  1: length ( R_set [i])
5        if  R_set  i  [j] :=1 then
6          count := count +1
7        else
8          count := count -  1
9      NV_r  := count
10     NV_set [i]:=  NV_r
11   End
```

Algorithm 5: updatePheromone

Input: R_{set} , NV_{set} , Q_{pher} , vol, τau
Output: τau

```
1  Begin
2    deltaτau  1, n   1, n  = 0
3    for i  =  1: m then
4      for j  =  1: n -  1 then
5        deltaτau[ R_set  i  [j], R_set  i  [j + 1]]:=  deltaτau[ R_set  i  [j],  R_set  i  [j + 1]]+ Q_pher / NV_set [i]
6        deltaτau[ R_set  i  [n], R_set  i  [1]]:= deltaτau[ R_set  i  [n], R_set  i  [1]] + Q_pher / NV_set [i]
7    τau = (1 -  vol) τau + deltaτau
8    return τau
9  End
```

In case the new vehicle load equals zero, the vehicle returns to the depot.

9. Lines 13 to 15: if the need of a node is higher than current vehicle load, this node is partially served, and the vehicle returns to the depot.

This phase is repeated $M \times n$ times with the condition that each ant m is a solution, each solution encompasses k tours, each tour starts and ends at the depot, and each node must be visited only once with respect to time window constraints.

Step 4: minimal vehicle number (Algorithm 1 and 4)

10. Lines 7 to 9 (Algorithm 1): for each iteration, when all the ants have built their solutions, for each solution, the number of vehicle is determined using Algorithm 4, and the result is saved in $\mathrm{NV_{set}}$.
11. Lines 10 to 23 (Algorithm 1): At the first iteration, minimal value NV is determined and saved in BestNV. At the following iterations, minimal NV value of current solutions is compared to BestNV. In case NV is smaller than BestNV, the latter is updated consequently.

Step 5: pheromone update (Algorithm 1 and 5)

12. Line 24 (Algorithm 1): at each iteration, after each solution has been constructed and minimal NV value has been found, each ant m deposes a quantity pheromone on its path depending on deltaτau_{ij}^{m} that is computed using Algorithm 5.

If the edge (i,j) is included in the route of the ant m, the quantity of pheromone deposited on this path is $\frac{\mathrm{Q}_{pher}}{NV_{set}(m)}$. Else it is equal to zero as presented in (12)

$$\text{delta}\tau au_{ij}^{m}(t) = \begin{cases} \frac{\mathrm{Q}_{pher}}{NV_{set}(m)} & if\ (i,j) \epsilon\ \ {R_{set}}^{m}(t) \\ 0 & if\ (i,j) \notin\ {R_{set}}^{m}(t) \end{cases} \quad (12)$$

with ${R_{set}}^{m}(t)$ the solution built by the ant m at the iteration t, $NV_{set}(m)$ the number of vehicles of the solution built by the ant m.

Line 7 (Algorithm 5): At the end of each iteration of the algorithm, quantity pheromones deposited at the previous iteration by the ants evaporate depending on $vol * \tau\text{au}\ (t)$.

13. At the next iteration t+1, the quantity of pheromones on the route of each ant after evaporation is given by (13).

$$\tau au_{ij}(t+1) = (1 - vol)\ \tau\text{au} + \text{delta}\tau\text{au}_{ij} \quad (13)$$

$$\text{With} \quad deltat\tau au_{ij} = \sum_{m=1}^{M} deltat\tau au_{ij}^{m}(t) \quad (14)$$

To neglect all the bad solutions obtained, and thus avoid convergence towards local optimum, the concept of evaporation of the pheromone tracks is simulated through a parameter vol called the evaporation rate ($0 < vol < 1$).

5. Simulation results

5.1. Parameters and Datasets

The algorithm has been implemented in MATLAB 2018 and tested on the well-known Solomon benchmark composed of six datasets (C1, C2, R1, R2, RC1, RC2) [29]. Each dataset contains 08 to 12 instances of 100 customers with their respective locations and demands. Customers are grouped into clusters in C1 and C2, while they are randomly distributed geographically in R1 and R2. RC1 and RC2 are combination of both previous distributions. The results of ACOTP algorithm are compared with that of DP and that of best-known results from the literature.

Table 1 presents the important information of the datasets used in this study, and Table 2 shows the parameter settings.

5.2. Results and discussions

Comparison of results are performed in two phases. We first compare the performance of ACOTL with DP on small datasets. Secondly, we compare the results of ACOTL to best-known results.

Comparison of performance ACOTL vs DP

In this phase the performance of $ACOTL$ on instances of size 15 is compared with the one of DP on the same reduced instances. DP is run once while $ACOTL$ is run twenty times per instance.

Table 3 shows results of DP and $ACOTL$ on datasets C_1, C_2, R_2, RC_1, and RC_2. Both algorithms provide the same number of vehicles except on R_{201} instance where $ACOTL$ provide the smallest number of vehicles. This result shows the efficiency of $ACOTL$ and proves that DP cannot optimally solve some problems. In fact, it has been proven that DP cannot optimally solve the longest path problem. In fact, in this paper, we assume that determining the longest path of each vehicle can reduce the number of vehicles. This justifies the fact that DP cannot always solve the problem to the optimality.

Comparison of performance ACOTL vs Best-known results

In this phase the performance of $ACOTL$ on instances of size 100 is compared with of best-known results [30]. $ACOTL$ is run independently 30 times on each instance of datasets.

Table 4 presents performances of $ACOTL$ compared to best-known results on the dataset C_1. The results show that $ACOTL$ provides the same number of vehicles compared to the best-known results on some instances (C_{102}, C_{107}, C_{109}).

For the rest of instances, there is a difference of one vehicle.

On dataset C_2, ACOTL provide a number of vehicles equals to the best-know result just for the instance C_{208}. number of vehicles than the best-known results on C_2, as presented in Table 5.

Table 6 compares the results of $ACOTL$ on the dataset R_2 to best-known results. $ACOTL$ presents the same number of vehicles to the best-known results on most the

Table 1. Datasets

Variable	C_1	C_2	R_1	R_2	RC_1	RC_2
Number of customers (n)	100	100	100	100	100	100
Vehicle capacity (Load_max)	200	700	200	1000	200	1000

Table 2. Parameter settings

Parameters	ACO_{15}	$ACOTL$	DP
Number of execution (R)	10	10	1
Number of customers (N)	15	101	15
Vehicle capacity (Load_max)	50	[200 700 1000]	50
Iteration maximal (MaxIt)	500	250	-
Number of ants (M)	100	100	-
Visibility parameter (β)	1	1	-
Pheromone parameter (α)	1	1	-
Evaporation Rate of pheromone (vol)	0.5	0.5	-
velocity	45	45	45

Table 3. Comparison between ACOTL and DP

Data sets	C_1	C_2	R_{201}	$R_{202\ to\ 211}$	RC_1	RC_2
$ACOTL$	5	5	4	4	6	6
DP	5	5	5	4	6	6

Table 4. Performance of ACOTL and Best-known results on dataset C_1

Instances	C_{101}	C_{102}	C_{103}	C_{104}	C_{105}	C_{106}	C_{107}	C_{108}	C_{109}
$ACOTL$	11	10	11	11	11	11	10	11	10
Best-known results	10	10	10	10	10	10	10	10	10

Apart from that instance ACOTL provides a greater

instances. However, on other instances, $ACOTL$ provides a

Table 5. Performance of ACOTL and Best-known results on dataset C_2

Instances	C_{201}	C_{202}	C_{203}	C_{204}	C_{205}	C_{206}	C_{207}	C_{208}
$ACOTL$	4	4	5	4	4	4	4	3
Best-known results	3	3	3	3	3	3	3	3

Table 6. Performance of ACOTL and Best-known results on dataset R_2

Instances	R_{201}	R_{202}	R_{203}	R_{204}	R_{205}	R_{206}	R_{207}	R_{208}	R_{209}	R_{210}	R_{211}
$ACOTL$	5	5	4	3	3	3	3	2	3	3	2
Best-known results	4	3	3	2	3	3	2	2	3	3	2

Table 7. Performance of ACOTL and Best-known results on dataset RC_1

Instances	RC_{101}	RC_{102}	RC_{103}	RC_{104}	RC_{105}	RC_{106}	RC_{107}	RC_{108}
$ACOTL$	11	10	10	9	11	9	9	9
Best-known results	14	12	11	10	13	11	11	10

Table 8. Performance of $ACOTL$ and best-known results on dataset RC_2

Instances	RC_{201}	RC_{202}	RC_{203}	RC_{204}	RC_{205}	RC_{206}	RC_{207}	RC_{208}
ACO_{101}	5	4	4	3	5	3	3	2
Best-known results	4	3	3	3	4	3	3	3

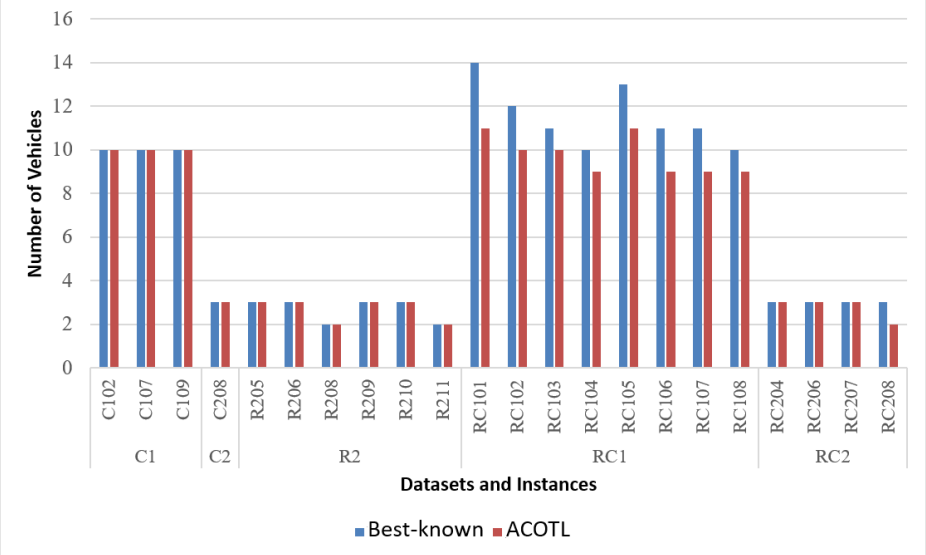

Figure 3. Performance of ACOTL over Best-known results in terms of number of vehicles.

greater number of vehicles.

From Table 7 $ACOTL$ outperforms previous best-know results on all the instances from datasets RC_1. In fact, ACOTL can reduce the number of vehicles by decreasing by up to three unit, the best-known result.

Finally, Table 8 compares the results from $ACOTL$ to best-known results on instance RC_2. ACOTL provides the same number of vehicles as the best-known on three instances and improve the known result of the instance RC_{208}. However, its number of vehicles is greater on half the sample. Figure 3 recapitulates the best results of ACOTL. From this figure, the proposed approach is suitable for scenarios with customers geographically located in clusters randomly distributed and vehicle with low or medium charges.

6. Conclusion

This paper has tackled the Vehicle Routing Problem with Time Window constraints from an economical point of view in which the CAPEX (in terms of number of vehicles) should be minimized. We proposed a new meta-heuristic called Ant Colony Optimization-based Tabu List to minimize this number of vehicles. Experimental results showed that the proposed approach can reduce by up to three the number of vehicles in some instances of the Solomon Benchmark, especially in RC1 dataset. In general, ACOTL performs well in scenarios with customers geographically located in clusters randomly distributed, especially with vehicles bearing low or medium charges.

However, the approach still needs an improvement for some instances in C and R datasets. A possible enhancement can be the introduction of some operators in the combination of ACO and the Tabu List, to extend the number of cities an ant can visit. For instance, a swap operator can be used to replace a city in the current solution by one or several cities in the Tabu List. But this approach will require a customization of the inner features of the ACO approach.

References

[1] Rodrigue J-P (2016) The geography of transport systems. Taylor & Francis

[2] Toth P, Vigo D (2002) The vehicle routing problem. SIAM

[3] Ghezavati VR, Hooshyar S, Tavakkoli-Moghaddam R (2017) A Benders' decomposition algorithm for optimizing distribution of perishable products considering postharvest biological behavior in agri-food supply chain: a case study of tomato. Cent Eur J Oper Res 25:29–54

[4] Laporte G (1992) The vehicle routing problem: An overview of exact and approximate algorithms. Eur J Oper Res 59:345–358

[5] Ho SC, Haugland D (2004) A tabu search heuristic for the vehicle routing problem with time windows and split deliveries. Comput Oper Res 31:1947–1964

[6] Bräysy O (2003) A reactive variable neighborhood search for the vehicle-routing problem with time windows. Inf J Comput 15:347–368

[7] Lenstra JK, Kan AR (1981) Complexity of vehicle routing and scheduling problems. Networks 11:221–227

[8] Ghoseiri K, Ghannadpour SF (2010) Multi-objective vehicle routing problem with time windows using goal programming and genetic algorithm. Appl Soft Comput 10:1096–1107

[9] Jozefowiez N, Semet F, Talbi E-G (2008) Multi-objective vehicle routing problems. Eur J Oper Res 189:293–309

[10] Xu S-H, Liu J-P, Zhang F-H, Wang L, Sun L-J (2015) A combination of genetic algorithm and particle swarm optimization for vehicle routing problem with time windows. Sensors 15:21033–21053

[11] Alzaqebah M, Abdullah S, Jawarneh S (2016) Modified artificial bee colony for the vehicle routing problems with time windows. SpringerPlus 5:1298

[12] Sicilia JA, Oliveros M-J, Larrode E, Royo B (2015) Solving a long-distance routing problem using ant colony optimization.

[13] Yalian T (2016) An improved ant colony optimization for multi-depot vehicle routing problem. Int J Eng Technol 8:385–388

[14] Kok AL, Meyer CM, Kopfer H, Schutten JMJ (2009) Dynamic programming algorithm for the vehicle routing problem with time windows and EC social legislation. Transp. Sci.

[15] Potvin J-Y, Kervahut T, Garcia B-L, Rousseau J-M (1996) The vehicle routing problem with time windows part I: tabu search. Inf J Comput 8:158–164

[16] Van Breedam A (1995) Improvement heuristics for the vehicle routing problem based on simulated annealing. Eur J Oper Res 86:480–490

[17] Hansen P, Mladenović N, Pérez JAM (2010) Variable neighbourhood search: methods and applications. Ann Oper Res 175:367–407

[18] Shaw P (1998) Using constraint programming and local search methods to solve vehicle routing problems. In: Int. Conf. Princ. Pract. Constraint Program. Springer, pp 417–431

[19] Kilby P, Prosser P, Shaw P (1999) Guided local search for the vehicle routing problem with time windows. In: Meta-Heuristics. Springer, pp 473–486

[20] Kumar SN, Panneerselvam R (2012) A Survey on the Vehicle Routing Problem and Its Variants. https://doi.org/10.4236/iim.2012.43010

[21] Deneubourg J-L, Aron S, Goss S, Pasteels JM (1990) The self-organizing exploratory pattern of the argentine ant. J Insect Behav 3:159–168

[22] Dorigo M (1992) Optimization, learning and natural algorithms. PhD Thesis Politec. Milano

[23] Dorigo M, Maniezzo V, Colorni A (1996) Ant system: optimization by a colony of cooperating agents. IEEE Trans Syst Man Cybern Part B Cybern 26:29–41

[24] Stützle T, Hoos HH (2000) MAX–MIN ant system. Future Gener Comput Syst 16:889–914

[25] Dorigo M, Gambardella LM (1997) A cooperative learning approach to the traveling salesman problem. IEEE Trans Evol Comput 1:53–66

[26] Bullnheimer B, Hartl RF, Strauss C (1999) An improved ant System algorithm for thevehicle Routing Problem. Ann Oper Res 89:319–328

[27] Toklu NE, Gambardella LM, Montemanni R (2014) A multiple ant colony system for a vehicle routing problem with time windows and uncertain travel times. J. Traffic Logist. Eng. 2:

[28] Reed M, Yiannakou A, Evering R (2014) An ant colony algorithm for the multi-compartment vehicle routing problem. Appl Soft Comput 15:169–176

[29] Solomon MM (1987) Algorithms for the vehicle routing and scheduling problems with time window constraints. Oper Res 35:254–265

[30] 100 customers. /projectweb/top/vrptw/solomon-benchmark/100-customers/. Accessed 24 Jul 2020