RAIRO-Oper. Res. 55 (2021) 639–652
https://doi.org/10.1051/ro/2021034

RAIRO Operations Research
www.rairo-ro.org

OPEN CAPACITATED ARC ROUTING PROBLEM BY HYBRIDIZED ANT COLONY ALGORITHM

Bilal Kanso[1,*], Ali Kansou[1] and Adnan Yassine[2]

Abstract. The Open Capacitated Arc Routing Problem OCARP is a well-known NP-hard real-world combinatorial optimization problem. It consists of determining optimal routes for vehicles in a given service area at a minimal cost distance. The main real application for OCARP is the Meter Reader Routing Problem (MRRP). In MRRP problem, each worker in the electric (or gas) company must visit and read the electric (or gas) meters to a set of customers by starting his route from the first customer on his visit list and finishing with the last one. The worker leaves where he wants once all the associated customers have been visited. In this paper, a metaheuristic called an Hybridized Ant Colony Algorithm (HACA) is developed and hybridized with a local search algorithm that involves the 2-opt, Swap, Relocate and Cross-exchange moves to solve OCARP problem. Computational results conducted on five different sets of OCARP-instances showed that our proposed algorithm HACA has reached good and competitive results on benchmark instances for the problem.

Mathematics Subject Classification. 90C32, 90C26, 90C59.

Received June 16, 2019. Accepted March 2, 2021.

1. Introduction

The Open Capacitated Arc Routing Problem (OCARP) is a special variant of the Capacitated Arc Routing Problem (CARP) where a vehicle starts at a given edge and does not return to the depot after servicing the last edge on a route. In the last ten years, OCARP received attention in the operation research literature. Indeed, it turns out that it is suitable to model for a wide range of real-life applications such as package delivery problem, newspaper home delivery problem, school bus problem or meter reader routing problem [14, 21]. In such applications, the employees use their own vehicles to serve edges and are not required to return to depot after completing service of edges. The objective is to design a set of routes of total minimum distance travelled by all vehicles. The authors in [17] showed that OCARP is NP-hard and proposed a formal definition of this variant of CARP which can be described as follows: (1) the graph $G(N, E, E_r)$ is directed where N is the set of nodes, E is the set of all edges and E_r is the set of required edges (called *tasks*) with demands strictly positive; (2) a fleet of M identical vehicles of capacity Q where M is predefined; (3) each task must be served by one direction and by one and only one vehicle; (4) every route (made by a vehicle) starts from the first task, serves

Keywords. Open Capacitated Arc Routing Problem, metaheuristic, Ant Colony Algorithm, local search, Simulated Annealing.

[1] Department of Computer Science, Lebanese University, Beirut, Lebanon.
[2] Normandie University, 25 rue Philippe Lebon, Le Havre, France.
*Corresponding author: bilal.kanso@hotmail.com, bilal_kanso@hotmail.com

Article published by EDP Sciences

an order of tasks and finally ends at the last served task; (5) routes must verify the capacity constraint. Thus, OCARP aims to find at most M set of feasible routes with minimum total cost distance.

The authors in [17] proposed a heuristic based on a Reactive Path-Scanning (RPS) with ellipse rule to solve OCARP and found the first known solutions. In [18], the authors developed an exact branch-and-bound algorithm and improved the best known lower bounds. In [19], the authors developed a greedy randomized adaptive search procedure (GRASP) with evolutionary path-relinking which slightly improved the solutions found in [18]. In the last work on OCARP [1], the authors developed Hybrid Genetic Algorithm (HGA) and provided the best metaheuristic solutions for OCARP.

In this paper, we introduce a new approach based on Hybrid Ant Colony Algorithm (HACA). The representation of ants is considered as a giant route containing all tasks of E_r in such a way each task is served in one direction. Each ant is represented by only one vehicle and does not verify the capacity constraint. An evaluation procedure adapted from the splitting algorithm introduced in [16] is developed to make the ants feasible and then evaluate them. Our method is hybridized with a local search procedure based on Simulated Annealing (SA) which operates by sequentially performing moves to existing feasible solutions in order to eventually improve them as the search progresses [9]. The main move used in SA algorithm is the cross-exchange which is adapted for OCARP for the first time in this work. Finally, experiments that we have conducted are capable of solving significantly larger instances of the OCARP existing in the literature [2, 6] and found good (some are optimal) solutions without expending much computational execution time.

This paper is organized as follows. Section 2 describes our studied problem OCARP. Section 3 presents our hybrid HACA algorithm which is a combination of an ant colony method with a local search procedure based on Simulated Annealing (SA) to efficiently solve OCARP. Section 4 shows the computational experiments that we conducted and analyses the results. Finally, a conclusion about this work is summarized in Section 5.

2. Problem description

The Open Capacited Arc Routing Problem (OCARP) was introduced and described in [17]. In this variant of CARP, each vehicle starts in its first task (required edge with a demand strictly positive) and finish at the last visited task. The number of vehicles is limited and each task can be visited once by a specific vehicle. The capacity of vehicles Q is predefined and homogeneous for all used vehicles. A travelling distance C_{pq} is considered between each couple of tasks (p, q). A feasible solution is a set of feasible routes where each feasible route is a sequence of tasks visited and serviced by the same vehicle considering the capacity constraint. The main objective of OCARP is to minimize the total travelling distance by all feasible routes in the feasible solution. Other objective functions and constraints can be considered, like time windows and heterogeneous vehicles, but they are out of our work.

Therefore, the details of OCARP can be described as follows: let K be a set of M identical vehicles and $G = (N, E, E_r)$ be a graph where N is the node set that represents the starting and ending points of edges and $E = E_r \cup E_{nr}$ is the edge set. E_r contains the required edges (called *tasks*) where each task p represents an edge with demand $d_p > 0$ and must be served by one and only one vehicle $k \in K$. E_{nr} is the set of all no required edges with demand zero. Firstly, a travelling distance D_{xy} between nodes is associated to each couple $(x, y) \in N \times N$. The cost C_{pq} between two tasks $p = (x, y)$ and $q = (z, t)$ will be calculated by Dijkstra's algorithm from the final node y of p to the origin node z of q. The notation of the cost $C_p = D_{xy}$ is used to traverse the task $p = (x, y)$, $\forall p \in E_r$ where $(x, y) \in N \times N$. The notation $Cost(S)$ is used to get the total cost of a given solution S which is considered as the total sum of costs of all routes involved in the solution S.

3. An ant colony algorithm for the OCARP

We propose an ant colony algorithm to solve the OCARP. Ant Colony Optimization algorithms (ACO) are global search optimization techniques based on the foraging behaviour of real ants which have been found to be very effective for solving the travelling salesman problem (TSP) [3, 5, 15] and the capacitated arc routing problem (CARP) [7]. In this work, we construct an initial solution S_0 by the basic Path Scanning algorithm

(PS) [17]. The ACO algorithm uses a colony of N_a artificial ants where each ant g represents a non-capacitated giant route (order of all different tasks in E_r), *i.e.* the constraint capacity is not verified. At each iteration of the ACO, every ant starts its route at a random task. Moving of ants depends on two amounts: (1) visibility amount μ_{pq} which is a constant and represents the inverse of the cost between two tasks p and q; (2) pheromone amount τ_{pq} which is a variable and initially represents the inverse of the cost of initial solution, *i.e.* $\tau_{pq} = \frac{1}{Cost(S_0)}$ $\forall p, q \in E_r$. The amount of pheromone will be updated at the end of each iteration t by the following rule $\tau_{pq}(t+1) = \rho\tau_{pq}(t) + (1-\rho)\Delta\tau_{pq}$ where ρ is the evaporation parameter in $[0, 1]$ and $\Delta\tau_{pq}$ equals to $\frac{1}{Cost(best)}$ if task q is used just after task p by the ant that gives the best solution *best* and zero otherwise.

The following subsection explains how to find an initial solution S_0 by Path Scanning heuristic. The second subsection presents the evaluation procedure to split and make feasible a giant route. The four moves used in the local search procedure and the detailed algorithm are explained in the third subsection. Finally, the components and the pseudo-code of the ACO algorithm are presented in the last subsection.

3.1. Path Scanning heuristic for OCARP problem

The classic Path Scanning heuristic (PS) is used for building the feasible initial solution S_0 of our proposed ACO metaheuristic. PS constructs the routes in a sequential manner, *i.e.* it begins to build the first route without exceeding the capacity of the vehicle used on this route. Once a route is constructed, PS tries to build other routes in the same way. Note that, when the number of vehicles exceeds the predefined number of vehicles M, then the obtained solution is not taken into consideration.

PS builds each feasible route as follows. Firstly, PS selects a random task to be the starting one of the constructed route. Then , it chooses the next task s to add after the last selected task l among the candidate tasks using the nearest neighbour rule. If there are more than one task that meet the capacity constraint and have the same cost from l, then one rule of the following five rules is applied: (1) minimize $\frac{C_s}{d_s}$; (2) maximize $\frac{C_s}{d_s}$; (3) minimize the cost back to a virtual depot 0; (4) maximize the cost back to a virtual depot 0; and (5) use rule (3) if the vehicle has used more than half of its capacity Q, and rule (4) otherwise.

The used virtual depot 0 represents the node 1 in the node set N in each graph instance G used in the numerical results (instance files are explained in Sect. 4). Indeed, the node 1 is used to be the first depot in the basic instance files of CARP with single depot. Furthermore, in our experiments each node in N has been tested to be the virtual depot 0, but no better results are attained. The heuristic PS is repeated 1000 times for each instance and the best costs of feasible obtained solutions are recorded (see results in Sect. 4). Finally, the best solution with the minimal total cost distance is the initial feasible solution S_0 used for ACO.

3.2. Evaluation procedure

The split procedure is commonly used for capacitated arc routing problem CARP [10, 16]. We adopt it to make each ant feasible by splitting it into a set of feasible routes (*i.e.* a feasible OCARP solution) and to evaluate its fitness. Given an ant $g = (g_1, g_2, \ldots, g_r)$ of r tasks, we construct an acyclic auxiliary graph $H(T, Z)$ with $r + 1$ nodes, where T is a vertex set containing nodes indexed from 0 to r, and Z is a set of evaluated directed arcs. 0 is a fictitious node and every node $i \in T \setminus \{0\}$ represents the task g_i in g. An arc (i, j) such that $i < j$, is associated to Z if it represents a feasible route *i.e.* a subsequence of tasks $(g_{i+1}, \ldots, g_j)$ where $\sum_{k=i+1}^{j} d_{g_k} \leq Q$. Every arc $(i, j) \in Z$ will be evaluated by the cost z_{ij} which is calculated by the following rule:

$$z_{ij} = \begin{cases} C_{g_j} + \sum_{k=i}^{j-2} [C_{g_{k+1}} + C(g_{k+1}, g_{k+2})] & \text{if} \quad j - i \geq 2 \\ C_{g_j} & \text{if} \quad j - i = 1. \end{cases}$$

For each ant g, the shortest path SP^g from node 0 to node r of T in graph H by using at most M arcs of Z, is calculated by Algorithm 1. This algorithm is an adaptation of Bellman–Ford algorithm [4] to take into consideration at most M arcs on the target shortest path. Hence, the shortest path SP^g represents the optimal partition of the corresponding ant g (giant route). The fitness cost of the optimal solution is denoted by $Cost(S^g)$ where S^g is the corresponding feasible OCARP solution from ant g and $Cost(S^g) = \sum_{(i,j)\in \mathrm{SP}^g} z_{ij}$.

Algorithm 1: Shortest path with at most k arcs (vehicles).

Input: an auxiliary graph $H(T, Z)$, source node src, number of arcs M
Output: array $dist$ with $dist[i]$ is the shortest distance from node src to node i, matrix $pred$ with $pred[i][j]$ is the predecessor of node j at iteration $i (i \leq k)$
Objective: find shortest path in A from src to other nodes using at most M arcs
Initialization:
temporary array of previous distances $distTemp$
$distTemp[i] = \infty, dist[i] = \infty$ for each node $i \neq src$ in A and $distTemp[src] = dist[src] = 0$
$pred[i][j] = -1$ if $j = s$ otherwise ∞ for each node i and each iteration j
$iter = 0$

```
for iter ← 1 to k do
    for s ← 0 to |T| − 1 do
        distTemp[s] = dist[s]
    for (u, v) ∈ Z do
        if z_uv ≠ ∞ then
            if distTemp[u]! = ∞ and dist[v] > distTemp[u] + z_uv then
                dist[v] = distTemp[u] + z_uv
                pred[iter][v] = u
```

Table 1 illustrates an example of six tasks g_1, g_2, g_3, g_4, g_5 and g_6 required to explain the details of our split procedure. The number of vehicles is $M = 2$ and the vehicle capacity is $Q = 20$. Suppose that the costs among tasks are given as follows: $C_{g_1g_2} = 33, C_{g_2g_3} = 38, C_{g_3g_4} = 27, C_{g_4g_5} = 32$ and $C_{g_5g_6} = 30$.

Figure 1A illustrates the auxiliary graph $H(T, Z)$ associated to the following giant route g (or ant g):

g_1	g_2	g_3	g_4	g_5	g_6

As two examples, the arc $(0, 1)$ represents a feasible route containing g_1 (*i.e.* only g_1 is served) with cost $z_{01} = C_{g_1} = 22$ and the arc $(0, 2)$ means that a vehicle serves g_1 then g_2 with cost $z_{02} = C_{g_1} + C_{g_1g_2} + C_{g_2} = 22 + 33 + 20 = 75$. Figures 1B and 1C illustrate two routes generated from the auxiliary graph using our adapted Bellman–Ford algorithm presented in Algorithm 1. For the proposed example, SP^g is composed of two arcs $(0, 2)$ and $(2, 6)$ and then the corresponding feasible solution S^g is composed of two routes presented respectively in Figures 1B and 1C. The cost of S^g is $Cost(S^g) = z_{02} + z_{26} = 75 + 175 = 250$.

TABLE 1. Information about an example with 6 tasks.

Task	g_1	g_2	g_3	g_4	g_5	g_6
Demand	8	8	5	6	4	5
Cost	22	20	20	25	26	15

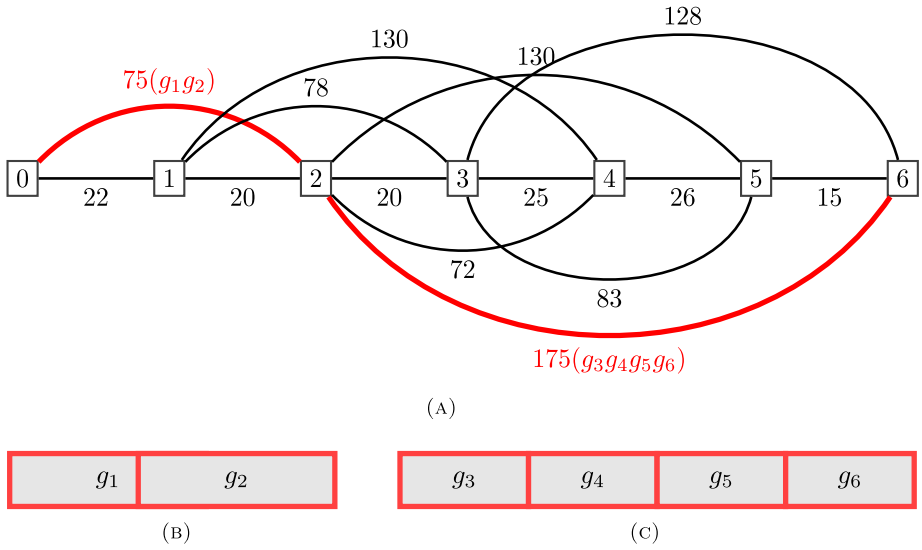

FIGURE 1. Example of the split procedure. (A) Auxiliary graph. (B) First route (cost = 75). (C) Second route (cost = 175).

3.3. Local search procedure

In order to improve the obtained feasible solutions by the proposed ACO algorithm, we apply the Simulated Annealing algorithm (SA) in the local search procedure. The SA is a metaheuristic which is commonly used to prevent the optimization algorithm to quickly fall into local minimum [12, 20]. It is based on the random acceptance strategy with certain probability. The local search moves that we implemented in the SA algorithm are the following:

(a) **Swap move**: two versions of this move have been implemented. The first one consists of exchanging two distinct tasks p and q belonging to the same feasible route. The second one consists of exchanging two distinct tasks belonging to two different feasible routes. For each call of the "Swap" move, we choose randomly the way to apply it.
(b) **Relocate move**: this move consists of removing one task from a position i on a route R and putting it in another position j chosen in the same route R or in another route R'.
(c) **2-opt move**: this move consists of choosing two positions i and j in a given route R and then re-organizes R by replacing the sequence of tasks $(i,\ i+1, \ldots,\ j)$ by the sequence $(inv(j),\ inv(j-1), \ldots,\ inv(i))$ where $inv(t)$ represents the inverse task for the task t. For each call of the "2-opt" move, we apply it to every route into the considered solution.
(d) **Cross-exchange move**: this move involves two routes R_1 and R_2 where it chooses a task in a position i of R_1 and a task in a position j of R_2. It generates two new routes R'_1 and R'_2 as follows: R'_1 takes the first part of R_1 from first task to task located at position i completed by another part of the route R_2 starting with the task located at position j. R'_2 is built in the same way and that by linking the remaining first part of R_2 with the second remaining part of R_1.

Note that all moves are applied only if the resulting solutions are feasible. The Simulated Annealing method in the local search procedure is presented in Algorithm 2.

Algorithm 2: Simulated Annealing algorithm.

Input: a feasible solution $S, T_{\min}, T_{\max}$ and K
Initialization: $T = T_{\max}$
while $T < T_{\min}$ **do**
  for $l \in \{$*Swap, Relocate, 2-opt, Cross-exchange*$\}$ **do**
    generate a feasible neighbourhood solution S' by applying the local search move l.
    calculate $\Delta E = Cost(S') - Cost(S)$ and the probability $P(\Delta E, T) = e^{-\frac{\Delta E}{T}}$
    if $\Delta E < 0$ **then**
      accept S' and update the best solution $S = S'$
    else
      accept S' with the probability $P(\Delta E, T)$
  update the temperature $T = K \times T$
return the best solution found so far S.

3.4. The proposed HACA algorithm

Suppose that $Visited_g$ is the set of already visited tasks by ant g. An ant g moves from the last visited task i to another task j by choosing the closest task j ($j \notin Visited_g$) to i if the random variable q is strictly less than q_0. Otherwise, it is chosen by using the following probability rule:

$$P_{ij}^{g} = \begin{cases} \frac{\tau_{ij}^{\alpha} \times \mu_{ij}^{\beta}}{\sum_{k \in Visited_g} \tau_{ik}^{\alpha} \times \mu_{ik}^{\beta}} & \text{if} \quad j \notin Visited_g \\ 0 & \text{otherwise} \end{cases}$$

where α and β are respectively the pheromone and visibility parameters, q_0 is a constant parameter belongs to $[0, 1]$ and q is a random parameter in $[0, 1]$. The proposed HACA algorithm is shown in Algorithm 3.

Algorithm 3: HACA algorithm.

Initialization:
find a feasible initial solution S_0 with cost $Cost(S_0)$ by the classic heuristic PS
$\tau_{ij} = \frac{1}{Cost(S_0)}$ and $\mu_{ij} = \frac{1}{C_{ij}}$, $\forall i, j \in$ set of tasks
make empty N_a ants, define α, β, ρ, p_r and $iterA$
$iter = 1$
while $iter < iterA$ **do**
  every ant g starts its giant route randomly by a task g_1
  every g completes the route by using the probability P_{ij}^{g}
  evaluate each ant by using the evaluation procedure
  apply the local search procedure on each ant with a probability p_r
  update the best solution so far
  update the pheromone quantities
  $iter = iter + 1$
return the best solution found so far.

TABLE 2. Results of the HACA algorithm on ogdb's instances.

Instance	M	lb	RPS	GRASP	HGA	HACA	G–R	G–G	G–H	T
ogdb1	7	252	252	252	252	252	0	0	0	<1
ogdb2	8	291	291	291	291	291	0	0	0	<1
ogdb3	7	233	233	233	233	233	0	0	0	<1
ogdb4	6	238	238	238	238	238	0	0	0	<1
ogdb5	8	316	316	316	316	316	0	0	0	<1
ogdb6	7	260	260	260	260	260	0	0	0	<1
ogdb7	7	262	262	262	262	262	0	0	0	<1
ogdb8	12	210	210	210	210	210	0	0	0	<1
ogdb9	12	219	219	219	219	219	0	0	0	<1
ogdb10	6	252	252	252	252	252	0	0	0	<1
ogdb11	7	356	360	358	358	358	0	0	0	<1
ogdb12	9	336	336	336	336	336	0	0	0	<1
ogdb13	8	509	509	509	509	509	0	0	0	<1
ogdb14	7	96	96	96	96	96	0	0	0	<1
ogdb15	6	56	56	56	56	56	0	0	0	<1
ogdb16	7	119	119	119	119	119	0	0	0	<1
ogdb17	7	84	84	84	84	84	0	0	0	<1
ogdb18	7	158	158	158	158	158	0	0	0	<1
ogdb19	5	45	45	45	45	45	0	0	0	<1
ogdb20	6	105	105	105	105	105	0	0	0	<1
ogdb21	8	149	149	149	149	149	0	0	0	<1
ogdb22	10	191	191	191	191	191	0	0	0	<1
ogdb23	12	223	223	223	223	223	0	0	0	<1

4. EXPERIMENTS AND RESULTS

4.1. Benchmark instances and experiments

For our computational experiments, we use five sets of standard test instances: ogdb, oAi, oBi, oval and oegl. These sets are extensions of standard sets of CARP[1] instances. Thus, our OCARP Benchmark includes 23 ogdb [8], 32 oAi, 24 oBi, 34 oval [2] and 24 oegl [6], totalling 137 instances. $M = M^* + 2$ is the number of vehicles used in our experiments where M^* represents the minimal value of the number of vehicles required for finding a feasible solution reported in [17]. The experiments were implemented in Java and run on a 2.6 GHz Dual Core computer with a memory of 16 GB under Windows Ten.

Throughout the section, the results of our experiments are compared with respect to three methods from literature: RPS (Reactive Path-Scanning with ellipse rule) heuristic [17], GRASP (Greedy Randomized Adaptive Search Procedure with path relinking) metaheuristic [19] and HGA (Hybrid Genetic Algorithm) [1]. Tables 2–6 show the obtained results of all instances after running our algorithm. For each table, the columns correspond respectively to: (1) problem name instance; (2) the number of vehicles M; (3) lower bound (lb) which is the corresponding bound to the best feasible solution as reported in [17]; (4) cost obtained by RPS method; (5) cost obtained by GRASP method; (6) cost obtained by HGA method; (7) cost obtained by our hybrid ant algorithm (HACA); (8) gap in percentage (G–R(%)) of RPS cost compared to HACA; (9) gap in percentage (G–G(%)) of GRASP cost compared to HACA; (10) gap in percentage (G–H(%)) of HGA cost compared to HACA; (11) running time in seconds for HACA (T). The gaps G–R(%), G–G(%) and G–H(%) are computed by the following rule: $\text{G–X}= \frac{\text{X–HACA}}{\text{X}} \times 100$ with $\text{X} \in \{\text{RPS}, \text{GRASP}, \text{HGA}\}$.

[1]CARP's instances can be found at http://www.uv.es/Belengue/carp.html; http://www.hha.dk/sanw.

TABLE 3. Results of the HACA algorithm on oAi's instances.

Instance	M	lb	RPS	GRASP	HGA	HACA	G–R	G–G	G–H	T
oAi10A	6	43	43	43	43	43	0	0	0	<1
oAi10B	5	43	43	43	43	43	0	0	0	<1
oAi10C	4	43	43	43	43	43	0	0	0	<1
oAi10D	3	43	45	43	43	43	4.44	0	0	<1
oAi13A	10	85	85	85	85	85	0	0	0	<1
oAi13B	6	85	85	85	85	85	0	0	0	<1
oAi13C	4	85	88	85	85	85	3.41	0	0	<1
oAi13D	4	85	91	85	85	85	6.59	0	0	<1
oAi15A	10	92	92	92	92	92	0	0	0	<1
oAi15B	7	92	92	92	92	92	0	0	0	<1
oAi15C	5	92	92	92	92	92	0	0	0	<1
oAi15D	4	92	94	92	92	92	2.13	0	0	<1
oAi20A	13	113	113	113	113	113	0	0	0	<1
oAi20B	9	113	113	113	113	113	0	0	0	<1
oAi20C	6	113	113	113	113	113	0	0	0	<1
oAi20D	5	113	116	113	113	113	2.59	0	0	<1
oAi24A	14	139	139	139	139	139	0	0	0	<1
oAi24B	9	139	139	139	139	139	0	0	0	<1
oAi24C	6	139	151	145	145	145	3.98	0	0	<1
oAi24D	5	139	154	148	148	148	3.9	0	0	<1
oAi27A	12	188	188	188	188	188	0	0	0	<1
oAi27B	8	188	191	188	188	188	1.57	0	0	<1
oAi27C	5	194	202	197	197	197	2.48	0	0	1
oAi27D	4	197	208	202	202	202	2.88	0	0	<1
oAi31A	21	271	274	271	271	271	1.1	0	0	7.6
oAi31B	13	271	271	271	271	271	0	0	0	4.5
oAi31C	8	271	282	277	277	277	1.77	0	0	5.4
oAi31D	6	271	292	284	284	284	2.74	0	0	4
oAi40A	27	329	385	355	355	355	8.4	0	0	151.5
oAi40B	15	329	334	329	329	329	1.5	0	0	63
oAi40C	9	329	345	336	336	336	0	0	0	43.1
oAi40D	7	329	356	344	344	344	2.61	0	0	16.3

4.2. Comparison based on benchmark sets

Table 7 reports the obtained average of all gaps and the average of computational times in seconds required for solving the problem by each method. From this table, we can conclude the following:

- For ogdb's set of instances, HACA obtained the same results as all the other methods. The gaps G–R, G–G and G–H are equal to zero. 21 from 23 solutions (91.3%) were proven optimal *i.e.* the value obtained by HACA is the same as the lower bound lb. The CPU computing time was less than one second on average.
- For oAi's set of instances, 20 from 32 solutions (62.5%) were proven optimal (*i.e.* HACA = lb). The computational time average is 9.98 s. HACA significantly performs better than RPS where G–R is 1.75%. In fact, it outperforms RPS method by 16 solutions and obtains the same solutions as RPS for the remaining instances. HACA slightly performs better than both GRASP and HGA methods where G–G and G–H are both equal to 0.15. It outperforms both GRASP and HGA methods by one solution and attains the same results obtained by GRASP and HGA methods for the rest of instances.
- For oBi's set of instances, 22 from 24 solutions (91.667%) were proven optimal (*i.e.* HACA = lb). The CPU computing time was 3.3 s on average. HACA clearly performs better than RPS where G–R is 0.44%. Indeed,

TABLE 4. Results of the HACA algorithm on oBi's instances.

Instance	M	lb	RPS	GRASP	HGA	HACA	G–R	G–G	G–H	T
oBi10B	7	45	45	45	45	45	0	0	0	<1
oBi10C	5	45	45	45	45	45	0	0	0	<1
oBi10D	4	45	45	45	45	45	0	0	0	<1
oBi13B	9	60	60	60	60	60	0	0	0	<1
oBi13C	6	60	60	60	60	60	0	0	0	<1
oBi13D	5	60	60	60	60	60	0	0	0	<1
oBi15B	9	74	74	74	74	74	0	0	0	3
oBi15C	6	74	74	74	74	74	0	0	0	<1
oBi15D	5	74	74	74	74	74	0	0	0	<1
oBi20B	11	99	99	99	99	99	0	0	0	<1
oBi20C	7	99	99	99	99	99	0	0	0	<1
oBi20D	5	99	102	99	99	99	2.94	0	0	<1
oBi24B	11	109	109	109	109	109	0	0	0	1
oBi24C	7	107	110	107	107	107	0	0	0	<1
oBi24D	5	113	116	113	113	113	0	0	0	7
oBi27B	18	188	188	188	188	188	0	0	0	<1
oBi27C	10	185	185	185	185	185	0	0	0	2
oBi27D	8	185	185	185	185	185	0	0	0	9
oBi31B	21	274	282	282	282	282	0	0	0	3
oBi31C	12	274	274	274	274	274	0	0	0	<1
oBi31D	9	274	277	274	274	274	1.08	0	0	2
oBi40B	23	267	275	267	267	267	2.9	0	0	2
oBi40C	13	267	272	267	267	267	1.84	0	0	23
oBi40D	9	267	279	274	274	274	1.79	0	0	13

it outperforms RPS method in 5 out of 24 instances and obtains the same results obtained by GRASP and HGA where G–G and G–H are both equal to 0.

– For oval's set of instances, 8 from 34 solutions (23.53%) were proven optimal (*i.e.* HACA = lb). The CPU computing time was 7.8 s on average. HACA clearly performs better than RPS where G–R is 1.64%, outperforms RPS method by 28 solutions and obtains the same solutions as RPS method for the rest of instances. HACA exactly performs as both GRASP and HGA methods where the G–G and G–H are both equal to zero.
– For oegl's set of instances, HACA fails to reach lb for all instances (similarly as RPS, GRASP and HGA methods). The CPU computing time was 335.08 s on average. HACA outperforms the RPS method for all the 24 solutions and G-RPS on average is 9.45. HACA succeeds to perform better than both GRASP and HGA in one solution ($s1B$). It obtains the same results as GRASP and HGA for seven solutions. The G–G is −1.81 and the G–H is −1.94 which are considered as very slight gaps.

By analysing the results presented in Table 7, it can be deduced that our method HACA similarly performs or better than RPS, GRASP and HGA for four sets (*i.e* ogdb, oval, oAi and oBi), significantly outperforms RPS for oegl set and slightly gets worse results than both GRASP and HGA for some oegl's instances.

RPS, GRASP and HGA were executed at 2.4 GHz processor while HACA was executed at 2.6 GHz processor. To give a fair comparison that takes into consideration the different speeds of the machines used in the computational experiments, we normalized the average computational time obtained by HACA. This is done by multiplying the recorded time with the CPU speed ratio which equals to CPU speed of used machine over CPU speed of the slowest machine in other studies (*i.e.* 2.4 GHz) as proposed in [11,13]. The CPU speed ratio is then equal to 1.08. Hence, in Table 7, we added the column "NT-HACA" which indicates the normalized

TABLE 5. Results of the HACA algorithm on oval's instances.

Instance	M	lb	RPS	GRASP	HGA	HACA	G–R	G–G	G–H	T
oval1A	4	146	154	149	149	149	3.25	0	0	<1
oval1B	5	146	149	147	147	147	1.43	0	0	<1
oval1C	10	146	146	146	146	146	0	0	0	<1
oval2A	4	185	195	189	189	189	3.08	0	0	<1
oval2B	5	185	192	186	186	186	3.13	0	0	<1
oval2C	10	185	185	185	185	185	0	0	0	<1
oval3A	4	65	71	67	67	67	5.63	0	0	<1
oval3B	5	65	67	66	66	66	0	0	0	<1
oval3C	9	65	65	65	65	65	1.49	0	0	<1
oval4A	5	344	358	350	350	350	2.23	0	0	<1
oval4B	6	343	354	347	347	347	1.98	0	0	6
oval4C	7	343	350	345	345	345	1.43	0	0	<1
oval4D	11	343	346	343	343	343	0.87	0	0	<1
oval5A	5	367	381	374	374	374	0	0	0	<1
oval5B	6	367	376	371	371	371	1.84	0	0	2
oval5C	7	367	372	368	368	368	1.08	0	0	3
oval5D	11	367	370	367	367	367	0.82	0	0	<1
oval6A	5	190	195	192	192	192	1.54	0	0	2
oval6B	6	190	192	191	191	191	0.52	0	0	<1
oval6C	12	190	190	190	190	190	0	0	0	<1
oval7A	5	249	263	256	256	256	2.66	0	0	1
oval7B	6	249	259	253	253	253	2.32	0	0	7
oval7C	11	249	250	249	249	249	0.4	0	0	32
oval8A	5	347	359	354	354	354	1.39	0	0	2
oval8B	6	347	354	351	351	351	0.85	0	0	4
oval8C	11	347	348	347	347	347	0.29	0	0	8
oval9A	5	278	299	292	292	292	2.34	0	0	19
oval9B	6	278	298	290	290	290	2.68	0	0	7
oval9C	7	278	294	288	288	288	2.04	0	0	13
oval9D	12	278	288	280	280	280	0	0	0	21
oval10A	5	376	403	391	391	391	2.98	0	0	9
oval10B	6	376	399	388	388	388	2.76	0	0	34
oval10C	7	376	394	385	385	385	2.28	0	0	23
oval10D	12	376	387	377	377	377	2.58	0	0	57

average running time of HACA. It is clear that HACA is many times faster than RPS, GRASP and HGA for all instances. For example, the overall time performance for oegl's set of instances is 5.49, 2.86 and 2.67 faster than RPS, GRASP and HGA respectively.

Table 8 shows for each set of instances and for each method the number of same solutions with the same cost, the number of worse solutions and the number of better solutions in comparison with HACA. Table 8 shows for each set of instances and for each method the number of same solutions with the same cost, the number of worse solutions and the number of better solutions in comparison with HACA. HACA overcomes RPS in 74 instances and obtains the same solutions in 63 instances. HACA outperforms GRASP in one instance, obtains the same solutions in 120 instances and is overcame by GRASP in 16 solutions. Finally, HACA outperforms HGA in one instance, obtains the same solutions as it in 120 instances and is outperformed by HGA in 16 solutions.

TABLE 6. Results of the HACA algorithm on oegl's instances.

Instance	M	lb	RPS	GRASP	HGA	HACA	G–R	G–G	G–H	T
oegl-e1-A	7	1590	1755	1659	1659	1659	5.47	0	0	99
oegl-e1-B	9	1524	1726	1589	1589	1589	7.94	0	0	125
oegl-e1-C	12	1490	1610	1542	1542	1542	4.22	0	0	165
oegl-e2-A	9	1965	2256	2035	2035	2043	9.44	−0.39	−0.39	119
oegl-e2-B	12	1912	2166	1971	1971	1971	9	0	0	181
oegl-e2-C	16	1879	2151	1964	1964	1964	8.69	0	0	189
oegl-e3-A	10	2245	2676	2372	2366	2382	10.99	−0.42	−0.68	171
oegl-e3-B	14	2203	2596	2321	2321	2350	9.48	−1.25	−1.25	286
oegl-e3-C	19	2188	2565	2270	2260	2280	11.11	−0.44	−0.88	181
oegl-e4-A	11	2453	2825	2556	2554	2573	8.92	−0.67	−0.74	344
oegl-e4-B	16	2453	2853	2517	2517	2537	11.08	−0.79	0.79	237
oegl-e4-C	21	2453	2805	2491	2497	2524	10.02	−1.32	−1.08	371
oegl-s1-A	9	1503	1787	1604	1604	1604	10.24	0	0	399
oegl-s1-B	12	1426	1729	1579	1579	1565	9.49	0.89	0.89	144
oegl-s1-C	16	1397	1757	1512	1512	1512	13.94	0	0	222
oegl-s2-A	16	3205	4068	3567	3566	3665	9.91	−2.75	−2.78	354
oegl-s2-B	22	3174	4009	3442	3428	3595	10.33	−4.45	−4.87	392
oegl-s2-C	29	3174	3904	3341	3340	3534	9.48	−5.78	−5.81	651
oegl-s3-A	17	3381	4242	3734	3704	3776	10.99	−1.12	−1.94	612
oegl-s3-B	24	3379	4158	3564	3558	3704	10.92	−3.93	−4.1	439
oegl-s3-C	31	3379	4102	3492	3495	3588	12.53	−2.75	−2.66	197
oegl-s4-A	21	4186	4965	4409	4399	4544	8.48	−3.06	−3.3	484
oegl-s4-B	29	4186	4973	4323	4312	4523	9.05	−4.63	−4.89	552
oegl-s4-C	37	4186	5019	4309	4282	4536	9.62	−5.27	−5.93	1128

TABLE 7. Gap and time comparisons of HACA with RPS, GRASP and HGA.

Set	G–R	G–G	G–H	T–RPS	T–GRASP	T–HGA	T–HACA	NT–HACA
ogdb	0	0	0	66.2	<0.2	<0.1	<0.1	<1
oAi	1.63	0	0	447.7	17.2	40.1	9.98	10.077
oBi	0.44	0	0	345.8	32	32	3.3	3.564
oval	1.64	0	0	756.1	71.8	12	7.8	8.42
oegl	9.45	−1.81	−1.94	1989.6	1034.5	968.1	335.08	361.9

TABLE 8. Comparison of different obtained solutions of HACA with RPS, GRASP and HGA.

		RPS			GRASP			HGA		
Set	nb	Same	Worse	Better	Same	Worse	Better	Same	Worse	Better
ogld	23	23	0	0	23	0	0	23	0	0
oAi	32	16	0	16	32	0	0	32	0	0
oBi	24	18	0	6	24	0	0	24	0	0
oval	34	6	0	28	34	0	0	34	0	0
oegl	24	0	0	24	7	16	1	7	16	1
Total	137	63	0	74	120	16	1	120	16	1

TABLE 9. Comparison of HACA with RPS, GRASP and HGA.

	Class 1	Class 2	Class 3
	$\lvert E_r\rvert < 50$	$50 \leq \lvert E_r\rvert \leq 100$	$\lvert E_r\rvert > 100$
ogdb	22	1	0
oAi	24	8	0
oBi	21	3	0
oval	9	25	0
oegl	0	15	9
Total	76	52	9
G–R	0.45	3.9	9.63
G–G	0	−0.08	−4.35
G–H	0	−0.09	−4.63
Final gap	0.15	1.24	0.21

4.3. Comparison based on instance sizes

The purpose of this subsection is to provide a new comparison of experiment results based on the size of the instances. Indeed, there is still a need to make a reasonably consistent and fair comparison taking into consideration the number of tasks in each instance. To do so, we classify all instances presented in Section 4.1 into three classes according to the number of tasks $|E_r|$. The first class contains all instances whose number of tasks is less than 50. The second class contains all instances whose number of tasks is between 50 and 100. The third class contains all instances whose number of tasks is greater than 100. Table 9 represents the number of instances in each class and the new gaps. The total number of instances belonging to the first class is 76 and HACA reaches a much better performance than RPS, GRASP and HGA where the gaps are respectively 0.45, 0 and 0. The total number of instances belonging to the second class is 52 and HACA gets better results than RPS (G–R is 3.9) and obtains almost the same results as GRASP and HGA where both gaps G–G (−0.08) and G–H (−0.09) are of negligible significance. Finally, the number of instances in the third class is 9 and HACA succeeds to surpass RPS where G–R is 9.63 and slightly gets worse than GRASP and HGA with G–G is −4.35 and G–H is −4.63.

The experiments that we conducted using HACA have shown the production of a population of infeasible solutions at high rates because of number of vehicles constraint. The algorithm maintains the population in only the feasible search space, thus excluding infeasible solutions during the algorithm evolution. This significantly reduces the quality of solutions since the search space is restricted to feasible region. Thus, our proposed algorithm fails to find solutions for some instances with M^* and $M^* + 1$ and when finding such solutions, their costs are not competitive with those reported in [1]. The authors in [1] showed that the feasibility rate for almost all instances with the number of vehicles M^* and $M^* + 1$ is very small and this rate is higher when running large-scale problems (*i.e.* Eglese's instances). Then, they equipped their method with a feasibility technique in order to overcome the presence of infeasible solutions with respect to the number of vehicles. That justifies our choice to perform our experiments only with number of vehicles $M^* + 2$ and why our algorithm HACA works efficiently and succeeds to obtain competitive results with both GRASP and HGA methods for small and medium instances (as shown in both class 1 and class 2 – Tab. 9), but fails to obtain good quality solutions for large instances (as shown in class 3 – Tab. 9).

4.4. Parameter setting

For each set of instances, several runs of the program with different parameter values were performed and the best obtained values are summarized in the following. The values of N_a, α, β, ρ and q_0 are found and recommended by Dorigo *et al.* [5]. We use between 500 and 2500 maximum iterations N_{iter} in HACA algorithm.

In the Simulated Annealing algorithm, K is tested between 0.5 and 0.9, $T_{\max}$ is tested between 180 and 260 and $T_{\min}$ is tested between 10 and 30. The parameter values obtaining the best improvement solutions with an acceptable computational time are $N_a = 20, \alpha = 1, \beta = 2, \rho = 0.85, q_0 = 0.6, N_{\text{iter}} = 1750, K = 0.7, T_{\max} = 200$ and $T_{\min} = 10$.

5. Conclusion

Open Capacitated Arc Routing Problem (OCARP) has important real-world applications like the Meter Reader Routing Problem (MRRP), but it has not been extensively studied in literature. We have adapted a constructive heuristic to find an initial solution for OCARP. We used the strategy of a giant route and then an optimal splitting method is applied to make the solution feasible and then evaluate it. We have proposed an ant colony algorithm hybridized with a local search method based on four important moves to solve this problem. We have presented results for different OCARP instances on five different sets from literature. The obtained results are very good in comparison with the well-known Reactive Path Scanning (RPS) which is guided by a cost-demand edge-selection and ellipse rules. Comparing the solution quality of the proposed algorithm with the existing meta-heuristics GRASP and HGA in literature, the computational results indicate that HACA performs very good and provides competitive results.

Extending description of OCARP problem to cover more operational constraints such as maximum worker time or customer time windows could be a promising direction to be more studied in the future. We also intend to equip our approach with a feasibility method in order to tackle instances with number of vehicles M^* and $M^* + 1$.

Acknowledgements. This work has been jointly funded with the support of the National Council for Scientific Research in Lebanon CNRS-L and Lebanese University.

References

[1] R.K. Arakaki and F.L. Usberti, Hybrid genetic algorithm for the open capacitated arc routing problem. *Comput. Oper. Res.* **90** (2018) 221–231.

[2] E. Benavent, V. Campos, A. Corberán and E. Mota, The capacitated arc routing problem: lower bounds. *Networks* **22** (1992) 669–690.

[3] A. Colorni, M. Dorigo and V. Maniezzo, Distributed optimization by ant colonies (1991) 134–142.

[4] T.H. Cormen, C.E. Leiserson, R.L. Rivest and C. Stein, Introduction to Algorithms, 3rd edition. The MIT Press (2009).

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

[6] R.W. Eglese, Routing winter gritting vehicles. *Disc. Appl. Math.* **48** (1994) 231–244.

[7] B.L. Golden and R.T. Wong, Capacitated arc routing problems. *Networks* **11** (1981) 305–315.

[8] B.L. Golden, J.S. Dearmon and E.K. Baker, Computational experiments with algorithms for a class of routing problems. *Comput. Oper. Res.* **10** (1983) 47–59.

[9] S. Kirkpatrick, C.D. Gelatt and M.P. Vecchi, Optimization by simulated annealing. *Science* **220** (1983) 671–680.

[10] P. Lacomme, C. Prins and W. Ramdane-Chérif, Evolutionary algorithms for periodic arc routing problems. *Eur. J. Oper. Res.* **165** (2005) 535–553. Project Management and Scheduling.

[11] L. Muyldermans and G. Pang, A guided local search procedure for the multi-compartment capacitated arc routing problem. *Comput. Oper. Res.* **37** (2010) 1662–1673.

[12] I.H. Osman and C.N. Potts, Simulated annealing for permutation flow-shop scheduling. *Omega* **17** (1989) 551–557.

[13] L. Santos, J. Coutinho-Rodrigues and J.R. Current, An improved ant colony optimization based algorithm for the capacitated arc routing problem. *Trans. Res. B: Methodol.* **44** (2010) 246–266.

[14] H.I. Stern and M. Dror, Routing electric meter readers. *Comput. Oper. Res.* **6** (1979) 209–223.

[15] T. Stutzle and M. Dorigo, A short convergence proof for a class of ant colony optimization algorithms. *IEEE Trans. Evol. Comput.* **6** (2002) 358–365.

[16] G. Ulusoy, The fleet size and mix problem for capacitated arc routing. *Eur. J. Oper. Res.* **22** (1985) 329–337.

[17] F.L. Usberti, P.M. França and A.L.M. França, The open capacitated arc routing problem. *Comput. Oper. Res.* **38** (2011) 1543–1555.

[18] F.L. Usberti, P.M. França and A.L.M. França, Branch-and-bound algorithm for an arc routing problem. Annals XLIV SBPO, Rio de Janeiro (2012).

[19] F.L. Usberti, P.M. França and A.L.M. França, Grasp with evolutionary path-relinking for the capacitated arc routing problem. *Comput. Oper. Res.* **40** (2013) 3206–3217.

[20] P.J.M. van Laarhoven, E.H.L. Aarts and J.K. Lenstra, Job shop scheduling by simulated annealing. *Oper. Res.* **40** (1992) 113–125.

[21] J. Wunderlich, M. Collette, L. Levy and L. Bodin, Scheduling meter readers for Southern California gas company. *INFORMS J. Appl. Anal.* **22** (1992) 22–30.