Computers and Operations Research 90 (2018) 221–231

Contents lists available at ScienceDirect

Computers and Operations Research

journal homepage: www.elsevier.com/locate/cor

Hybrid genetic algorithm for the open capacitated arc routing problem

Rafael Kendy Arakaki, Fábio Luiz Usberti*

Institute of Computing, University of Campinas, Av. Albert Einstein 1251, 13083-852, Campinas, SP, Brazil

ARTICLE INFO

Article history:
Received 29 November 2016
Revised 15 September 2017
Accepted 18 September 2017
Available online 22 September 2017

Keywords:
Open capacitated arc routing problem
Hybrid genetic algorithm
Metaheuristic

ABSTRACT

The Open Capacitated Arc Routing Problem (OCARP) is an NP-hard arc routing problem where, given an undirected graph, the objective is to find the least cost set of routes that services all edges with positive demand (required edges). The routes are subjected to capacity constraints in relation to edge demands. The OCARP differs from the Capacitated Arc Routing Problem (CARP) since OCARP does not consider a depot and routes are not constrained to form cycles. A hybrid genetic algorithm with feasibilization and local search procedures is proposed for the OCARP. Computational experiments conducted on a set of benchmark instances reveal that the proposed hybrid genetic algorithm achieved the best upper bounds for almost all instances.

1. Introduction

The objective of arc routing problems consists of determining the least cost traversal of a given subset of edges in a graph, with one or more collateral constraints. Many real world applications are modeled as arc routing problems, such as street sweeping, garbage collection, mail delivery, meter reading, etc. Estimated expenditure on these services exceeds billions of dollars per year in the United States, thus revealing the economical importance of such problems (Corberán and Laporte, 2014; Corberán and Prins, 2010; Dror, 2012; Eiselt et al., 1995).

The Capacitated Arc Routing Problem (CARP) (Golden and Wong, 1981) is a combinatorial optimization problem defined on an undirected graph $G(V, E)$ with non-negative costs and demands associated to the edges. A fleet of identical vehicles with limited capacity is considered. The set of vehicles must service all edges with positive demand (*required edges*). The objective is to find a minimum cost set of routes that start and finish in a distinguished node, called depot. The CARP has been extensively studied over the last decades and we refer the reader to Prins (2014) and Muyldermans and Pang (2014) for a comprehensive survey.

A CARP variation consists in allowing both closed and open routes. An open route can use different nodes to start (*source*) and end (*sink*) the route, while a closed route is a cycle in which the source and sink nodes are necessarily the same. This problem in which routes are not constrained to form cycles is called the Open Capacitated Arc Routing Problem (OCARP) (Usberti et al., 2011). There are at least two problems that can be formulated as OCARP instances, the Meter Reader Routing Problem (Bodin and Levy, 1991; Stern and Dror, 1979; Wunderlich et al., 1992) and the Cutting Path Determination Problem (Moreira et al., 2007; Rodrigues and Ferreira, 2012).

The OCARP has been proved *NP-hard* (Usberti et al., 2011). Attempts were made to solve OCARP to optimality, including a branch-and-bound algorithm (Usberti et al., 2012) and an integer linear programming formulation (Usberti et al., 2011). These methods solved to optimality only small-sized instances (up to 27 nodes and 55 required edges). Thus it is of interest the development of heuristics that achieve good solutions in reasonable time in practice, which usually involve larger instances.

This work proposes a Hybrid Genetic Algorithm (HGA) for the OCARP. Computational experiments were performed to evaluate the proposed method. The results are compared to other heuristic methods from literature hence revealing the HGA good performance.

This paper is organized as follows. The OCARP is formally defined in Section 2. Section 3 presents a review of previous works for OCARP and applications. Section 4 describes the proposed HGA for OCARP. Section 5 contains computational experiments on a set of benchmark instances. Section 6 provides the final remarks.

2. Problem definition

The Open Capacitated Arc Routing Problem (OCARP) proposed by Usberti et al. (2011) is defined as follows. Let $G(V, E)$ be an undirected connected graph where non-negative costs c_{ij} and

* Corresponding author.
E-mail address: fusberti@ic.unicamp.br (F.L. Usberti).

https://doi.org/10.1016/j.cor.2017.09.020
0305-0548/

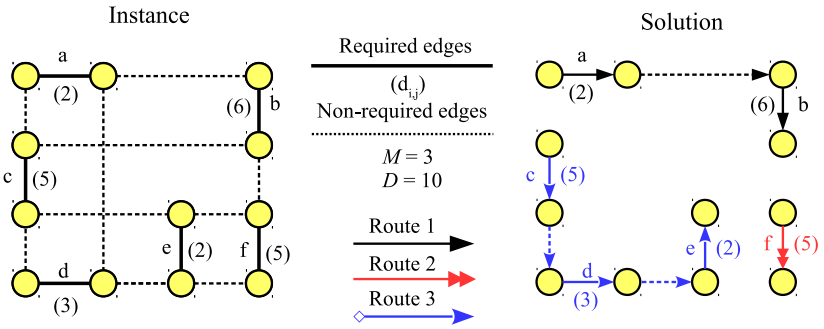

Fig. 1. OCARP: an instance and a feasible solution.

non-negative demands d_{ij} are assigned to each edge $(i, j) \in E$. If an edge (i, j) has positive demand $d_{ij} > 0$ then it is called a required edge. Let $E_R \subseteq E$ be the set of required edges. A fleet of M identical vehicles is available, each with capacity D. While traversing the graph, a vehicle might (i) service an edge, which deducts its capacity by the edge demand and increases the solution cost by the edge cost or (ii) deadhead an edge, which only increases the solution cost by the edge cost. A vehicle route is defined by a sequence of directed edges (arcs) traversed by the vehicle, and here both open and closed routes are considered, i.e., an OCARP route may start and end at distinct nodes.

A feasible OCARP solution is composed by at most M routes, which collectively service all required edges and do not exceed the vehicles capacity D (Fig. 1). OCARP aims to find a feasible set of routes with minimum cost.

The OCARP is a variant of the CARP that considers two important differences. First, OCARP allows routes to be open while CARP requires that all routes must start and end at a specific node (depot). Second, many CARP heuristics considers the number of vehicles M as decision variable, while in OCARP the number of vehicles is a fixed parameter. Instances with a very tight number of vehicles can be harder to solve or even to obtain good solutions, as will be discussed in Section 5.

3. Previous work

The OCARP was introduced by Usberti et al. (2011) and the authors presented an integer linear programming formulation to the problem. This formulation has an exponential number of variables and constraints, which could be the reason why the authors reported difficulties to solve medium and large instances. In the following work, Usberti et al. (2012) proposed a branch-and-bound algorithm that improved the known lower bounds. This algorithm exploited the relationship between OCARP and the Capacity and Degree Constrained Minimum Spanning Forest Problem (CDCMSFP).

With respect to heuristic approaches, Usberti et al. (2011) proposed a reactive path-scanning with ellipse rule (RPS) that obtained the first known solutions for OCARP. Afterwards Usberti (2012) proposed a greedy randomized adaptive search procedure (GRASP) with path-relinking that improved the known best solutions. Nevertheless, the largest instances still presented significant optimality gaps (greater than 30%).

Fung et al. (2013) presented a memetic algorithm for another problem, which was also called open capacitated arc routing problem. Their problem differs from the original OCARP on three points: (i) routes starts from a depot but need not to return to the depot; (ii) the number of vehicles is a decision variable with an associated cost; (iii) the graph and the required edges are directed.

3.1. Applications

3.1.1. Meter Reader Routing Problem

The Meter Reader Routing Problem (MRRP) considers the routing of employees responsible for metering electric, water and gas consumption data within urban areas. The employees are taken by car to the starting point of their routes and after finishing their routes they take public transport. The MRRP can be modeled as an OCARP, since the walking routes performed by the employees start and end at possibly different nodes. The objective is to minimize the routes travel-time and also consider working shifts. Stern and Dror (1979) consider a route-first cluster-second heuristic where initially a single non-capacitated route cover all required edges. The route is then partitioned, and each part is designated to a single employee. Bodin and Levy (1991) consider an arc partitioning algorithm that was later applied by Wunderlich et al. (1992) to route employees for the Southern California Gas Company (SOCAL) from Los Angeles, USA.

3.1.2. Cutting Path Determination Problem

The Cutting Path Determination Problem (CPDP) concerns in finding the trajectories for a set of blowtorches to perform a cut pattern on a steel plate within minimum time. Moreira et al. (2007) investigated a CPDP version where the problem was transformed into a dynamic rural postman problem. Rodrigues and Ferreira (2012) modeled another CPDP version as rural postman problem and proposed a heuristic method for the problem. Instances of CPDP can be formulated as OCARP instances by a polynomial transformation algorithm explained in Usberti et al. (2011).

4. Hybrid genetic algorithm

Genetic algorithms are metaheuristics inspired by the theory of evolution, using concepts such as natural selection, reproduction, genetic heritage and mutation (Holland, 1992). Genetic algorithms with local search are called hybrid genetic algorithms. Several routing problems have been successfully addressed by hybrid genetic algorithms, some examples are: VRP (Vehicle Routing Problem) (Prins, 2004), CARP (Capacitated Arc Routing Problem) (Lacomme et al., 2001) and PCARP (Periodic Capacitated Arc Routing Problem) (Lacomme et al., 2005). This paper proposes a hybrid genetic algorithm for the OCARP with the intent to overcome the feasibility issues presented by previous approaches (Usberti, 2012; Usberti et al., 2011).

4.1. Algorithm overview

The proposed HGA for OCARP is composed of the traditional genetic algorithm components (population initialization, selection

and crossover) in addition to the following procedures: (i) feasibilization procedure, (ii) local search, (iii) population restart. The feasibilization procedure is responsible for obtaining a feasible solution from a chromosome. The local search explores the neighborhood of a feasible solution in an attempt to improve the solution. The population restart avoids premature convergence of the population. The HGA does not employ mutation given the difficulty of finding and maintaining feasible OCARP solutions. Algorithm 1 presents the pseudocode.

Algorithm 1: HGA.

Input: $G(V, E)$: instance graph; M: number of vehicles; D: vehicle capacity; **d**: demand vector; P: population size parameter; Q: inter-route local search parameter; W: local search filter parameter; K: restart interval parameter;
Output: *bestSol*: best solution found;
Objective: Search heuristically for a good OCARP solution within time limit.
begin
  SP ← distance_matrix_initialization(G); // Section 4.2
  (**POP**, *bestSol*) ← initial_population(**SP**, M, D, **d**, P, Q); // Section 4.5
  while time limit not exceeded **do**
    parents_selection_breeding(**POP**); // Section 4.6
    for pair of selected parents (*parent*1, *parent*2) **do**
      C ← crossover(*parent*1, *parent*2); // Section 4.7
      sol ← feasibilization(C, **SP**, M, D, **d**); // Section 4.8
      if *sol* is feasible **then**
        (sol', C') ← local_search(*sol*, **SP**, M, D, **d**, Q); // Section 4.9
        New individual added to population;
        if ($cost(sol') < cost(bestSol)$) **then**
          bestSol ← sol';
    POP ← update_population(**POP**, **SP**, M, D, **d**, K, Q); // Section 4.10
  return (*bestSol*);

4.2. Distance matrix initialization

The first step of HGA is to initialize the matrix of distances between required arcs. Let E_R be the set of required edges and A_R be the set of required arcs, where for each required edge $\{i, j\} \in E_R$ there are two corresponding required arcs (i, j), $(j, i) \in A_R$. A matrix SP of dimensions $|A_R| \times |A_R|$ is computed such that each entry $SP[e, f]$ is the shortest path cost from the ending node of arc $e \in A_R$ to the starting node of arc $f \in A_R$. For sparse graphs (where $|E|$ is much less than $|V|^2$) the SP can be computed within $O(|V|^3)$ time and $O(|V|^2)$ space by using the Floyd–Warshall algorithm (Cormen et al., 2001). The SP allows HGA to retrieve the distances between required arcs in $O(1)$ time throughout the optimization process.

4.3. Chromosome encoding

The genetic algorithm uses a non-capacitated route encoding for the chromosome. The chromosome is composed by a sequence of $|E_R|$ required arcs representing a single vehicle route of unlimited capacity that services all required edges $(i, j) \in E_R$ (Fig. 2).

The non-required edges are implicitly inserted in the shortest path of successive pairs of required arcs. An OCARP solution associated to a chromosome is obtained only after the feasibilization procedure (Section 4.8).

This chromosome codification is an example of an *Indirect Solution Representation* (ISR) (Prins et al., 2014) with the benefit that any ISR can be optimally solved to obtain the best OCARP solution associated with it. This will be further discussed in Section 4.8.

4.4. Fitness

The fitness of a solution *sol* is defined as $1/(cost(sol) - LB_0)$, where LB_0 is a trivial lower bound: the sum of cost of all required edges. If any solution *sol* is found where $cost(sol) = LB_0$ then the HGA is halted since an optimal solution was found.

4.5. Initial population

This procedure aims the initialization of P feasible individuals to fill the initial population of the HGA, where P is the population size. The construction of an individual has two steps. The first step constructs a non-capacitated route and the second step splits the non-capacitated route into capacitated sub-routes in an attempt to generate an OCARP solution.

First-step. A non-capacitated route is constructed by a nearest neighbor heuristic. The route is initialized with a random required arc. The route is then augmented by iteratively adding the closest required arc, considering both route endpoints, until all E_R required edges are covered. Tie-breaking rule is to choose one of the closest required arcs at random. The non-capacitated route is then improved through 2-opt local search (Groves and Van Vuuren, 2005) until a local minimum is found. The nearest neighbor heuristic time complexity is $O(|E_R|^2)$ and each 2-opt iteration is $O(|E_R|^2)$. Therefore, the construction of a non-capacitated route is bounded by $O(c|E_R|^2)$, where c is an upper bound for the number of 2-opt iterations. Note that c is not actually employed by HGA but rather used exclusively for the analysis of complexity.

Second-step. The feasibilization procedure (Section 4.8) attempt to split the non-capacitated route into M capacitated sub-routes. If the feasibilization is successful, the solution is inserted in the population. Otherwise, the infeasible individual is discarded.

The whole process is repeated until the population contains P feasible individuals. If initial population procedure is unable to generate P feasible individuals within time limit, the HGA is halted and the best feasible solution (if any) is returned.

4.6. Parents selection for breeding

At each generation, $P - 1$ crossovers are performed and each crossover requires the selection of two parent chromosomes to generate a single offspring.

The selection of the parents is made by the Stochastic Universal Selection (Baker, 1986). In this technique a wheel is created and partitioned in sections, one associated with each individual and that have size proportional to the fitness of the individual. Then a single pointer is spun at random, and all parents are selected by selecting them at evenly spaced intervals starting from the random pointer. This method has the advantage to select the parents in a more predictable behavior than the Roulette Wheel Selection, reducing the bias of the selection operation (Baker, 1986).

4.7. Crossover

The $C1$ crossover operator, as described by Reeves (1996), was adopted. First a point is chosen at random. The sequence of genes from the first parent to the left of the crossover point is copied to the offspring. The remaining genes are copied to the offspring in the same order they appear in the second parent. Fig. 3 gives an example with $E_R = \{(a, b), (c, d), (e, f)(g, h), (i, j), (k, l)\}$.

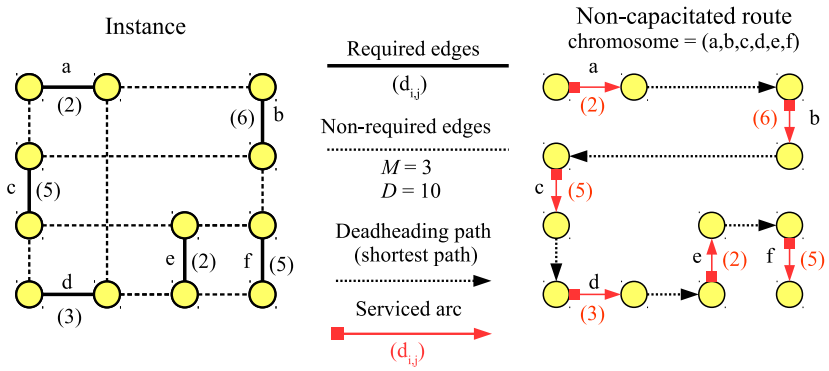

Fig. 2. An instance and a corresponding chromosome.

Parent 1	(A,B)	(C,D)	(E,F)	(G,H)	(I,J)	(K,L)
Parent 2	(L,K)	(J,I)	(A,B)	(D,C)	(G,H)	(F,E)
Child	(A,B)	(C,D)	(L,K)	(J,I)	(G,H)	(F,E)

Fig. 3. *C*1 Crossover operation.

4.8. Feasibilization

The feasibilization procedure starts with a non-capacitated route given by a chromosome C and tries to split it into capacitated sub-routes, using a *Split* algorithm. The *Split* algorithm proposed by Ulusoy (1985) make an optimal decomposition of the non-capacitated route into M capacitated sub-routes, if there is at least one such decomposition. The output of *Split* is a set of M minimum-cost sub-routes, subject to vehicle capacity and that cover all required edges.

Algorithm 2 shows the *Split* algorithm while Fig. 4 exemplifies the process. The *Split* algorithm starts by creating an auxiliary graph H. Fig. 4(i) presents the instance and a chromosome C that is composed by a sequence of required arcs. Fig. 4(ii) shows the cost of shortest path between each successive pair of required arcs. Fig. 4(iii) shows the auxiliary graph H associated to C. The graph H is a directed acyclic graph (DAG) where each vertex represents a required arc from C and each arc represents a capacitated sub-route of C that is feasible with respect to the vehicle capacity. The cost of each arc of H is given by the cost of each corresponding sub-route.

The *Split* algorithm finds the shortest path in H starting from the first vertex, in topological order, to the last vertex composed by at most M arcs. Since graph H is a DAG, the shortest path can be obtained by Algorithm 3 , which is a straightforward adaptation of Bellman-Ford algorithm (Cormen et al., 2001). The OCARP solution is the set of routes associated to the arcs selected in the shortest path of H, as shown in Fig. 4(iv).

If the *Split* algorithm fails to find a shortest path with at most M arcs, then another feasibilization procedure is attempted as explained next. Considering all capacitated sub-routes generated by the *Split* algorithm (arcs of H), a sub-route with the least remainder capacity is inserted into a new partial solution (example in Fig. 5). Tie-breaking rule is to choose one sub-route at random. Then, a non-capacitated route is created containing all required edges still uncovered by the partial solution. This non-capacitated route is constructed by the same process employed in population initialization's first step. Finally, the *Split* algorithm is applied on the new non-capacitated route, this time using one less vehicle. If the algorithm find a feasible solution, the corresponding capacitated sub-routes join the partial solution into a complete OCARP solution. Otherwise, the process is repeated up to M iterations. If the *Split* algorithm does not find a feasible solution within M iterations, the chromosome C is declared infeasible and discarded. The feasibilization procedure pseudocode is shown in Algorithm 4.

The feasibilization procedure goal is to intensify the search for feasible solutions even when the *Split* algorithm alone is not enough to achieve feasible solutions. Additional experiments have

Algorithm 2: Split.

Input: C: chromosome; **SP**: distance matrix; M: number of vehicles; D: vehicle capacity; **d**: demand vector;
Output: *sol*: solution; $H(V_H, A_H)$: auxiliary graph;
Objective: Find the optimal splitting of the non-capacitated route associated to chromosome C.
begin
  // Graph $H(V_H, A_H)$ initialization
  Let C be defined as a sequence $(a_1, \ldots, a_{|C|})$ of required arcs;
  $V_H \leftarrow \emptyset$; $A_H \leftarrow \emptyset$; **cost**[ ] $\leftarrow \emptyset$;
  for $i \leftarrow 1$ *to* $|C|$ **do**
    $V_H \leftarrow V_H \cup \{a_i\}$;
  $V_H \leftarrow V_H \cup \{0\}$; // sink node
  for $i \leftarrow 1$ *to* $|C|$ **do**
    // sub-routes attending exactly one required arc
    sum_demand $\leftarrow$**d**$[a_i]$;
    sum_deadheading $\leftarrow 0$;
    $A_H \leftarrow A_H \cup \{(a_i, a_{i+1})\}$;
    cost$[(a_i, a_{i+1})] \leftarrow 0$;
    for $j \leftarrow i+1$ *to* $|C|$ **do**
      // sub-routes attending more than one required arc
      sum_demand $\leftarrow$ *sum_demand* + **d**$[a_j]$;
      sum_deadheading $\leftarrow$ *sum_deadheading* + **SP**$[a_{j-1}][a_j]$;
      if *sum_demand* $\leq D$ **then**
        $A_H \leftarrow A_H \cup \{(a_i, a_{j+1})\}$;
        cost$[(a_i, a_{j+1})] \leftarrow$ *sum_deadheading*;
  DAG_shortest_path(V_H, A_H, **cost**, M, a_1); // Algorithm 3
  if exists shortest path with at most M arcs **then**
    sol $\leftarrow$ OCARP solution associated to the found shortest path;
    return (*sol*, $\emptyset$);
  else
    return ($\emptyset$, H);

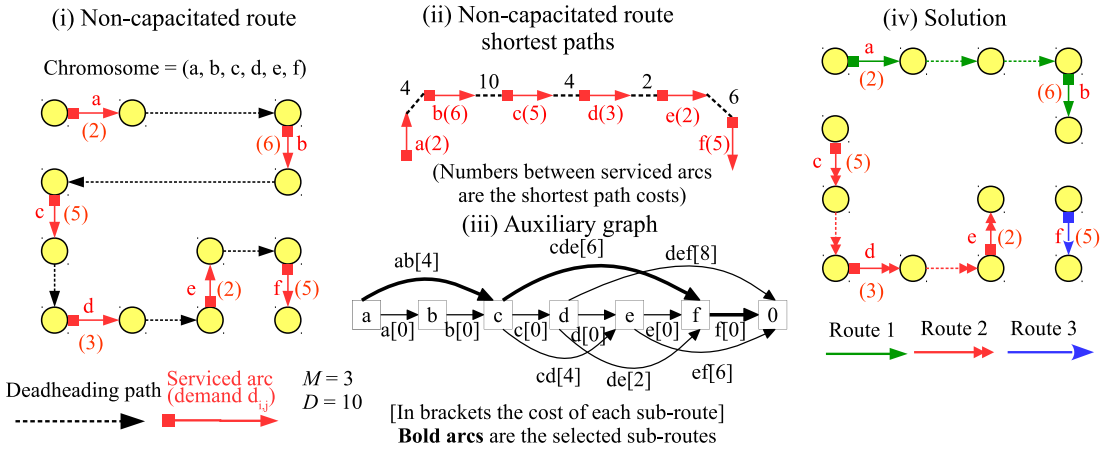

Fig. 4. Operation of *Split* algorithm.

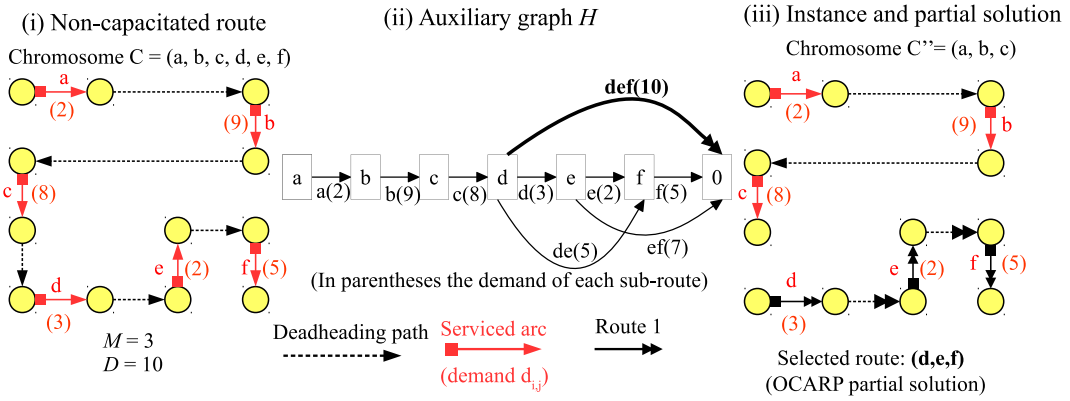

Fig. 5. Feasibilization procedure operation: strategy to improve feasibility.

Algorithm 3: DAG shortest path with at most M arcs.

Input: $H(V_H, A_H)$: directed acyclic graph; **cost**: cost vector for arcs $\in A_H$; M: number of vehicles; v_1: first vertex in topological order of $H(V_H, A_H)$;
Output: **dst**: distances; **pred**: predecessors;
Objective: Find shortest path in H from v_1 to other vertices using at most M arcs.

```
begin
    Initialize dst[v] ← ∞ and pred[v] ← NIL for each v ∈ V_H.
    Initialize dst[v_1] ← 0;
    for k ← 1 to M do
        for v_i ∈ V_H in reverse topological order do
            for (v_i, v_j) ∈ A_H do
                if dst[v_i]+cost[(v_i, v_j)] <dst[v_j] then
                    dst[v_j] ←dst[v_i]+cost[(v_i, v_j)];
                    pred[v_j] ← v_i;
```

shown that the *Split* algorithm alone has a high ratio of failure for large instances with very tight number of vehicles (Section 5.1).

A complexity analysis of Algorithms 2–4 is described below. Algorithm 3 runs in $O(M|E_R|^2)$ since $|A_H|$ is bounded by $|E_R|^2$. The *Split* algorithm (Algorithm 2) first creates the auxiliary graph H in $O(|E_R|^2)$ and then calls Algorithm 3. Thus Algorithm 2 has complexity $O(M|E_R|^2)$. The feasibilization procedure (Algorithm 4) main loop is executed M times in the worst case (in practice it can be much less if the instance is not tight on the number of vehicles). At each iteration, it executes the *Split* algorithm and may call the creation of a new partial solution through the population initialization's first step (Section 4.5). The *Split* algorithm is $O(M|E_R|^2)$, while the population initialization's first step is $O(c|E_R|^2)$, where c is an upper bound of 2-opt local search iterations. Therefore, the whole feasibilization procedure worst case complexity is bounded by $O(M(M+c)|E_R|^2)$.

4.9. Local search

Local search algorithms search for better solutions that are similar (neighboring) to the current solution. Local search is often the most computational expensive routine of metaheuristics, therefore it seems wise to apply local search only to solutions that are promising. This work uses a statistical filter for local search, proposed by Usberti et al. (2013), able to classify bad solutions within a certain confidence interval. A variable representing the improvement ratio between initial and local search solution costs is considered. The improvement ratio of a solution s is defined as $improvement(s) = cost(s)/cost(s')$, where s' is the local minimum solution obtained through s after local search. The filter starts by applying local search to the first W solutions $s_1, s_2, \ldots, s_W$ and storing a sampling of $improvement(s_i)$, $\forall i \in \{1, 2, \ldots, W\}$. From the sampling it is computed μ as average and ρ as standard deviation. Then this information is used to decide whether a solution is allowed to local search or not. A solution s is considered good enough for local search if it satisfies the following condition: $cost(s)/cost(s^*) \leq (\mu + 2\rho)$, where s^* is the incumbent solution. The filter gives a confidence interval of approximately 95% probability to reject solutions that cannot be improved by

Algorithm 4: Feasibilization procedure.

Input: C: chromosome; **SP**: distance matrix; M: number of vehicles;D: vehicle capacity; **d**: demand vector;
Output: sol: solution;
Objective: Generates an OCARP solution based on a non-capacitated route defined by C, using the Split algorithm and a feasibilization strategy.
begin
  $C' \leftarrow C$; $sol \leftarrow \emptyset$; $found \leftarrow$ **false**;
  for $k \leftarrow 0$ *to* $M-1$ **do**
    $M' \leftarrow M - k$;
    $(sol', H) \leftarrow Split(C', \mathbf{SP}, M', D, \mathbf{d})$; // Algorithm 2
    if sol' is feasible **then**
      $sol \leftarrow sol \cup sol'$;
      $found \leftarrow$ **true**;
      $k \leftarrow M$; // Feasible solution found
    else
      Let $R_{\max}$ be a sub-route from H that services most demand;
      $sol \leftarrow sol \cup R_{\max}$; // *Route $R_{\max}$ is added to partial solution.*
      Create a non-capacitated route C'' using population initialization first step servicing only the required edges that are not serviced in partial solution sol;
      $C' \leftarrow C''$;
  if $found$ is true **then**
    return sol; // Feasibilization success
  else
    return $\emptyset$; // Feasibilization failed

local search to overcome the incumbent solution s^*, assuming that $improvement(s)$ is an independent random variable with normal distribution.

The local search pseudocode is shown by Algorithm 5 . The

Algorithm 5: Local search.

Input: sol: initial solution; **SP**: distance matrix; M: number of vehicles; D: vehicle capacity; **d**: demand vector; Q: local search set size parameter;
Output: sol: local search solution; C: chromosome of local search solution;
Objective: Improve an OCARP feasible solution.
begin
  $intraCost \leftarrow \infty$;
  while cost(sol) $<$ $intraCost$ **do**
    $sol \leftarrow$ inter-route_local_search(sol, **SP**, M, D, **d**, Q); // Algorithm 6
    $sol \leftarrow$ intra-route_local_search(sol, **SP**, M);
    $intraCost \leftarrow$ cost(sol);
    $C \leftarrow$ Split^{-1}(sol, **SP**, M); // Algorithm 7
    $sol \leftarrow$ Split(C, **SP**, M, D, **d**); // Algorithm 2
  return (sol, C);

local search is composed by the following procedures: (i) the inter-route local search that reconstructs pair of routes, (ii) the intra-route local search that executes 2-opt for each route, and (iii) the *Split* local search that first executes the *Split*$^{-1}$ algorithm to make a new chromosome for the current solution and then executes *Split* algorithm which explores the solutions neighborhood by optimally splitting the chromosome.

Inter-route. The inter-route local search uses a deconstruct/reconstruct approach applied to pairs of nearby routes (Algorithm 6). Consider a set $S_{(u,\,v)}$ composed by the Q nearest

Algorithm 6: Inter-route local search.

Input: sol: initial solution; **SP**: distance matrix; M: number of vehicles; D: vehicle capacity; **d**: demand vector; Q: set size parameter;
Output: sol: local search solution;
Objective: Improve an OCARP feasible solution by reconstructing pair of routes;
begin
  Let sol be an array of routes (r_1, r_2, ... , $r_{|M|}$);
  for $(u, v) \in E_R$ **do**
    Let $\mathbf{dist}_{(\mathbf{u},\mathbf{v})}[1..M]$ be a vector with M empty positions;
    for $i \leftarrow 1$ *to* M **do**
      $distance \leftarrow +\infty$;
      for each required arc $(a, b) \neq (u, v)$ in the route r_i of solution sol **do**
        $distance \leftarrow$ min{$distance$, **SP**$[(u, v)][(a, b)]$, **SP**$[(u, v)][(b, a)]$, **SP**$[(v, u)][(a, b)]$, **SP**$[(v, u)][(b, a)]$};
      $\mathbf{dist}_{(\mathbf{u},\mathbf{v})}[i] \leftarrow (distance, i)$;
    Sort vector $\mathbf{dist}_{(\mathbf{u},\mathbf{v})}[1..M]$ by the distances in non-decreasing order;
    Let $S_{(u,v)}$ be the set of the first Q routes of sol in the order defined by $\mathbf{dist}_{(\mathbf{u},\mathbf{v})}[1..M]$;
    for each pair of routes r_i and r_j in $S_{(u,v)}$ **do**
      $ncr\_pair \leftarrow$ Create a non-capacitated route using initial population first step servicing the required edges in r_i and r_j;
      $sol\_pair \leftarrow$ feasibilization(ncr_pair, **SP**, 2, D, **d**); // Algorithm 4
      if routes of sol_pair have lower cost than r_i and r_j added **then**
        Replace routes r_i and r_j in sol by the routes of sol_pair;
  return sol;

routes to the required edge $(u,\ v) \in E_R$. Each pair of routes from set $S_{(u,\,v)}$ is submitted to the reconstruction phase, which starts by creating a new pair of routes servicing the same required edges. The new pair of routes is created using the same process employed by population initialization but servicing only the required edges from the original pair and using two vehicles. The new pair of routes replaces the original pair if a lower cost is achieved. The inter-route local search is applied for each set $S_{(u,\,v)}$, where $(u,\ v) \in E_R$. An example of the inter-route local search is presented in Fig. 6, where the set of routes $S_a = \{r_1, r_3, r_4\}$ is reconstructed.

Intra-route. The intra-route local search consists of the 2-opt local search applied to each route individually. Each 2-opt local search is executed until a local minimum is achieved. Considering that each route is a sequence of required arcs linked by deadheading shortest paths, the 2-opt operator consists of removing two shortest path links and replacing them by two other links with lower costs and that maintains the route integrity (Groves and Van Vuuren, 2005).

Split local search. The *Split*$^{-1}$ takes a solution s and generates an associated chromosome. For this, the *Split*$^{-1}$ (Algorithm 7) operates on an undirected graph $I(V,\ E \cup E_s)$ where each edge $(u,\ v) \in E_s$ represents one route of s, where u is the starting vertex of the first

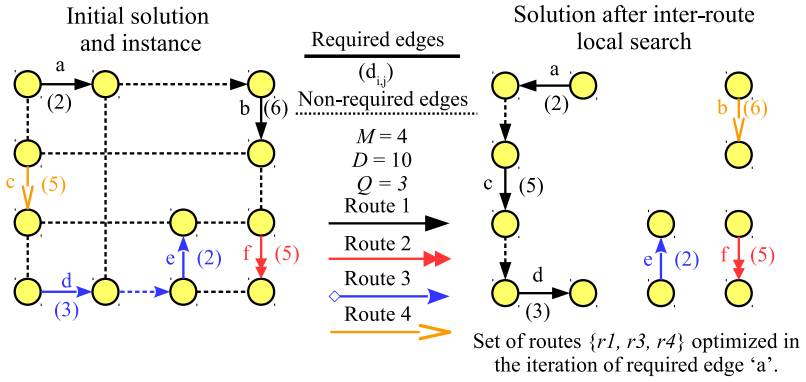

Fig. 6. Example of inter-route local search optimization.

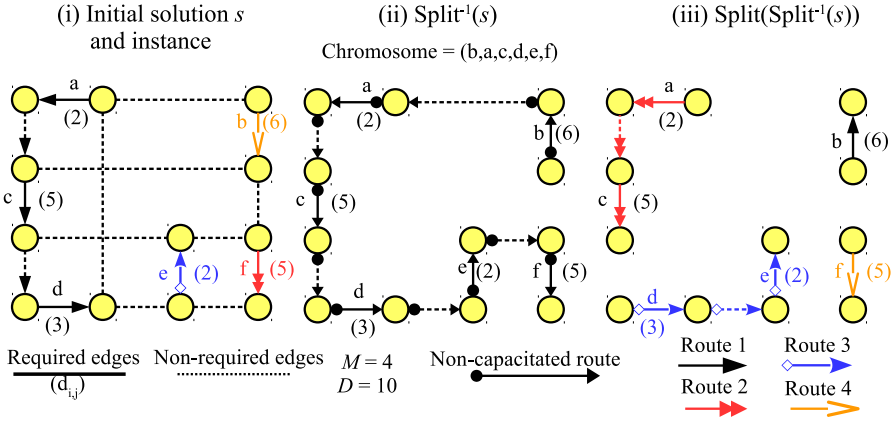

Fig. 7. Example of $Split^{-1}$ followed by *Split* execution.

Algorithm 7: $Split^{-1}$.

Input: s: solution; **SP**: distance matrix; M: number of vehicles;
Output: C: chromosome representing s;
Objective: Create a chromosome representing an OCARP solution.
begin
  Initialize graph $I(V, E \cup E_s)$;
  $E_s \leftarrow \emptyset$;
  for $i \leftarrow 1$ *to* M **do**
    Add to E_s the edge (u, v) where u is the starting vertex of the first required arc of the ith route of s and v the ending vertex of the last required arc of the same route;
  Create a non-capacitated route $ncr(s)$ on I using the first step method from population initialization servicing the edges of E_s;
  Create the chromosome (non-capacitated route) C on G associated to $ncr(s)$ by replacing each edge of E_s by its corresponding route of s;
  return C;

required arc of that route and v is the ending vertex of the last required arc of the same route. On graph I a single non-capacitated route servicing the edges of E_s is constructed applying the first step of population initialization. The resulting non-capacitated route on graph I is transformed into a non-capacitated route on original graph G by replacing each edge of E_s by its corresponding route in s. The new non-capacitated route is then optimally split by the *Split* algorithm. It is worth noticing that the solution obtained by $Split(Split^{-1}(s))$ in this case is guaranteed to be feasible and to have a cost at least as good as s since the routes of s is a viable solution for the *Split* algorithm.

A complexity analysis of the local search procedure is provided. Each iteration of inter-route main loop creates and sorts a set $S_{(u,\,v)}$ in $O(|E_R| + MlogM)$, followed by the creation of $O(Q^2)$ new pairs of routes in the reconstruction phase (where Q is a fixed parameter). Each pair of routes is created and feasibilized in $O(c|E_R|^2)$, considering that c is an upper bound for 2-opt iterations. The complexity of each inter-route iteration is $O(|E_R| + MlogM + c|E_R|^2)$. A non-trivial OCARP problem has $M < |E_R|$, so the sum can be expressed simply as $O(c|E_R|^2)$. Therefore, the inter-route local search (Algorithm 6)

Table 1
Benchmark instances for OCARP.

Instance set	$\|V\|$	$\|E\|$	$\|E_R\|$	M^*
gdb	7–27	11–55	11–55	3–10
val	24–50	34–97	34–97	2–10
egl	77–140	98–190	51–190	5–35
A	10–40	15–69	11–63	1–25
B	13–40	15–69	11–53	2–21

Table 2
Hybrid genetic algorithm parameters.

Parameter	Description	Value
P	Population size	20
K	Population restart interval (generations)	150
W	Local search filter sampling size	100
Q	Inter-route local search set size	4

Table 3
Computational experiment overall results.

group	class	*Gap*(%)			*Feas*(%)			*FeasDiff*(%)			*CPU*(*s*)		
		RPS	GRASP	HGA	RPS	GRASP	HGA	RPS	GRASP	HGA	RPS	GRASP	HGA
gdb	M^*	0.09	**0.07**	**0.07**	77.45	85.50	98.55	77.38	78.50	97.38	172.73	0.3	< 0.1
	M^*+1	**0.05**	**0.05**	**0.05**	98.47	98.91	100.00	98.47	96.44	99.69	11.5	0.1	< 0.1
	M^*+2	0.05	**0.02**	**0.02**	99.95	100.00	100.00	99.95	97.73	99.65	14.5	0.1	< 0.1
	overall	0.06	**0.05**	**0.05**	91.96	94.81	99.52	91.93	90.89	98.91	66.2	0.2	< 0.1
val	M^*	3.83	3.19	**3.18**	83.81	87.58	99.22	83.73	22.47	87.66	658.4	170.1	32.1
	M^*+1	3.29	**2.24**	**2.24**	97.96	99.01	100.00	97.95	46.93	93.64	818.5	32.1	3.0
	M^*+2	3.30	**1.46**	**1.46**	99.70	100.00	100.00	99.69	46.82	93.39	791.4	13.2	1.1
	overall	3.48	**2.29**	**2.29**	93.82	95.53	99.74	93.79	38.74	91.56	756.1	71.8	12.0
egl	M^*	31.74	15.94	**11.83**	49.22	60.14	95.26	49.13	3.07	79.04	2066.4	1142.5	1424.0
	M^*+1	20.69	7.20	**6.75**	81.10	87.69	99.99	80.87	12.95	77.80	1844.4	923.6	951.4
	M^*+2	18.49	5.33	**5.19**	91.07	94.48	100.00	90.90	18.27	76.06	2057.9	1037.3	528.8
	overall	23.64	9.49	**7.92**	73.79	80.77	98.41	73.63	11.43	77.63	1989.6	1034.5	968.1
A	M^*	4.90	**3.54**	3.58	65.07	69.35	94.81	58.51	30.51	64.50	492.87	35.10	95.44
	M^*+1	3.20	**1.72**	**1.72**	84.48	87.47	97.66	78.72	60.64	86.76	439.74	16.16	23.30
	M^*+2	2.98	**1.14**	**1.14**	90.55	94.05	98.85	84.83	76.69	93.50	410.49	0.28	1.63
	overall	3.69	**2.13**	2.14	80.04	83.62	97.11	74.02	55.95	81.58	447.7	17.2	40.1
B	M^*	3.66	**2.91**	2.94	55.60	62.61	89.00	50.58	26.43	40.79	479.06	94.94	92.56
	M^*+1	1.34	**0.73**	**0.73**	79.74	87.74	97.70	76.77	63.66	75.85	316.74	0.73	1.33
	M^*+2	1.08	**0.23**	**0.23**	88.32	94.40	99.45	85.35	78.18	84.99	268.49	0.26	1.98
	overall	2.03	**1.29**	1.30	74.56	81.59	95.38	70.90	56.09	67.21	354.8	32.0	32.0
overall	M^*	8.31	4.93	**4.22**	67.14	73.60	95.56	64.66	30.54	73.97	753.44	267.25	295.95
	M^*+1	5.43	2.36	**2.27**	88.67	92.29	99.04	86.75	55.09	87.06	268.49	0.26	1.98
	M^*+2	4.95	1.61	**1.58**	94.06	96.64	99.63	92.16	62.57	89.88	702.28	185.12	93.63
	overall	6.23	2.96	**2.69**	83.29	87.51	98.08	81.19	49.40	83.64	714.0	208.7	187.6

In **bold**: best results regarding *Gap*(%).
Gap(%): average deviation from lower bound (%).
Feas(%): average ratio of feasible solutions (%).
FeasDiff(%): average ratio of feasible and different solutions (%).
CPU(*s*): average time for the heuristic to attain its best solution (in seconds).

complexity, including all $|E_R|$ main loop iterations, is $O(c|E_R|^3)$. The intra-route consists of the 2-opt local search operations on each route. Since there are $|E_R|$ required arcs distributed among the routes, the local search second step is bounded by $O(c|E_R|^2)$. The $Split^{-1}$ (Algorithm 7) is bounded by $O(cM^2)$ and the *Split* algorithm is $O(M|E_R|^2)$. The complexity of the entire local search procedure (Algorithm 5) is bounded by $O(c|E_R|^3)$ for each iteration.

4.10. Population update

The new population is updated for the next generation by including the best individual and the offspring. For each infeasible offspring, an individual from the current population is chosen at random to be inserted into the new population. Therefore, the population is always composed of P feasible individuals.

To maintain diversity, the HGA employs a population restart procedure executed every K generations. This procedure was necessary to avoid the population premature convergence, specially since HGA does not apply mutation. At each restart half of the population is replaced by new solutions. The discarded individuals are chosen at random, but the best solution is never selected. The new individuals are created by the same method employed by population initialization.

5. Computational experiments

The computational experiments were conducted on a benchmark of CARP instances, which includes 23 *gdb* (Golden et al., 1982), 34 *val* (Benavent et al., 1991), 24 *egl* (Li and Eglese, 1996), 32 *A* (Hertz, 2005), and 24 *B* (Hertz, 2005) instances, totaling 137 instances. The depot was considered a common node while the rest of the data left intact. Full experimental data, instances and HGA source code are available on-line.[1]

The computational tests considered three classes of instances regarding the number of vehicles: $M = M^*$, $M = M^* + 1$ and $M = M^* + 2$, where M^* is the minimum number of vehicles required for a feasible solution. Consequently, 137 CARP instances and three different numbers of vehicles summed up 411 OCARP instances. Table 1 summarizes the data for the OCARP instances.

The experiment compared the results obtained by the HGA (Hybrid Genetic Algorithm) with two methods from literature: RPS (Reactive Path-scanning with ellipse rule) heuristic (Usberti et al., 2011) and GRASP (Greedy Randomized Adaptive Search Procedure with path-relinking) metaheuristic (Usberti, 2012).

The RPS, GRASP and HGA were executed in an Intel Xeon X3430 2.4 GHz with 8GB of RAM and Linux 64-bit operating system. HGA was implemented in C++ and uses the LEMON (Dezs et al., 2011) library for graph algorithms. The RPS and GRASP were implemented in C and its source codes were provided by the authors. The parameters of HGA are given by Table 2, these values were selected after previous empirical tests.

Table 3 contains the overall results for each group-class of instances. The $Gap(\%) = 100 * (UB - LB)/LB$ is the average deviation from lower bound, where upper bounds (UB) are provided by the heuristics and the lower bounds (LB) were reported by Usberti et al. (2012). The column *Feas*(%) measures the percentage ratio of feasible solutions, while *FeasDiff*(%) measures the percentage of feasible and different solutions. Both ratios were calculated for the set of solutions obtained until the time limit or until one million solutions were generated. The field *CPU*(*s*) shows the average processing time to attain the best solution (in seconds). All instances were processed for one hour by each heuristic.

Table 3 shows that HGA obtained significantly better overall solutions than RPS and GRASP with less processing time. The experiment also confirmed that the value of the parameter M has meaningful impact on results, since all algorithms have a better performance as the value of M is increased from M^* to $M^* + 1$ and $M^* + 2$.

[1] http://www.ic.unicamp.br/~fusberti/problems/ocarp.

Table 4
Detailed results for *egl* with $M = M^*$ instances.

instance	$\|E_R\|$	M^*	*LB*	*UB*			*Gap*(%)			*FeasDiff*(%)		
				RPS	GRASP	HGA	RPS	GRASP	HGA	RPS	GRASP	HGA
egl-e1-A	51	5	1673	1802	**1775**	**1775**	7.71	6.10	6.10	66.61	0.76	69.11
egl-e1-B	51	7	1591	1823	**1749**	**1749**	14.58	9.93	9.93	59.16	0.35	81.10
egl-e1-C	51	10	1523	1723	**1652**	**1652**	13.13	8.47	8.47	66.63	0.16	67.51
egl-e2-A	72	7	2019	2302	**2173**	**2173**	14.02	7.63	7.63	63.44	1.01	90.75
egl-e2-B	72	10	1944	2249	2079	**2062**	15.69	6.94	6.07	64.46	0.88	84.56
egl-e2-C	72	14	1900	2424	**2084**	**2084**	27.58	9.68	9.68	39.25	0.16	74.92
egl-e3-A	87	8	2277	2856	2541	**2533**	25.43	11.59	11.24	52.36	2.90	95.92
egl-e3-B	87	12	2221	2759	2438	**2409**	24.22	9.77	8.46	58.88	2.40	93.53
egl-e3-C	87	17	2188	2804	2362	**2357**	28.15	7.95	7.72	47.55	0.90	81.14
egl-e4-A	98	9	2453	3095	2656	**2631**	26.17	8.28	7.26	52.51	2.78	94.61
egl-e4-B	98	14	2453	3198	2720	**2708**	30.37	10.88	10.40	46.43	1.04	91.50
egl-e4-C	98	*19*	*2453*	*3789*	*3208*	***2812***	*54.46*	*30.78*	*14.64*	*0.14*	*0.04*	*22.48*
egl-s1-A	75	7	1584	1942	**1799**	**1799**	22.60	13.57	13.57	77.55	1.64	69.45
egl-s1-B	75	10	1475	1859	**1745**	**1745**	26.03	18.31	18.31	80.89	1.79	60.97
egl-s1-C	75	14	1415	2080	1891	**1874**	47.00	33.64	32.44	42.67	0.10	84.74
egl-s2-A	147	14	3228	4303	**3693**	**3693**	33.30	14.41	14.41	68.29	8.45	98.07
egl-s2-B	147	*20*	*3176*	*4979*	*4360*	***3775***	*56.77*	*37.28*	*18.86*	*3.65*	*0.55*	*87.78*
egl-s2-C	147	*27*	*3174*	*4812*	*3856*	***3691***	*51.61*	*21.49*	*16.29*	*7.73*	*0.78*	*84.06*
egl-s3-A	159	15	3393	4298	3824	**3808**	26.67	12.70	12.23	82.65	18.39	97.06
egl-s3-B	159	22	3379	4403	3742	**3677**	30.30	10.74	8.82	55.03	7.25	94.47
egl-s3-C	159	29	3379	4782	3758	**3733**	41.52	11.22	10.48	22.45	3.65	79.15
egl-s4-A	190	19	4186	5174	4522	**4486**	23.60	8.03	7.17	69.91	12.97	97.16
egl-s4-B	190	27	4186	5403	4497	**4393**	29.07	7.43	4.95	50.80	4.74	88.10
egl-s4-C	190	*35*	*4186*	*8023*	*6937*	***4971***	*91.66*	*65.72*	*18.75*	*0.01*	*0.01*	*8.70*

In **bold**: best results regarding *UB*.
In *italics*: instances that HGA produced a major *Gap*(%) improvement (> 5%).
LB: lower bound reported by Usberti et al. (2012). *UB*: best solution cost;
Gap(%): deviation from lower bound (%).
FeasDiff(%): ratio of feasible and different solutions (%).

Table 5
Feasibilization procedure experiment results.

group	class	*n_infeasible*		*Feas*(%)		*FeasDiff*(%)	
		HGA′	HGA	HGA′	HGA	HGA′	HGA
gdb	M^*	**0**	**0**	71.04	98.55	70.69	97.38
	M^*+1	**0**	**0**	96.71	99.01	96.40	96.44
	M^*+2	**0**	**0**	100.00	100.00	99.65	99.65
	overall	**0**	**0**	89.25	99.52	88.91	98.91
val	M^*	**0**	**0**	88.40	99.22	75.88	87.66
	M^*+1	**0**	**0**	99.02	100.00	92.17	93.64
	M^*+2	**0**	**0**	99.02	100.00	93.61	93.39
	overall	**0**	**0**	95.81	99.74	87.22	91.56
egl	M^*	7	**0**	34.55	95.26	18.70	79.04
	M^*+1	1	**0**	73.24	99.99	52.45	77.80
	M^*+2	0	**0**	89.77	100.00	66.00	76.06
	overall	8	**0**	65.85	98.41	45.71	77.63
A	M^*	4	**0**	68.93	94.81	45.40	64.50
	M^*+1	1	**0**	84.71	97.66	74.24	86.76
	M^*+2	1	**0**	91.22	98.85	88.07	93.50
	overall	6	**0**	81.62	97.11	69.24	81.58
B	M^*	6	**0**	62.87	89.00	28.03	40.79
	M^*+1	1	**0**	81.41	97.70	66.06	75.85
	M^*+2	0	**0**	91.23	99.45	81.20	84.99
	overall	7	**0**	78.50	95.38	58.43	67.21
overall	M^*	17	**0**	66.77	94.67	49.12	73.97
	M^*+1	3	**0**	87.60	99.04	76.99	87.06
	M^*+2	1	**0**	94.58	99.63	86.22	89.88
	overall	21	**0**	82.98	97.78	70.78	83.64

n_infeasible: number of instances with no feasible solutions.
Feas(%): average ratio of feasible solutions.
FeasDiff(%): average ratio of feasible and different solutions.

The results in Table 3 for *gdb* and *val* report that HGA can attain equal or better solutions with less processing time for small instances. The results with *egl* show the superior performance of HGA for larger instances. Especially for *egl* instances with $M = M^*$ class, which are the hardest instances, the deviation from lower bound *Gap*(%) was substantially reduced from 15.94% (GRASP) to 11.83% (HGA). For *A* and *B* instances the HGA shows a slightly worse but competitive performance in comparison to GRASP. The overall *Gap*(%) show that HGA performed better than GRASP and RPS.

From Table 3 it is worth noticing the difference between *Feas*(%) and *FeasDiff*(%) for each heuristic. The RPS has very similar *Feas*(%) and *FeasDiff*(%), explained by the fact that RPS does not employ local search which could in turn reduce solution variability. On

the other hand, GRASP relies heavily on local search methods and therefore has *FeasDiff*(%) substantially lower than *Feas*(%). In overall, HGA has *FeasDiff*(%) higher than GRASP for all instances and is very competitive with RPS, where the HGA is better for $M = M^*$ and $M = M^* + 1$ while RPS is better for $M = M^* + 2$.

Table 4 presents the detailed results for the *egl* instances and class $M = M^*$. The number of required edges ($|E_R|$), number of vehicles (M^*), lower bound (*LB*), upper bound (*UB*), deviation from lower bound *Gap*(%) and the ratio of unique feasible solutions *FeasDiff*(%). In bold are the best upper bounds and in italics the instances for which HGA produced a major *Gap*(%) improvement ($> 5\%$).

Table 4 shows that HGA obtained the best upper bounds for all instances and improved the best known upper bounds for 16 instances. In special, HGA produced a major *Gap*(%) improvement over the highlighted (in italic) instances. The highlighted instances are also the most difficult ones regarding feasibility. Instance *egl-s4-C*, for example, has *UB* reduced from 6937 (GRASP) to 4971 (HGA), while *FeasDiff*(%) was increased from 0.01 (GRASP) to 8.70 (HGA). In conclusion, the improvement can be credited to HGA effectiveness of generating feasible solutions with reasonably variability when the number of vehicles is tight.

5.1. Feasibilization procedure evaluation

This section aims to evaluate the importance of the feasibilization procedure (Section 4.8) for the HGA performance. Let HGA′ be the HGA where the feasibilization procedure (Algorithm 4) is replaced by a single call of the *Split* algorithm (Algorithm 2) and, if *Split* fails, the solution is discarded. An experiment comparing HGA and HGA′ was executed using the same experimental settings in Section 5.

In Table 5 *n_infeasible* is the number of instances for which no feasible solution was found within the time limit. Column *Feas*(%) measures the percentage ratio of feasible solutions, while *FeasDiff*(%) measures the percentage ratio of feasible and different solutions. Both ratios were calculated for the set of solutions obtained until the time limit or until one million solutions were generated.

The results in Table 5 show that HGA′ could not find feasible solutions for 17 instances with class $M = M^*$ and 4 instances for classes $M = M^* + 1$ and $M = M^* + 2$ combined. Table 5 shows that *Feas*(%) and *FeasDiff*(%) from HGA′ are substantially lower than the ratios from HGA, especially for *egl* instances with $M = M^*$ where a difference of *FeasDiff*(%) from 18.70% to 79.04% is reported. On the other hand, the *FeasDiff*(%) difference for the *egl* instances with $M = M^* + 2$ varies only from 66.00% to 76.06%.

As conclusion the feasibilization procedure is a key feature to make HGA competitive on instances with very tight number of vehicles. The strategy employed by feasibilization procedure is generic and its potential to be applied on more complex routing problems with either (i) small fixed number of vehicles, or (ii) very onerous vehicle cost, is an interesting research topic.

6. Conclusion

This paper proposed a hybrid genetic algorithm (HGA) for the open capacitated arc routing problem. The HGA is composed by the genetic algorithm combined with local search and feasibilization procedures. The key features of HGA are: (i) solutions are represented by a non-capacitated route; (ii) *Split* algorithm, which optimally splits a non-capacitated route into feasible routes; (iii) feasibilization procedure to tackle instances with a tight number of vehicles; (iv) inter-route local search with a deconstruct/reconstruct approach; (v) statistical filtering of solutions for local search.

The computational experiments, using a set of benchmark instances from literature, has shown that HGA outperformed state-of-the-art methods from literature for almost all instances. A major difficulty reported by previous approaches was the obtainability of feasible solutions for instances with very tight number of vehicles. The HGA performance depended on the feasibilization procedure to overcome the feasibility issue. Future research should focus on assessing the effectiveness of HGA methods if applied to more complex routing problems. The design of better lower bounding procedures in order to reduce the optimality gaps presented is also an interesting field of research.

Acknowledgment

This work was supported by grants #2016/00315-0, São Paulo Research Foundation (FAPESP) and 307472/2015-9, National Counsel of Technological and Scientific Development (CNPq). We also thank to two anonymous referees for their valuable comments.

References

Baker, E.J., 1986. Reducing bias and inefficiency in the selection algorithm. In: Proceedings of the Second International Conference on Genetic Algorithms, pp. 14–21.
Benavent, E., Campos, V., Corberán, A., Mota, E., 1991. The capacitated arc routing problem: lower bounds. Networks 22, 669–690.
Bodin, L., Levy, L., 1991. The arc partitioning problem. Eur. J. Oper. Res. 53, 393–401.
Corberán, Á., Laporte, G., 2014. Arc Routing: Problems, Methods, and Applications. SIAM, Philadelphia, PA, USA.
Corberán, A., Prins, C., 2010. Recent results on arc routing problems: an annotated bibliography. Networks 56 (1), 50–69.
Cormen, T.H., Stein, C., Rivest, R.L., Leiserson, C.E., 2001. Introduction to Algorithms, 2nd MIT Press, Cambridge, MA.
Dezs, B., Jüttner, A., Kovács, P., 2011. Lemon - an open source C++ graph template library. Electron. Notes Theor. Comput. Sci. 264 (5), 23–45.
Dror, M., 2012. Arc Routing: Theory, Solutions and Applications. Springer Science & Business Media, New York, NY, USA.
Eiselt, H.A., Gendreau, M., Laporte, G., 1995. Arc routing problems, part I: the Chinese postman problem. Oper. Res. 43 (2), 231–242.
Fung, R.Y., Liu, R., Jiang, Z., 2013. A memetic algorithm for the open capacitated arc routing problem. Transp. Res. Part E 50, 53–67.
Golden, B.L., Wong, R.T., 1981. Capacitated arc routing problems. Networks 11, 305–315.
Golden, L.B., DeArmon, S.J., Baker, K.E., 1982. Computational experiments with algorithms for a class of routing problems. Comput. Oper. Res. 10, 47–59.
Groves, G., Van Vuuren, J., 2005. Efficient heuristics for the rural postman problem. ORiON 21 (1), 33–51.
Hertz, A., 2005. Recent trends in arc routing. In: Sharda, R., Voß, S., Golumbic, M.C., Hartman, I.B.A. (Eds.), Graph Theory, Combinatorics and Algorithms. Springer US.
Holland, J.H., 1992. Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence. MIT Press, Cambridge, MA, USA.
Lacomme, P., Prins, C., Ramdane-Chérif, W., 2001. A genetic algorithm for the capacitated arc routing problem and its extensions. In: Applications of Evolutionary Computing. Springer, pp. 473–483.
Lacomme, P., Prins, C., Ramdane-Chérif, W., 2005. Evolutionary algorithms for periodic arc routing problems. Eur. J. Oper. Res. 165 (2), 535–553.
Li, Y.L., Eglese, W.R., 1996. An interactive algorithm for vehicle routeing for winter-gritting. J. Oper. Res. Soc. 47 (2), 217–228 https://link.springer.com/article/10.1057/jors.1996.20.
Moreira, L.M., Oliveira, J.F., Gomes, A.M., Ferreira, J.S., 2007. Heuristics for a dynamic rural postman problem. Comput. Oper. Res. 34, 3281–3294.
Muyldermans, L., Pang, G., 2014. Variants of the capacitated arc routing problem. In: Corberán, Á., Laporte, G. (Eds.), Arc Routing: Problems, Methods, and Applications. SIAM, chapter 10, pp. 223–253.
Prins, C., 2004. A simple and effective evolutionary algorithm for the vehicle routing problem. Comput. Oper. Res. 31 (12), 1985–2002.
Prins, C., 2014. The capacitated arc routing problem: Heuristics. In: Corberán, Á., Laporte, G. (Eds.), Arc Routing: Problems, Methods, and Applications. SIAM, chapter 7, pp. 131–157.
Prins, C., Lacomme, P., Prodhon, C., 2014. Order-first split-second methods for vehicle routing problems: a review. Transp. Res. Part C 40, 179–200.
Reeves, R.C., 1996. Feature article-genetic algorithms for the operations researcher. INFORMS J. Comput. 9, 231–250.
Rodrigues, A.M., Ferreira, J.S., 2012. Cutting path as a rural postman problem: solutions by memetic algorithms. Int. J. Comb. Optim. Probl. Inform. 3 (1), 31–46.
Stern, H.I., Dror, M., 1979. Routing electric meter readers. Comput. Oper. Res. 6, 209–223.
Ulusoy, G., 1985. The fleet size and mix problem for capacitated arc routing. Eur. J. Oper. Res. 22 (3), 329–337.

Usberti, F.L., 2012. Heuristic and Exact Approaches for the Open Capacitated Arc Routing Problem. Ph.D. thesis. University of Campinas.
Usberti, F.L., França, P.M., França, A.L.M., 2012. Branch-and-bound algorithm for an arc routing problem. Annals XLIV SBPO, Rio de Janeiro.
Usberti, F.L., França, P.M., França, A.L.M., 2011. The open capacitated arc routing problem. Comput. Oper. Res. 38 (11), 1543–1555.
Usberti, F.L., França, P.M., França, A.L.M., 2013. Grasp with evolutionary path-relinking for the capacitated arc routing problem. Comput. Oper. Res. 40 (12), 3206–3217.
Wunderlich, J., Collette, M., Levy, L., Bodin, L., 1992. Scheduling meter readers for Southern California Gas Company. Interfaces 22, 22–30.