A Matheuristic for the Distance-Constrained Close-Enough Arc Routing Problem

Ángel Corberán
Dept. d'Estadística i Investigació Operativa, Universitat de València, Spain

Isaac Plana
Dept. de Matemáticas para la Economía y la Empresa, Universitat de València, Spain

Miguel Reula*
Dept. d'Estadística i Investigació Operativa, Universitat de València, Spain

José M. Sanchis
Dept. de Matemática Aplicada, Universidad Politécnica de Valencia, Spain

Abstract

The Close-Enough Arc Routing Problem, also called Generalized Directed Rural Postman Problem, is an arc routing problem with interesting real-life applications, such as routing for meter reading. In this application, a vehicle with a receiver travels through a series of neighborhoods. If the vehicle gets within a certain distance of a meter, the receiver is able to record the gas, water, or electricity consumption. Therefore, the vehicle does not need to traverse every street, but only a few, in order to be close enough to each meter. In this paper we deal with an extension of this problem, the Distance-Constrained Generalized Directed Rural Postman Problem or Distance-Constrained Close Enough Arc Routing Problem, in which a fleet of vehicles is available. The vehicles have to leave from and return to a common vertex, the depot, and the length of their routes must not exceed a maximum distance (or time). For solving this problem we propose a matheuristic that incorporates an effective exact procedure to optimize the routes obtained. Extensive computational experiments have been performed on a set of benchmark instances and the results are compared with those obtained with an exact procedure proposed in the Literature.

Keywords: Close-Enough Arc Routing Problem, Generalized Directed Rural Postman Problem, distance constraints, matheuristic, multi-start.

1 Introduction

Arc routing problems consist basically of finding one or several routes traversing a fixed set of arcs or edges that must be served. An exhaustive recent annotated bibliography of this area can be found in Mourão and Pinto [11], while Corberán and Laporte [5] summarizes the state of the art of this family of problems and its applications. Some examples of real-life applications include routing for meter readings corresponding to electricity, water or gas. While traditionally these readings

*corresponding author: miguel.reula@uv.es

had to be performed door to door and thus the vehicles had to traverse all the streets in which such meters where located, new technologies allow companies to remotely collect the data by using radio frequency receivers installed in vehicles that can collect the information sent by the metering devices. Therefore, it will be enough for a vehicle to traverse any street within a certain distance of the meter.

This type of application was first studied for a single vehicle by Shuttleworth et al. [13], who called this problem the Close-Enough Traveling Salesman Problem since they considered it a node routing one. The same problem was studied by Hà et al. [10], who called it the Close-Enough Arc Routing Problem (CEARP) since they realized it is an arc routing one. These authors define the problem as follows. "Consider a set of customers that must be covered. These customers can be located anywhere in the area covered by the network, not only in the network itself. The CEARP consists in finding a minimum-cost tour, which begins and ends at the depot, such that every customer is covered by the tour, that is, lies within a distance r of an arc of the tour". Since each customer has associated a subset of arcs, of which at least one has to be traversed in order the customer being serviced, this problem can also be considered a Generalized Directed Rural Postman Problem (GDRPP). This problem was introduced by Drexl ([7], [8]).

Drexl ([7], [8]) proved that the GDRPP is NP-hard by noting that the Directed Rural Postman Problem (DRPP) reduces to a GDRPP in which each customer can be served from a single arc. He also proposed a formulation and a branch-and-cut algorithm producing good computational results. Hà et al. [10] proposed a new formulation for the CEARP/GDRPP and a branch-and-cut algorithm providing very good computational results on large-size instances. Two new mathematical formulations and a polyhedral study were presented in Ávila et al. [2], as well as a branch-and-cut algorithm using several families of new inequalities. The paper includes a comparison of the performance of this algorithm with that by Hà et al. [10]. More recently, Cerrone et al. [4] have proposed some techniques to reduce the size of the graph and a new flow-based formulation. The computational results obtained with this new formulation on a first set of instances proposed in [10] are better than those of Hà et al. and slightly worse than the ones reported in [2] although obtained in marginally shorter computing times. Moreover, the Stochastic CEARP has been studied in Renaud et al. [12]. The stochasticity lies in the uncertainty of collecting data due to failed transmissions. Authors assume that it is possible that the remote reading of the meter fails. Then, they introduce the probability of reading a meter as a function of the distance of the customer from the route taken by the operator. For this problem, authors propose a mathematical formulation, give some preprocessing properties, and propose a cutting-plane algorithm and several heuristics for its solution. Aráoz et al. [1] study the Generalized Arc Routing Problem (GARP) on an undirected graph. This problem is a special case of the GDRPP in which the customers are associated with clusters of edges that are pairwise-disjoint connected subgraphs. For this problem, authors present some facets and valid inequalities and propose a branch-and-cut algorithm. Similar problems have also been studied for node routing problems and are known as covering tour problems (see Gendreau et al. [9] for details).

All the previously mentioned papers deal with a single vehicle. As far as we know, the only article considering a fleet of vehicles is the one by Ávila et al. [3]. This work considers the problem of finding a set of K routes leaving from and entering at the depot and serving all the customers, such that the length (in distance or time) of each route does not exceed a certain value T_{max}. The objective is to minimize the total length traversed. They noted that the problem is NP-Hard, since it generalizes the CEARP. For this problem, several formulations and exact algorithms were proposed and compared on a set of instances with up to five vehicles, 196 vertices, 450 arcs, and 150 customers.

In this paper we study this latter problem, called the Distance-Constrained Close-Enough Arc Routing Problem (DC-CEARP), for which we propose a multi-start matheuristic that incorporates an effective branch-and-cut method for the CEARP in order to optimize the routes obtained. In Section 2 we define formally the DC-CEARP and introduce the notation used. Moreover, the most promising formulation of the problem, according to [3], is given. The matheuristic is presented in Section 3 and the computational experiments are described in Section 4. Finally, some conclusions and future lines of research are given in Section 5.

2 Problem definition and notation

The DC-CEARP is defined on a strongly connected directed graph $G = (V, A)$, where V denotes the set of vertices and A the set of arcs. Vertex 1 is the depot, and, for each arc $(i, j) \in A$, there are a cost c_{ij} and a distance or time t_{ij} associated with its traversal. There is a set of L customers, each customer m having an associated set of arcs $H_m \subseteq A$ from which it can be served. Let us call $\mathbb{H} = \{H_1, \ldots, H_L\}$ the family of these sets. We assume there is a fleet of K identical vehicles based at the depot. The duration or length of the routes of these vehicles must not exceed a maximum travel time or distance denoted by T_{max}. The objective of the DC-CEARP is to find a set of K routes, starting and ending at the depot, with minimum total cost and such that at least one arc form each H_m, $m = 1, \ldots, L$, is traversed by the routes.

In what follows, $\mathbb{K} = \{1, \ldots, K\}$ will represent the set of vehicles and $A_R = H_1 \cup H_2 \cup \ldots \cup H_L$ the set of required arcs. The arcs in the set $A_{NR} = A \setminus A_R$ are called non-required arcs. Given a set $S \subset V$, we define $\delta^+(S) = \{(i, j) \in A : i \in S,\ j \in V \setminus S\}$, $\delta^-(S) = \{(i, j) \in A : i \in V \setminus S,\ j \in S\}$ and $A(S) = \{(i, j) \in A : i, j \in S\}$. Finally, given a set F of arcs, $x^k(F) = \sum_{(i,j)\in F} x^k_{ij}$.

In the paper by Ávila et al. [3] four different formulations for the DC-CEARP were presented. The formulation that provided the best computational results was called F_{xz} and is presented in what follows. Let us define the variables:

x^k_{ij} = number of times that the vehicle k traverses arc $(i, j) \in A$,

$$z^k_m = \begin{cases} 1, & \text{if the customer } m \text{ is served by vehicle } k \\ 0, & \text{otherwise.} \end{cases}$$

The problem can be formulated as

$$\text{Minimize} \qquad \sum_{k\in\mathbb{K}} \sum_{(i,j)\in A} c_{ij}\, x^k_{ij}$$

s.t.:

$$x^k(\delta^+(i)) = x^k(\delta^-(i)) \qquad \forall\, i \in V,\ \forall k \in \mathbb{K} \tag{1}$$

$$x^k(\delta^+(S)) \geq z_m^k - x^k(H_m \cap A(V \setminus S)) \qquad \forall S \subset V \setminus \{1\},\ \forall m = 1, \ldots, L,\ \forall k \in \mathbb{K} \tag{2}$$

$$\sum_{k \in \mathbb{K}} z_m^k = 1 \qquad \forall m = 1, \ldots, L \tag{3}$$

$$\sum_{(i,j) \in H_m} x_{ij}^k \geq z_m^k \qquad \forall m = 1, \ldots, L,\ \forall k \in \mathbb{K} \tag{4}$$

$$\sum_{(i,j) \in A} t_{ij}\, x_{ij}^k \leq T_{max} \qquad \forall k \in \mathbb{K} \tag{5}$$

$$x_{ij}^k \geq 0 \ \text{ and integer} \qquad \forall (i,j) \in A,\ \forall k \in \mathbb{K} \tag{6}$$

$$z_m^k \in \{0, 1\} \qquad \forall m = 1, \ldots, L,\ \forall k \in \mathbb{K} \tag{7}$$

Inequalities (1) are the symmetry conditions on the vertices. Constraints (2) ensure the connectivity of the routes. Note that, if vehicle k does not serve customer m, $z_m^k = 0$, or it serves customer m by traversing an arc in $H_m \cap A(V \setminus S)$, then the inequality is trivially satisfied. On the other hand, if vehicle k serves customer m by traversing an arc not in $H_m \cap A(V \setminus S)$, it has to traverse the cutset $\delta(S)$. The mandatory service of the customers is established in inequalities (3), and inequalities (4) link the two sets of variables. Finally, constraints (5) limit the duration of the routes and (6) and (7) are the non-negativity and integrality constraints.

Using this formulation, Ávila et al. [3] implemented a branch-and-cut algorithm capable of solving instances with up to five vehicles, 196 vertices, 450 arcs, and 150 customers. Nevertheless, the algorithm could not find the optimal solution in 30 out of the 251 instances and in seven of these 30 was not even able to produce a feasible solution after one hour of computing time. This, and the need of solving larger instances, as well as to provide good upper bounds that can help the exact algorithm, motivated us to develop a heuristic algorithm for the DC-CEARP.

3 Matheuristic

The proposed algorithm is a multi-start matheuristic. In each iteration of the algorithm, the routes are first generated using a heuristic construction algorithm and they are then improved by two local search procedures. Finally, each route is optimized by means of an exact procedure. The algorithm stops when a certain number of iterations with no improvement is reached or a maximum computation time is exceeded.

Before applying this algorithm, we first remove those customers m_1 such that there is another customer m_2 for which $H_{m_2} \subseteq H_{m_1}$.

3.1 Constructive algorithm

In this section we describe how to construct feasible solutions for the DC-CEARP by iteratively assigning each customer to a route. Since the total duration of the routes could be improved in later phases of the algorithm, we allow the duration of the constructed routes to exceed the time limit by a certain amount given by a parameter *Marg* varying between 0 and 0.2. Let $T_L = (1+Marg) \times T_{max}$ be the maximum duration allowed for the routes in this phase of the algorithm.

A route for vehicle i will be represented as the sequence R_i of required arcs traversed by vehicle i. We assume that, in order to travel from the end of a required arc to the beginning of the following one, the vehicle uses the shortest path. Associated with each route i, we have a vector M_i containing the customers which are served from the required arcs in R_i.

The constructive algorithm consists of two phases: the first one initializes the routes, and the second one completes them by iteratively allocating the customers not yet assigned to any route. We have implemented six variants of the constructive method, resulting from the combination of three different initialization procedures with two completion methods that are described in what follows.

3.1.1 Initialization procedures

We define the service time of a customer m, and denote it st_m, as the shortest time needed for traveling from the depot to an arc in H_m, traversing it, and returning to the depot. Based on this service time, we have considered different ways of introducing the first arc in each route.

- *Random Initialization*
  To initialize a route i, we randomly select a customer m among those that have not been assigned to any route yet. Among all the arcs in H_m, we select the arc a_m associated with the minimum service time st_m. Likewise, a_m is introduced in R_i, and both m and the rest of unassigned customers that can be served by a_m are included in M_i.

- *Random Selection among the Best Applicants*
  We define BA (Best Applicants) as the set formed by the $min\{10, \frac{L}{2}\}$ unassigned customers with the highest st_m values.

  To initialize the route i, we randomly select a customer $m \in BA$ and its corresponding arc $a_m \in H_m$ with minimum service time st_m is inserted in R_i. Customer m, together with all the other unassigned customers that can be served by arc a_m, is added to M_i.

- *Weighted Selection among the Best Applicants*
  A customer m is randomly chosen from the set BA defined above using probabilities

  $$p_m = \frac{st_m}{\sum_{i \in BA} st_i}.$$

  Customer $m \in BA$ is introduced in M_i and the arc $a_m \in H_m$ associated with st_m is included in R_i. Again, all the unassigned customers that can be served by a_m are included in M_i.

Note that inserting an arc in an empty route implies traveling from the depot to this arc and returning back using shortest paths. These shortest paths are studied to check if they traverse any arcs from which unassigned customers could be served. If this is the case, these customers are added to the corresponding route and marked as assigned, and the arcs from which they are served are marked as required and added to the route in the corresponding position.

3.1.2 Route completion procedures

Once the routes have been initialized using one of the variants explained above, a deterministic completion procedure is used to introduce the remaining customers in the routes by iteratively

adding arcs to the partial routes. Note that an arc that is inserted in a route i may also serve other unassigned customers. These customers are also added to M_i. Likewise, inserting an arc in a route may result in changing the shortest paths traversed between required arcs. If these new shortest paths traverse any required arc that can serve unassigned customers, these customers are also added to M_i, and the corresponding required arc is included in R_i.

Two different completion methods are used: one completing all the routes simultaneously and another one working in a sequential way, completing each route before moving to the next one.

- *Parallel Completion*
  Once all the routes have been initialized, we start by allocating the unassigned customers m with $|H_m| = 1$, since these are the customers with less flexibility to be inserted in the routes. For each of these customers we check all the possible positions and routes in which its corresponding arc can be inserted, and choose the one that produces a minimum increase of the route duration.

  For each arc a of the remaining unassigned customers, we calculate how much the duration of the route increases for all the possible positions and all the possible routes in which we can insert a. The minimum of these values is called the insertion cost of a. Note that an insertion in a route is considered possible if the resulting route time does not exceed T_L. For each unassigned customer m, let a^*_m be the arc in H_m with minimum insertion cost. Then, we choose the customer m^* with the maximum insertion cost $a^*_{m^*}$. Arc $a^*_{m^*}$ is inserted in the best possible position and route, say R_i, and m^* is added to M_i.

- *Sequential Completion*
  In this case, after a route i is initialized, it is completed until no more customers can be allocated to it without exceeding T_L.

  For each unassigned customer, we obtain the arc and the position in which it should be introduced so that the increase of the duration of the route is minimum. This process is repeated until no customer can be inserted in the route without exceeding T_L, i.e, the current route has been completed. If there are some customers that are not allocated yet to any route and there is an empty route, a new route is initialized.

3.2 Local search

The solutions obtained in the construction phase are improved using two local search procedures. The methods *2-Exchange*, based on the exchange of two arcs from different routes, and *destroy and repair*, where some arcs and their corresponding customers are removed from the solution and then reintroduced, are described.

3.2.1 2-Exchange

The *2-Exchange* procedure consists of swapping two arcs of different routes. Note that when two arcs of different routes are exchanged, the customers associated with each arc must be exchanged too. As it is explained below, this exchange is applied following the *first improvement* strategy.

The routes are sorted randomly and the first two routes R_i and R_j are selected. Then, an arc a^*_i from R_i and an arc a^*_j from R_j are deleted and the customers served from a^*_i and a^*_j are removed from M_i and M_j, respectively. Arc a^*_j is placed in the best possible position of the route i, and arc a^*_i in the best possible position of the route j. The customers served from the arc a^*_j

are included now in M_i, and those served from a_i^* are included in M_j. If the obtained solution improves the current one and T_L is not exceeded, the movement is accepted and another pair of routes is selected. Otherwise, the movement is discarded and another pair of arcs of R_i and R_j is chosen until an improving movement is found or all the possible exchanges for this pair of routes have been tried.

3.2.2 Destroy and repair

This method consists of two phases: in the first one, some arcs are removed from the current solution, while in the second one of the customers that cannot be served due to the removed arcs are reallocated.

In the destruction phase of the algorithm, d arcs are removed from the solution but always keeping at least one arc in any route. Specifically, an arc is randomly selected and, if it is not the only arc in its route, it is removed. Let a_i be an arc removed from R_i and m a customer served from this arc. If there is another arc in R_i from which customer m can be served, m does not need to be reallocated. If this is not the case but there is an arc in another route R_j from which m can be served, m is reassigned to route R_j without modifying the routes. Otherwise, customer m will be marked as unassigned in order to be reallocated by the repair phase.

Once the destruction phase is finished, all the unassigned customers must be reallocated. This is done using the *parallel completion* procedure described in Section 3.1.2. If the resulting solution does not improve the original one, all the changes made in this phase are discarded.

This procedure is repeated for each value of $d \in \{1, 2, 5\}$ until 100 iterations without an improvement are performed.

3.3 Optimization of the routes

Finally, we optimize each single route by solving a single-vehicle CEARP problem using the exact branch-and-cut algorithm developed in [2].

Given a route i, we define a CEARP instance on graph G with the set of customers M_i. Each customer $m \in M_i$ has the same associated set of required arcs H_m as in the original DC-CEARP instance. This CEARP instance is then solved optimally. If the duration of the optimal route is greater than T_{max}, the route is not feasible and the current solution is discarded.

If the obtained route is feasible, it may now traverse some required arcs that serve customers which are currently assigned to other routes. If we can now serve a new customer m that belongs to another route j that has not been optimized yet, we add m to M_i and remove it from M_j. Since the order of selecting the routes to be optimized can have an effect in the optimization of the remaining routes, the selection is made at random.

3.4 Overall algorithm

The multi-start matheuristic consists basically of generating solutions using the different constructive algorithms presented above and applying the local-search procedures and the exact optimization to the individual routes. This procedure is repeated until a certain stopping criterium is met. Specifically, the matheuristic (Algorithm 1) is initialized by creating a solution with one of the six constructive algorithms proposed. Note that, since this solution is built using a time or distance

limit for the routes $T_L = (1+Marg) \times T_{max}$, if $Marg > 0$, the duration of some routes may exceed T_{max} and thus the solution may not be feasible. The obtained solution is improved by means of the local-search algorithm and then each route is optimized using the procedure described in Section 3.3. This construction and improvement phase is repeated for all the different constructive algorithms. If the best feasible solution (if any) obtained in this way is better than the current best solution, it is stored as the new best solution. This procedure is repeated until a maximum computing time *time_limit* is exceeded or a certain number of iterations *iter_max* without updating the best known solution is reached.

The overall algorithm (Algorithm 2) applies the multi-start procedure with values $Marg = 0, 0.05, 0.1$ and a maximum number of iterations without improvement of $iter\_max1$. If no feasible solution has been found, the multi-start procedure is used again with maximum number of iterations $iter\_max2$ and increasing the value of *Marg* by 0.02 iteratively. The procedure stops when a feasible solution is found, $Marg > 0.2$, or the computing time *time_limit* is exceeded, whatever happens first.

Input: G, $\mathbb{H}$, T_{max}, *Marg*, *iter_max*, *time_limit*
Output: S_{best}
$T_L \leftarrow (1+Marg) \times T_{max}$;
$iter \leftarrow 0$;
$S_{best} \leftarrow \emptyset$;
while *time_limit* is not reached AND $iter \leq iter\_max$ **do**
  $S_{iter} \leftarrow \emptyset$;
  for each Constructive algorithm **do**
    $S_c \leftarrow$ Constructive algorithm(T_L);
    $S_i \leftarrow$ Local-Search(S_c, T_L);
    $S_o \leftarrow$ Routes optimization(S_i, T_{max});
    if S_o is feasible and better than S_{iter} **then**
      $S_{iter} \leftarrow S_o$;
  if S_{iter} is feasible and better than S_{best} **then**
    $S_{best} \leftarrow S_{iter}$;
    $iter \leftarrow 0$;
  else
    $iter \leftarrow iter + 1$;

Algorithm 1: Multi-start

Input: G, $\mathbb{H}$, T_{max}, $iter\_max1$, $iter\_max2$, *time_limit*
Output: S_{best}
$Marg \leftarrow 0$;
$S_{best} \leftarrow \emptyset$;
while *time_limit* is not reached AND $Marg \leq 0.1$ **do**
  $S_{best} \leftarrow$ Multi-start(G, $\mathbb{H}$, T_{max}, *Marg*, $iter\_max1$, *time_limit*);
  $Marg \leftarrow Marg + 0.05$;
if $S_{best} = \emptyset$ **then**
  while $S_{best} = \emptyset$ AND *time_limit* is not reached AND $Marg \leq 0.2$ **do**
    $S_{best} \leftarrow$ Multi-start(G, $\mathbb{H}$, T_{max}, *Marg*, $iter\_max2$, *time_limit*);
    $Marg \leftarrow Marg + 0.02$;

Algorithm 2: Overall

4 Computational experiments

In this section we study the performance of the proposed algorithm. The instances tried and the computational results obtained are described in the following sections. The procedures have been coded in C++ and all the computational experiments have been executed on a single thread of an Intel Core i7 at 3.4GHz with 32GB RAM running Windows 10 Enterprise 64 bits. The branch-and-cut algorithm for the single-vehicle CEARP used for optimizing the routes (see Section 3.3), developed in [2], was also implemented in C++ using CPLEX 12.4 MIP Solver with Concert Technology 2.9. Although this exact procedure is able to solve large-sized GDRPP instances, it can be quite time consuming. Therefore, we have limited its execution time to 10 seconds.

4.1 Instances

The algorithm has been tested on the four sets of instances proposed in Ávila et al. [3]. The instances of the two first sets, called *Random50* and *Random75*, were generated randomly in a 1000 x 1000 square, with $|V| \in \{50, 75\}$. The other two sets of instances, called *Albaida* and *Madrigueras* are based on the street networks of these two Spanish towns. There is a total of 251 instances whose characteristics are shown in Table 1. All the data, including the values of T_{max} and the number of vehicles, as well as the detailed results described in Section 4.2, can be found in [6].

			$\|A_R\|$		$\|A_{NR}\|$		L	
	$\|V\|$	$\|A\|$	Min	Max	Min	Max	Min	Max
Random50	50	300	105	292	7	193	10	100
Random75	75	450	143	438	10	305	15	150
Albaida	116	174	83	99	75	91	19	34
Madrigueras	196	316	152	181	135	164	23	48

Table 1: *Characteristics of the instances*

4.2 Computational results

The results obtained with two versions of the matheuristic are compared to those obtained with the branch-and-cut in Ávila et al. [3]. Table 2 summarizes the results obtained with a version of the algorithm with $iter\_max1 = 5$, $iter\_max2 = 20$, and $time\_limit = 100$ (seconds), which will be called *Matheuristic 1* (MH1). The goal of *Matheuristic 1* is to obtain good solutions in low computing times. Table 2 shows the results obtained only for the instances for which the branch-and-cut obtained an optimal solution. Columns 1 and 2 contain the name of the instance set and the number of vehicles. Column 3 reports the number of instances with known optimal solution. Column 4 shows the number of instances for which *Matheuristic 1* reached the known optimal solution, while Column 5 shows the number of instances for which the algorithm found feasible but not optimal solutions. For the instances reported in Column 5, we have computed the gap between the cost of the solution provided by *Matheuristic 1* and the optimal value. The average gap values are shown in Column 6. The number of instances for which the algorithm was not even capable of finding a feasible solution is given in Column 7. The last two columns report the average time in seconds taken by *Matheuristic 1* and the branch-and-cut, respectively, for all the instances. Note that the times reported for *Matheuristic 1* are lower than $time\_limit = 100$.

This is due to the values used for parameters $iter\_max1$ and $iter\_max2$ limiting the running time of the algorithm.

Globally, the algorithm *Matheuristic 1* has been capable of reaching the optimal solution on 115 out of 221 instances, in 22.91 seconds on average. The average gap from the optimal value in the 101 instances for which a feasible solution was found is 3.93%. However, there are 5 instances for which *Matheuristic 1* has not been able to find a feasible solution with the given number of vehicles. This may be explained by the fact that the values of T_{max} limiting the length of the routes are very tight. As expected, the *Random75* instances are more difficult to solve to optimality than the *Random50* ones because of their larger sizes. Something similar happens with the *Madrigueras* instances with respect to the *Albaida* ones, although, perhaps due to the structure of their underlying graphs, the number of instances for which no feasible solution has been found is greater in the *Albaida* set than in the *Madrigueras* set.

							Time (s)	
	#Veh	#Inst	#Opt	#No Opt	Gap(%)	#No Sol	MH1	B&C
Random50	2	12	8	4	3,76	0	1,97	45,90
	3	11	6	5	4,67	0	3,16	83,00
	4	9	4	5	4,76	0	4,33	179,30
	5	2	0	2	4,01	0	12,31	456,40
		34	**18**	**16**	**4,39**	**0**	**3,59**	**117,40**
Random75	2	12	6	6	4,30	0	2,65	388,5
	3	12	5	7	4,40	0	6,61	603,1
	4	10	3	7	3,94	0	13,27	771,1
	5	4	2	2	3,82	0	5,62	1301,1
		38	**16**	**22**	**4,17**	**0**	**7,01**	**654,28**
Albaida	2	24	21	3	0,76	0	14,89	34
	3	24	18	6	5,90	0	21,03	89,9
	4	21	9	12	3,17	0	22,87	338,7
	5	17	10	3	3,43	4	40,07	316,1
		86	**59**	**25**	**3,59**	**4**	**23,53**	**182,1**
Madrigueras	2	24	11	13	2,82	0	39,00	224,1
	3	21	6	15	3,96	0	43,09	894,1
	4	13	4	9	5,20	0	39,85	1625,2
	5	5	2	2	3,08	1	58,64	2447,2
		63	**23**	**39**	**3,82**	**1**	**42,09**	**912,9**
Total		**221**	**115**	**101**	**3,93**	**5**	**22,91**	**462,95**

Table 2: *Results of* Matheuristic 1 *for the instances with known optimal solution*

In order to obtain better solutions, we have tried another version of the algorithm, called *Matheuristic 2*, with the following values of the parameters: $iter\_max1 = iter\_max2 = 200$ and $time\_limit = 600$ (seconds). The obtained results are shown in Table 3. The reader can observe the different behavior of *Matheuristic 2* against *Matheuristic 1*. Now, the number of optimal solutions found is 161 out of 221 versus 115 of the faster version. Also, the average gap for the instances that were not solved optimally is 2.41% against 3.93% of *Matheuristic 1*. Finally, only in 2 out of the 221 instances was *Matheuristic 2* not capable of finding a feasible solution with the specified number of vehicles. Of course, all these better results have been obtained at the expense of a greater computational effort (151 seconds on average versus 22 seconds of *Matheuristic 1*).

Table 4 shows the computational results obtained on the instances of the *Random50*, *Random75*,

	#Veh	#Inst	#Opt	#No Opt	Gap(%)	#No Sol	Time (s) MH2	Time (s) B&C
	2	12	9	3	4,48	0	13,48	45,90
	3	11	9	2	0,42	0	26,90	83,00
Random50	4	9	5	4	2,08	0	21,07	179,30
	5	2	1	1	0,57	0	35,23	456,40
		34	**24**	**10**	**2,32**	**0**	**21,11**	**117,40**
	2	12	8	4	2,67	0	22,09	388,5
	3	12	6	6	3,20	0	26,20	603,1
Random75	4	10	5	5	3,41	0	36,08	771,1
	5	4	2	2	2,57	0	34,81	1301,1
		38	**21**	**17**	**3,06**	**0**	**28,41**	**654,28**
	2	24	21	3	0,32	0	117,19	34
	3	24	22	2	1,15	0	152,33	89,9
Albaida	4	21	18	3	1,23	0	159,47	338,7
	5	17	11	4	0,96	2	187,88	316,1
		86	**72**	**12**	**0,96**	**2**	**151,30**	**182,1**
	2	24	18	6	1,20	0	323,19	224,1
	3	21	13	8	3,23	0	327,85	894,1
Madrigueras	4	13	9	4	4,19	0	251,76	1625,2
	5	5	4	1	3,47	0	172,04	2447,2
		63	**44**	**19**	**2,80**	**0**	**298,01**	**912,9**
Total		**221**	**161**	**58**	**2,41**	**2**	**151,04**	**462,95**

Table 3: *Results of* Matheuristic 2 *for the instances with known optimal solution*

and *Madrigueras* sets with unknown (or unproven) optimal solutions. In fact, for some of these instances, the branch-and-cut algorithm described in [3] was not capable of finding even a feasible solution in one hour of computing time, which gives an idea of the difficulty of these instances. Columns 1 and 2 of Table 4 report the instance name and the number of vehicles. Column 3 shows the value of the best solution found (if any) by the exact algorithm, while column 4 reports the corresponding optimality gap. The values of the solutions provided by *Matheuristic 1* and *Matheuristic 2* and the computing times in seconds can be seen in the following four columns of the table. The last three columns present the gap obtained by the three algorithms with respect to the best solution found. Note that our algorithm *Matheuristic 2*, not only improves the solutions provided by the branch-and-cut method in many instances, but also finds a feasible solution in all of them.

Table 5 summarizes the results shown in Table 4 obtained by the branch-and-cut and *Matheuristic 2* algorithms for the 30 instances not solved to optimality. It can be seen in the first row that *Matheuristic 2* was able to find a feasible solution for all the 30 instances while the branch-and-cut algorithm was only able to find it for 23 of them. The average UB and the average time (second and fourth rows, respectively) refer only to these 23 instances. The number of instances for which *Matheuristic 2* obtained a better solution, 17, includes the 7 instances where the branch-and-bound did not obtain a feasible solution. As can be seen, in these 30 instances, *Matheuristic 2* clearly outperforms the branch-and-cut algorithm.

Instances	# Veh	B&C		MH 1		MH 2		Gap Best Sol (%)		
		UB	Gap	UB	Time	UB	Time	B&C	MH1	MH2
M3103_gdrpp	3	9052	0,05	9348	100	9237	600	0	0,03	0,02
M3105_gdrpp	3	9454	0,12	9086	100	9086	600	0,04	0	0
M3201_gdrpp	3	9795	0,04	10612	56,83	10131	600	0	0,08	0,03
LCGDRPP_75_3_20_1	4	9952	0,07	-	100	9910	11,21	0,01	-	0
LCGDRPP_75_3_20_2	4	10684	0,09	11119	4,14	10976	21,30	0	0,04	0,03
M3101_gdrpp	4	-	-	9369	78,01	9307	600	-	0,01	0
M3103_gdrpp	4	9628	0,11	9954	100	9709	600	0	0,03	0,01
M3105_gdrpp	4	-	-	9211	100	9211	600	-	0	0
M3111_gdrpp	4	30660	0,12	29639	9,91	29637	600	0,03	0	0
M3201_gdrpp	4	10451	0,08	11386	100	10829	600	0	0,09	0,04
M3203_gdrpp	4	10145	0,07	10959	100	10344	600	0	0,08	0,02
M3205_gdrpp	4	11526	0,19	10824	100	10768	600	0,07	0,01	0
M3211_gdrpp	4	38869	0,11	-	100	38992	600	0	-	0,01
M5209_gdrpp	4	8834	0,01	8943	38,83	8834	297	0	0,01	0
LCGDRPP_50_3_20_3	5	9648	0,08	9903	2,15	9807	51,33	0	0,03	0,02
LCGDRPP_75_3_10_3	5	9858	0,14	10094	4,70	9978	31,51	0	0,02	0,01
LCGDRPP_75_3_20_1	5	10615	0,15	10640	2,53	10826	17,88	0	0	0,02
LCGDRPP_75_3_20_2	5	11728	0,17	12001	15,29	11871	7,44	0	0,02	0,01
LCGDRPP_75_3_20_3	5	-	-	8847	61,45	8916	61,92	-	0	0
M3101_gdrpp	5	-	-	10374	100	10374	600	-	0	0
M3103_gdrpp	5	-	-	11477	100	11139	600	-	0,03	0
M3105_gdrpp	5	-	-	9461	65,17	9343	436	-	0,01	0
M3107_gdrpp	5	5368	0,10	5604	14	5234	214	0,03	0,07	0
M3109_gdrpp	5	10794	0,09	11272	21,61	10973	465	0	0,04	0,02
M3111_gdrpp	5	29686	0,09	30044	41,69	29686	600	0	0,01	0
M3201_gdrpp	5	12224	0,26	11947	100	11394	600	0,07	0,05	0
M3203_gdrpp	5	13536	0,42	11566	100	11041	600	0,23	0,05	0
M3205_gdrpp	5	-	-	11220	100	10638	600	-	0,05	0
M3209_gdrpp	5	11933	0,03	-	100	12453	75,51	0	-	0,04
M3211_gdrpp	5	40744	0,26	41837	65,49	39787	600	0,02	0,05	0

Table 4: *Results for the 30 instances with unknown optimal solution*

	B&C	Matheuristic 2
# of feasible solutions	23/30	30/30
Average UB	14573.22	14413.17
# of best solutions	15/30	17/30
Average time	3600	405.34

Table 5: *Overall results for the unsolved instances*

In order to assess the contribution of the different constructive algorithms, we have studied the number of times each algorithm has been able to produce the best solution. Table 6 reports this information for all the six variants of the constructive algorithms in all the 251 instances used. It presents the percentage of best solutions found both when the route optimization is used and when it is deactivated. It can be seen that heuristics working in a parallel way perform better than those that use sequential completion. Moreover, this behavior remains true whether the optimization phase is executed or not.

	Parallel completion			Sequential completion		
	RI	RSBA	WSBA	RI	RSBA	WSBA
Without optimization	20.92	39.04	33.07	13.94	10.56	13.55
With optimization	23.51	38.25	31.08	16.14	13.35	12.75

Table 6: *Percentage of best solutions found with each constructive algorithm with and without the exact route optimization phase*

		#Opt	#No Opt	Gap(%)	#No Sol	Time (s)	T_{Opt}
With optimization	MH1	115	101	3,93	5	22,91	95,97%
	MH2	161	58	2,41	2	151,04	91,74%
Without optimization	MH1	72	138	3,05	11	22,97	–
	MH2	89	128	2,29	4	151,55	–

Table 7: *Impact of the exact route optimization phase on the instances with known optimal solution*

Finally, in order to study the impact of the exact route optimization phase on the performance of the matheuristic, we have run a variant of MH1 and MH2 in which we have deactivated the optimization phase. These variants have been executed during the same running time used by the original versions of MH1 and MH2. Table 7 shows the obtained results. Last column reports the percentage of time used by the exact optimization procedure with respect to the total computing time. Although the time used in the optimization phase may seem high, if we compare, for example, the number of instances in which the optimum has been obtained or the number of those in which no solution has been found, we can conclude that the optimization of the routes plays an important role in the performance of the matheuristic.

5 Conclusions and future work

In this paper we have addressed the generalization of the Close-Enough Arc Routing Problem to the case with several vehicles and maximum distance (or time) constraints. For this problem, we have proposed a matheuristic. This procedure incorporates the exact algorithm for the single vehicle case presented in [2] in order to optimize the routes obtained. We have performed extensive computational experiments on a set of benchmark instances and the results have been compared with those obtained with the exact procedure proposed by Ávila et al. [3]. The proposed algorithm has been able to solve to optimality 117 out of 221 instances in short computing times.

In a future work, we plan to use the upper bounds provided by this algorithm in the design of an exact algorithm capable of solving instances of a larger size than those currently solved by the algorithm in [3].

Acknowledgements: This work was supported by the Spanish Ministerio de Economía y Competitividad and Fondo Europeo de Desarrollo Regional (FEDER) through project MTM2015-68097-P (MINECO/FEDER). Authors want to thank two anonymous referees for their suggestions and comments that have contributed to improve the paper.

References

[1] J. Aráoz, E. Fernández, and C. Franquesa. The generalized arc routing problem. *TOP*, 25:497–525, 2017.

[2] T. Ávila, Á. Corberán, I. Plana, and J. M. Sanchis. A new branch-and-cut algorithm for the generalized directed rural postman problem. *Transportation Science*, 50:750–761, 2016.

[3] T. Ávila, Á. Corberán, I. Plana, and J. M. Sanchis. Formulations and exact algorithms for the distance-constrained generalized directed rural postman problem. *EURO Journal on Computational Optimization*, 5:339–365, 2017.

[4] C. Cerrone, R. Cerulli, B. Golden, and R. Pentangelo. A flow formulation for the close-enough arc routing problem. In Sforza A. and Sterle C., editors, *Optimization and Decision Science: Methodologies and Applications. ODS 2017.*, volume 217 of *Springer Proceedings in Mathematics & Statistics*, pages 539–546. 2017.

[5] Á. Corberán and G. Laporte (editors). *Arc Routing: Problems, Methods, and Applications.* MOS-SIAM Series on Optimization, Philadelphia, 2014.

[6] Á. Corberán, I. Plana, and J.M Sanchis. Arc routing problems: data instances. *http://www.uv.es/∼corberan/instancias.htm.*

[7] M. Drexl. *On some generalized routing problems.* PhD thesis, Rheinisch-Westfälische Technische Hochschule, Aachen University, 2007.

[8] M. Drexl. On the generalized directed rural postman problem. *Journal of the Operational Research Society*, 65:1143–1154, 2014.

[9] M. Gendreau, G. Laporte, and F. Semet. The covering tour problem. *Operations Research*, 45:568–576, 1997.

[10] M.-H. Hà, N. Bostel, A. Langevin, and L.-M. Rousseau. Solving the close enough arc routing problem. *Networks*, 63:107–118, 2014.

[11] M. C. Mourão and L. S. Pinto. An updated annotated bibliography on arc routing problems. *Networks*, 70:144–194, 2017.

[12] A. Renaud, N. Absi, and D. Feillet. The stochastic close-enough arc routing problem. *Networks*, 69:205–221.

[13] R. Shuttleworth, B.L. Golden, S. Smith, and E.A. Wasil. Advances in meter reading: Heuristic solution of the close enough traveling salesman problem over a street network. In B.L. Golden, S. Raghavan, and E.A. Wasil, editors, *The Vehicle Routing Problem: Lastest Advances and New Challenges*, pages 487–501. Springer, 2008.