Computers & Operations Research 39 (2012) 2380–2394

Contents lists available at SciVerse ScienceDirect

Computers & Operations Research

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

Capacitated arc routing problem with deadheading demands

Gokhan Kirlik[a],*, Aydin Sipahioglu[b]

[a] College of Engineering, Koç University, Sarıyer, Istanbul 34450, Turkey
[b] Department of Industrial Engineering, Eskisehir Osmangazi University, Eskisehir 26480, Turkey

ARTICLE INFO

Available online 22 December 2011

Keywords:
Capacitated arc routing problem
Deadheading edge demands
Mathematical model
Constructive heuristic
Post-optimization procedure

ABSTRACT

Capacitated arc routing problem (CARP) is the determination of vehicle tours that serve all positive-demand edges (required edge) exactly once without exceeding vehicle capacity while minimizing sum of all tour costs. In CARP, total demand of a tour is calculated by means of all required edges on the tour. In this study, a new CARP variation is introduced, which considers not only required edges but also traversed edges while calculating total demand of the tour. The traversing demand occurs when the traversed edge is either servicing or non-servicing (deadheading). Since the new CARP formulation incurs deadheading edge demands it is called CARP with deadheading demands. An integer linear model is given for the problem which is used to solve small-sized instances, optimally. A constructive heuristic is presented to solve the problem which is a modified version of a well-known CARP heuristic. Furthermore, two post-optimization procedures are presented to improve the solution of the heuristic algorithm. The effectiveness of the proposed methods is shown on test problems, which are obtained by modifying CARP test instances.

1. Introduction

The CARP is a distribution problem with many practical applications. It was introduced by Golden and Wong [1] and can be defined briefly as follows: given a set of identical vehicles with a limited capacity and an undirected graph with positive edge demands and edge costs, find a set of minimum cost tours for the vehicles that service all the positive-demand edges. Each route must contain the depot (a distinguished vertex), and the total demand it services cannot exceed the capacity of the vehicle.

Golden and Wong [1] showed that even obtaining the 1.5 approximate solution for any CARP instance is NP-hard. Furthermore, finding a feasible solution of the CARP that uses a given number of vehicles is also NP-hard. Exact methods for the CARP have only been able to solve relatively small examples to optimality.

Since the CARP is an NP-hard problem several constructive heuristics were proposed, which were reviewed in [2]. To obtain better results several metaheuristics have been applied to CARP. The first metaheuristic for the CARP, a simulated annealing procedure, was designed by Eglese [3] for solving a winter gritting problem. Tabu search algorithms were proposed such as CARPET by Hertz and Mittaz [4], and deterministic tabu search by Brandão and Eglese [5]. A memetic algorithm described by Lacomme et al. [6]. A variable neighborhood search heuristic by Polacek et al. [7]. An ant colony optimization scheme is proposed by Ghiani et al. for the CARP [8]. An extensive survey was presented by Wøhlk in 2008 [9].

In this study, a new CARP formulation is introduced. Unlike the classical CARP, in this formulation vehicle uses capacity for the traversing. So that, a new demand is defined for all edges of the graph. Traversing demand can be combined with servicing demand for the positive-demand edges. However, for the deadheading edges (traversing on an edge without servicing), there is no such correspondence between the classical CARP and the new CARP formulation. Since we consider deadheading demands in the formulation, this CARP variation is called capacitated arc routing problem with deadheading demands (CARPDD). CARP is the special case of the CARPDD, because when the deadheading demands are ignored, the problem is transformed into CARP. Consider Theorem 3.1 in Section 3 related to computational complexity of CARPDD.

In this study, mathematical programming model of the CARPDD is presented. This model provides both formal definition for the problem and exact solution for the small-sized instances. CARPDD is an NP-hard problem (see Theorem 3.1), so one of the well-known CARP heuristics,

* Corresponding author. Tel.: +90 212 338 1630; fax: +90 212 338 1548.
E-mail addresses: gkirlik@ku.edu.tr (G. Kirlik), asipahi@ogu.edu.tr (A. Sipahioglu).

Ulusoy partitioning algorithm [10], is adapted for the problem to solve the new CARP variation. The modified algorithm is a two-phase constructive heuristic and is called modified Ulusoy algorithm. Since the heuristic is a constructive one, for some instances it may lead poor results. So, two post-optimization procedures are presented to improve the obtained result. The proposed mathematical model, heuristic, and post-optimization procedures are tested by using modified CARP test instances.

The paper is organized as follows. In Section 2, motivation of the new CARP formulation is presented. Mathematical model of the CARPDD is given in Section 3. The modified constructive heuristic for the problem is presented in Section 4. Post-optimization procedures are described in Section 5. An illustrative example is presented in Section 6. Test problem generation and computational results are presented in Section 7. Finally, conclusions are given in Section 8.

2. Motivation

Complete coverage path planning is the determination of a path such that every point in a given workspace is covered at least once [11]. It is required in a variety of applications such as vacuum cleaning robots [12] and lawn mowers [13]. In these applications robot apparatus is expected to move over all the points. In some other applications, such as patrolling and search–rescue operations [14], it is enough to move the robot so that sensors on the robot cover all the points of the environment. In these applications, robot has not to pass over all the points of the environment. This type of coverage is referred as sensor-based coverage [15].

In sensor-based coverage of narrow or cluttered spaces, where obstacles lie within detector range, Generalized Voronoi Diagram (GVD) [16] can be used to model the environment. For complete sensor-based coverage of the given area, it is sufficient for the robot to follow the GVD [15]. Additionally, a representative graph of the environment could be constructed based on GVD. Then, traversing all edges of the GVD-based graph guarantees the sensor-based coverage of the environment. This problem is similar to Chinese postman problem (CPP), because obtaining minimum-cost tour that covers every edge of a graph at least once is referred as CPP [17]. When the environment is modeled with GVD-based graph which is an undirected graph, sensor-based coverage problem can be solved by using undirected CPP solution approach.

In the sensor-based coverage problem, multi-robot is more preferable compared to single robot. Using a robot team instead of single robot reduces the time required to complete the coverage task and enhances robustness [18]. Multi-robot sensor-based coverage path planning problem can be defined as constructing tour of each robot in the team such that every point in a given workspace is covered by at least one member of the team using its sensors. Some examples of this problem are landmine detection, foraging, patrolling, search–rescue [14], etc.

The multi-robot sensor-based coverage problem of narrow environments may also be described as partitioning arcs of the GVD-based graph among robots without exceeding their energy capacities. This problem resembles CARP. Vehicle capacity in CARP corresponds energy capacity of a robot in the coverage problem. In CARP, amount of material to be collected (or distributed) on an edge is defined as the demand. However, in coverage problem two different type of edge demands occur due to robot energy consumption. First type of energy consumption is coverage energy which is same as the CARP's edge demand (servicing demand). Second type of energy consumption is traversing energy, robot consumes energy while passing through an edge on GVD with or without covering [19]. These situations require different edge demand for a given edge which is called deadheading demand. This is the main difference between CARP and CARPDD.

3. Problem formulation

Let $G=(V,E)$ be an undirected connected graph where $V=\{v_1,v_2,\dots,v_n\}$ is a set of vertices, E is a set of edges (v_i,v_j), number of edges in G is m ($|E|=m$) and $R\subseteq E$ is a set of required edges. A fleet of identical vehicles, $k=\{1,2,\dots,K\}$, is based at depot vertex v_0 with Q amount of capacity. Each edge of the graph (v_i,v_j) incurs a non-negative cost (or length) c_{ij} whenever a vehicle travels over it or services a required edge. Each required edge of the graph (v_i,v_j) has a positive demand q_{ij}, for the rest of edges demand is zero ($q_{ij}=0$, $(v_i,v_j)\in E\backslash R$). Unlike classical CARP, in CARPDD each edge (v_i,v_j) has also deadheading demand named e_{ij}. Since, the notation for the problem is clarified, consider the following theorem related to computational complexity of the CARPDD.

Theorem 3.1. *CARPDD is an NP-hard problem.*

Proof. From the definition of CARPDD, we know that deadheading demand is nonnegative for all edges ($e_{ij}\geq 0\ \forall(v_i,v_j)\in E$) in the graph. In CARPDD formulation, if we set $e_{ij}=0\ \forall(v_i,v_j)\in E$, then any CARP instance can be transformed into CARPDD instance. Since, CARP is an NP-hard problem [1], CARPDD is also NP-hard. □

Mathematical model of the CARPDD as follows:
Decision variables

$$y_{ijk}=\begin{cases}1 & \text{if vehicle } k \text{ services edge } (v_i,v_j)\\ 0 & \text{otherwise}\end{cases}$$

$$x_{ijk}=\begin{cases}1 & \text{if edge } (v_i,v_j) \text{ is traversed from } v_i \text{ to } v_j \text{ by the } k\text{th vehicle}\\ 0 & \text{otherwise}\end{cases}$$

Model

$$\min\left(\sum_{(v_i,v_j)\in E}\sum_{k=1}^{K} c_{ij}x_{ijk}\right) \tag{1}$$

s.t.

$$\sum_{(v_j,v_i)\in E} x_{jik}-\sum_{(v_i,v_j)\in E} x_{ijk}=0 \quad \forall v_i\in V \quad \forall k \tag{2}$$

$$\sum_{k=1}^{K} y_{ijk} = 1 \quad \forall (v_i,v_j) \in R, \quad i<j \tag{3}$$

$$\sum_{(v_i,v_j)\in E} e_{ij} x_{ijk} + \sum_{\substack{(v_i,v_j)\in R \\ i<j}} q_{ij} y_{ijk} \le Q \quad \forall k \tag{4}$$

$$x_{ijk} + x_{jik} \ge y_{ijk} \quad \forall (v_i,v_j) \in R \quad \forall k \tag{5}$$

$$n^2 \sum_{v_i \notin S, v_j \in S} x_{ijk} \ge \sum_{v_i \in S, v_j \in S} x_{ijk} \left\} \begin{array}{l} V' = V \backslash v_0 \\ \forall S \subseteq V', S \ne \emptyset \\ \forall k \end{array} \right. \tag{6}$$

$$y_{ijk} \in \{0,1\} \quad \forall (v_i,v_j) \in R, \quad i<j \quad \forall k \tag{7}$$

$$x_{ijk} \in \{0,1\} \quad \forall (v_i,v_j) \in E \quad \forall k \tag{8}$$

Eq. (1) is the objective function of the model that minimizes the total distance of all vehicle tours. Eq. (2) is the flow conservation constraints for network-flow formulations. Eq. (3) ensures that all required edges should be serviced exactly once. Eq. (4) is the capacity constraint, total deadheading and servicing demand of each vehicle cannot exceed the vehicle capacity Q. Eq. (5) ensures that if the vehicle covers edge (v_i,v_j), it has to traverse the edge v_i to v_j or v_j to v_i. Eq. (6) guarantees that each trip is connected to depot vertex v_0 which is stated in [20]. Eqs. (7) and (8) are the integrality constraints.

Eq. (6) consists of $K(2^{(n-1)}-1)$ number of constraints. For that reason, the model is unsolvable for the large-sized graphs. Therefore, in the following section a heuristic algorithm is proposed to solve the CARPDD for larger graphs.

4. Modified Ulusoy algorithm for the CARPDD

Since the CARP is an NP-hard problem, several constructive heuristics have been proposed in the literature. These heuristics are separated into two main classes which are simple constructive heuristics and two-phase constructive heuristics. Two-phase constructive heuristics, which divide the procedure into routing and clustering, consist of route first-cluster second and cluster first route second algorithms.

In this study, a heuristic algorithm is proposed to solve new CARP by modifying Ulusoy partitioning algorithm. Ulusoy-partitioning heuristic is a two-phase constructive algorithm and one of the route first-cluster second algorithms [10]. Modified Ulusoy algorithm gets a graph, required edge set, an Euler tour that covers all required edges as inputs and generates feasible vehicle tours for CARPDD. Modified Ulusoy algorithm generates a solution into two main steps. In the first step, the algorithm constructs a directed graph where each arc of the graph represents a feasible vehicle tour. In the second step, a shortest path problem is solved on the directed graph from source vertex to terminal vertex. By using result of the shortest path problem, minimum cost feasible vehicle tours that cover all required edges are determined. Since, the modified Ulusoy algorithm gets and Eulerian tour as an input, in the following subsection tour Eulerian tour construction methods are explained. Thereafter, the details of the modified Ulusoy algorithm is presented.

4.1. Constructing Euler tour

Given a connected graph, a tour that visits every edge in the graph exactly once is called Euler tour. Euler proved that an Euler tour exists if all vertices have even degree. Finding the minimum-cost tour that traverses each edge in an undirected graph is called undirected Chinese postman problem (CPP) [21]. The CPP can be solved in polynomial time by using minimum-cost perfect matching algorithm [17]. Steps of the CPP solution algorithm is presented in Algorithm 1.

In the following sections, several algorithms are defined and explained. These algorithms incur some basic calculations, which are shortest path, length calculation and total load calculation. Shortest path function represented with SP (v_i,v_j). This function determines the shortest path between the vertices v_i and v_j. Length calculation function represented with L(X). This function calculates the length of the X where X could be a complete tour, cycle, tour partition or path. Final procedure is total load calculation procedure, which is represented with TotalLoad(T). This function takes a tour T as an input, and calculates total deadheading and servicing demand on T.

Algorithm 1. AlgorithmCPP(G).

```
Input: Undirected connected graph $G=(V,E)$
Output: A minimum cost CPP tour $T$ that cover all edges on $G$
1  if all vertices are even-degree then
2      Go to Step 12
3  else
4      Determine the set of odd-degree vertices $V_o$ on $G$
5      Construct a complete graph $G_o$ with vertex set $V_o$
6      $c^o_{ij} \leftarrow \mathrm{L}(\mathrm{SP}(v^o_i,v^o_j))\ \forall (v^o_i,v^o_j \in V_o)$
7      $M \leftarrow \mathrm{MPM}(G_o)$
8      foreach $((v^o_i,v^o_j) \in M)$ do
9          Add all edges on the $\mathrm{SP}(v^o_i,v^o_j) \in G$
10     end
11 end
12 $T \leftarrow$ EulerianCycle($G$)   /* $G$ is an Eulerian graph */
13 return $T$
```

As seen in Algorithm 1, if the input graph is Eulerian, the CPP tour on G is determined by using the Euler tour (EulerianCycle(G)) algorithm. Hierholzer's algorithm was used to obtain an Euler tour where the complexity of the algorithm is $O(m)$ [22]. On the other hand, the input graph may not be Eulerian, if so the graph has to be transformed into Eulerian form by using minimum cost perfect matching (MPM(G)) algorithm. In this study, Blossom V algorithm was used to determine minimum cost perfect matching [23]. Blossom V is the most recent algorithm to compute the minimum cost perfect matching, and the worst complexity is $O(n^2m)$. This implies that the worst case complexity of Algorithm 1 is $O(n^2m)$.

Instead of visiting all edges in the graph, it may be necessary to service only a subset of all edges called required edges (R). This problem is called as rural postman problem (RPP) [24]. Unlike the CPP, the RPP is an NP-hard problem [25].

Definitions for the RPP are as follows. The set of vertices in G incident to at least one edge in R is denoted V_R. A partial graph that is induced from R is called required subgraph and represented as $G_R=(V_R,R)$. Since G may contain many non-required vertices, the RPP is solved on a simplified graph $G_S=(V_R,R\cup E_S)$ derived from G_R as follows. An edge (v_i,v_j) is included in E_S for each $v_i,v_j\in V_R$. The cost of an added edge (v_i,v_j) in E_S is equal to length of the $SP(v_i,v_j)$. The set of added edges (E_S) is then reduced by eliminating edges that satisfy following two cases [2]:

- All edges $(v_i,v_j)\in E_S$ for which $c_{ij}=c_{ik}+c_{kj}$ for some $v_k\in V_R$.
- One of two parallel edges if they have the same cost.

An RPP tour T is then determined on G_S, and a tour on G is obtained by replacing each non-required edge on T by its corresponding shortest path in G. In RPP solution methods, we assumed that G has already been simplified with the mentioned procedure. The RPP can be solved in polynomial time by means of the CPP solution approach if the graph (G) is directed or undirected and required subgraph (G_R) is connected. Since the input graph is undirected, if G_R is connected, then the optimal RPP tour can be obtained by using Algorithm 1. On the other hand, G_R may be disconnected, in that case an algorithm that was proposed by Frederickson can be used to construct the RPP tour [26]. If the cost matrix of the G satisfies the triangle inequality, Frederickson's algorithm has a worst case performance ratio of 3/2. Steps of the RPP solution algorithm is given in Algorithm 2.

Algorithm 2. AlgorithmRPP(G,R).

```
Input: An undirected connected graph $G=(V,E)$, a required edge set $R$.
Output: RPP tour $T$ that cover all edges on $G$.
1  if ($G_R$ connected) then
2  |  Go to Step 12
3  else
4  |  Determine the connected components of $G_R$ as $C_1,\dots,C_c$
5  |  Construct a complete graph $G'$ with vertex set $V'=\{1,\dots,c\}$
6  |  $c_{pq}\leftarrow$ SP($C_p,C_q$) where $v_p,v_q\in V'$ and SP($C_p,C_q$) $\in G$
7  |  $F\leftarrow$ MST($G'$)
8  |  foreach (($v_p,v_q$) $\in F$) do
9  |  |  Connect the $C_p$ and $C_q$ in $G$ by using SP($v_p,v_q$)
10 |  end
11 end
12 $T\leftarrow$ AlgorithmCPP($G$)
13 return $T$
```

Mainly, Frederickson's heuristic (Steps 4–10 in Algorithm 2) determines the connected components of G_R where the number of components (c) is strictly less than number of vertices ($c<n$). Since, we use Floyd's algorithm [27] to obtain distance matrix, it takes $O(n^3)$ time to construct G'. Then, connected components are combined by using minimum spanning tree algorithm (MST). In this study, Prim's algorithm is used to determine the minimum spanning tree where the complexity is $O(nm)$ [27]. After connecting the components, G_R turns into connected graph, RPP tour can be obtained by using CPP solution algorithm. The worst case complexity of CPP solution algorithm is $O(n^2m)$, and since $m\geq n$ the complexity of Algorithm 2 is also $O(n^2m)$. It should be noted that, if G_R does not contain the v_0, the output tour T of the Algorithm 2 may not include v_0. If V_R does not contain v_0 ($v_0\notin V_R$), then v_0 is added to V_R ($V_R=V_R\cup\{v_0\}$) to prevent this circumstance.

Frederickson's algorithm does not prove the optimality, so in some cases an unnecessary long tour can be obtained depending on the instance. Hertz et al. proposed an improvement procedure called Shorten algorithm, to decrease the length of the RPP tour [28]. Shorten algorithm gets a tour T as an input. Although, some edges on T are required edge, it may be non-serviced edge on T. This case occurs when some of the required edges traversed more than once. If an edge $(v_i,v_j)\in R$ is traversed more than once, we assume that (v_i,v_j) is serviced when its first occur on T. Consider the following RPP tour T where $(v_i,v_j)\in R$, $T=\{\dots,v_i,v_j,\dots,v_j,v_i,\dots\}$. Required edge (v_i,v_j) is repeated twice on T, so that (v_i,v_j) is a serviced edge, but (v_j,v_i) is a non-serviced edge on T. Set of serviced edges are represented with S. Steps of the Shorten algorithm is given in Algorithm 3.

Algorithm 3. Shorten (G,R,T).

```
Input: An undirected connected graph $G=(V,E)$, a required edge set $R$, the RPP tour $T$.
Output: Possibly better RPP tour $T$ on $G$.
```

```
1  foreach ($v_i$ and orientation on $T$) do
2    foreach ($v_j$ along the orientation on $T$) do
3      $P_1 \leftarrow (v_i,\ldots,v_j)$ on $T$
4      $P_2 \leftarrow (v_j,\ldots,v_i)$ on $T$ where $T \leftarrow \{v_i,\ldots,v_j,\ldots,v_i\}$
5      if ($P_2$ covers $R$) then
6        if ($(v_l,v_j) \in S\ \forall v_l \in P_2$) then
7          if ($\mathtt{L}(\mathtt{SP}(v_i,v_j)) < \mathtt{L}(P_1)$) then
8            $T \leftarrow \mathtt{SP}(v_i,v_j) \cup P_2$
9          end
10       else
11         Let $(v_k,v_j) \notin S$ on $P_2$
12         Induce a cycle where $C \leftarrow (v_j,\ldots,v_k,v_j)$ on $P_2$
13         $C \leftarrow (v_j,v_k,\ldots,v_j)$
               /* Reverse the orientation of $C$ */
14         $T \leftarrow P_1 \cup C \cup (v_j,\ldots,v_i)$
15       end
16     end
17   end
18 end
19 return $T$
```

Mainly, Shorten algorithm attempts to identify shorter tour including v_0 and covering the required edges. As seen in Algorithm 3, Shorten tries to collect non-serviced edges in P_1 while gathering serviced edges in P_2. Then, tour may be improved by replacing P_1 with a shortest path between the end points of P_1 [4].

Shorten algorithm, which is given in Algorithm 3, is slightly different from the original version of the algorithm with respect to termination criteria. In the original version of the algorithm, which is proposed in [28], algorithm calls itself as long as cost is decreased. So that, original version of the Shorten is a pseudo-polynomial algorithm. Instead of searching for a non-improving solution, we terminate the procedure after examining all possible tour partitions, so we are able to find a polynomial bound for the Shorten. The complexity of the Shorten algorithm depends on the all possible tour partitions along the given orientation which is equal to number of edges on the input tour. For any graph G, when each edge of the graph is duplicated, it is possible to find an Eulerian tour that covers E. So, number of edges on an Eulerian tour ($|T|$) can be at most $2m$ ($|T| \leq 2m$). Since $R \subseteq E$, this bound satisfies the RPP case. Consider a graph with one edge ($m=1$) with vertices 1 and 2. The Eulerian tour on this graph is $T=\{(1,2),(2,1)\}$. Since, the number of edges on the Eulerian tour is equal to $2m$, the upper bound for the $|T|$ is tight. The $|T|$ effects the computational cost of two outer loops in Algorithm 3, which is $O(4m^2)$. Inside these loops control operations take $O(m^2)$ time. So, the overall complexity of Shorten is $O(4m^4)$.

In conclusion, three different cases occur when constructing an Eulerian tour that covers given set of required edges (R) on undirected graph G. If the case is $R=E$, so the tour can be obtained optimally by using CPP solution algorithm. Second case is $R \subset E$ and required subgraph (G_R) is connected, in this case the tour can obtain RPP solution algorithm, optimally. Finally, $R \subset E$ and required subgraph (G_R) is unconnected, in this case Frederickson's heuristic can be used to determine the tour that is not optimal. The Shorten algorithm can be used to improve the tour. These cases are presented in Algorithm 4, which constructs an Eulerian tour.

Algorithm 4. `ConstructTour`(G,R).

```
Input: An undirected connected graph $G=(V,E)$, a required edge set $R$.
Output: Eulerian tour $T$ on $G$ that covers $R$.
1 if ($R=E$) then
2   $T \leftarrow \mathtt{AlgorithmCPP}(G)$
3 else
4   $T \leftarrow \mathtt{AlgorithmRPP}(G,R)$
5   if ($G_R$ connected) then
        /* Solution is optimal */
6   else
7   $T \leftarrow \mathtt{Shorten}(G,R,T)$
8   end
9 end
```

4.2. Modified Ulusoy algorithm for the CARPDD

In the Ulusoy-partitioning heuristic, firstly a giant Eulerian tour is obtained that covers all required edges, then this tour is partitioned into several feasible vehicle tours [4,10]. In this study, Ulusoy-partitioning algorithm is modified to solve CARPDD. Unlike Ulusoy-partitioning algorithm, modified Ulusoy algorithm considers deadheading demands. The inputs of the algorithm are undirected graph G, required edge set R, Eulerian tour T that covers R, and vehicle capacity Q. In this study, input tour is obtained by using CPP and RPP solution methods, which have already been mentioned in the previous subsection.

After obtaining the giant tour (T), tour is labeled v_1 to v_r where r is the largest index of a vertex incident to a serviced edge on T. Then, a directed graph $G' = (V',A)$ is constructed. Vertex set of G' is $V' = \{v_1, v_2, \ldots, v_r\}$ and arc set of G' is $A = \emptyset$, initially. Iteratively every possible tour partition between v_1 and v_r are transformed into a vehicle tour. If the obtained vehicle tour is feasible in terms of capacity then corresponding tour partition is included to A. During the tour construction phase, two possible cases can be occurred. Tour partition either contains the depot vertex (v_0) or does not contain v_0. Depending on these two cases, tours are constructed, differently. After all possible tour partitions are examined, a directed graph $G' = (V',A)$ is obtained. Then, shortest path is determined between v_1 and v_r on G' that corresponds to minimum total travel cost among possible alternatives. Each arc on the shortest path represents a feasible vehicle tour (T_k) on G. Additionally, tours guarantee that all required edges (R) are covered. The detailed steps of the modified Ulusoy algorithm is given in Algorithm 5.

Algorithm 5. Modified Ulusoy algorithm.

```
Input: An undirected connected graph $G=(V,E)$, a required edge set $R$, Eulerian tour $T$
Output: Feasible vehicle tours $T_k$, $k=1,\ldots,m$
1  $T \leftarrow (v_1, v_2, \ldots, v_t = v_1)$   /* Re-label the vertices in $G$ */
2  $r \leftarrow$ Largest index of a vertex incident to a serviced edge on $T$
3  Construct a directed graph $G'=(V',A)$ where $V'=\{v_1,v_2,\ldots,v_r\}$ and $A=\emptyset$
4  foreach (($v_a,v_b$) where $a,b = 1,2,\ldots,r$ and $a<b$) do
5      $A \leftarrow A \cup (v_a,v_b)$
6  end
7  foreach (($v_a,v_b$) where $a,b = 1,2,\ldots,r$ and $b>a+1$) do
8      if (($v_a,v_{a+1}) \notin S$ or $(v_{b-1},v_b) \notin S$) then
9          $A \leftarrow A \setminus (v_a,v_b)$
10     end
11 end
12 foreach (($v_a,v_b) \in A$) do
13     if ($b=a+1$ and $(v_a,v_{a+1}) \notin S$) then
14         $d'_{ab} \leftarrow 0$
15     end
16     if ($b>a+1$ or $(v_a,v_{a+1}) \in S$) then
17         if ($P_{ab}$ contains $v_0$) then
               /* $P_{ab}=\{v_a,\ldots,v_0,\ldots,v_b\}$ */
18             Add SP($v_a,v_b$) to $P_{ab}$
19             $P_{ab} \leftarrow \{v_0,\ldots,v_a,\mathrm{SP}(v_a,v_b),v_b,\ldots,v_0\}$
20         else
               /* $P_{ab}=\{v_a,\ldots,v_b\}$ */
21             Add SP($v_0,v_a$) to $P_{ab}$
22             Add SP($v_b,v_0$) to $P_{ab}$
23             $P_{ab} \leftarrow \{\mathrm{SP}(v_0,v_a),v_a,\ldots,\mathrm{SP}(v_b,v_0)\}$
24         end
25     end
26     $load \leftarrow$ TotalLoad($P_{ab}$)
27     if ($load \le Q$) then
28         $d'_{ab} \leftarrow$ L($P_{ab}$)
29     else
30         $A \leftarrow A \setminus (v_a,v_b)$
31     end
32 end
33 $Path \leftarrow$ SP($v_1,v_r$)
34 $k \leftarrow 0$
35 foreach (($v_a,v_b$) on $Path$) do
36     $k \leftarrow k+1$
37     $C \leftarrow$ Apply Step 17−24
38     $T_k \leftarrow C$
39 end
40 $K \leftarrow k$   /* Number of vehicles usage */
41 return $T_k$, $k=1,\ldots,K$
```

Modified Ulusoy algorithm partitions the input tour and constructs feasible vehicle tours. In the algorithm, each feasible tour is represented with an arc on the directed graph $G' = (V',A)$. Shortest path between source and terminal vertices on G' guarantees servicing all required edges. Moreover, each arc, that lies on the shortest path, shows a feasible vehicle tour.

The main difference between the modified Ulusoy algorithm and Ulusoy-partitioning heuristic is consideration of the deadheading demands. Inclusion of the deadheading demands turns both the problem and the algorithm into much more complex form. Ulusoy-partitioning algorithm is not necessary to construct the tour before controlling the capacity violation for any tour partition (v_a,v_b) on T, because the serviced edges on (v_a,v_b) and corresponding demands have already known. On the other hand, modified Ulusoy algorithm has no initial information related to the deadheading load for (v_a,v_b), because deadheading load can be calculable after obtaining the tour. So that, in Algorithm 5, firstly we obtain the tour, then we calculate the total load on (v_a,v_b).

Complexity of the modified Ulusoy algorithm also depends on the number of edges on the input tour as in Shorten algorithm. The cost of examining each tour partition is $O(4m^2)$. For each tour partition, firstly a tour is constructed by using shortest path information with $O(n)$, then total load on the tour is determined with $O(m)$. After the directed graph is obtained, shortest path is determined on this graph with $O(4m^2)$ complexity. So, overall complexity of the modified Ulusoy algorithm is $O(4m^2(m+n)+4m^2)$, this implies that the worst case complexity is $O(m^3)$.

The objective function of the CARPDD is the minimization of total travel cost, which is mentioned in Section 3. Unlike total travel cost different objective functions can be defined for the CARPDD. Ulusoy-partitioning algorithm may also incur the vehicle assignment cost together with travel cost [10]. After constructing the directed graph (G'), where each arc of G' represents a feasible vehicle tour. It is possible to add fixed vehicle assignment cost to each arc of G' for a homogeneous fleet.

5. Post-optimization methods

Modified Ulusoy algorithm is a two-phase constructive method, and its performance is highly correlated with the input Eulerian tour. Therefore, the obtained tours may be unnecessarily long. To avoid this circumstance, two post-optimization procedures are proposed. These algorithms get the same inputs of modified Ulusoy algorithm, including vehicle tours. The result of the post-optimization procedures is possibly better vehicle tours.

First post-optimization algorithm tries to improve each tour individually by using CPP and RPP solution approaches. Assume that obtained tour for the k-th vehicle by using modified Ulusoy algorithm is as follows, $T_k = v_0,SP_{01},v_1,v_2,v_3,SP_{30},v_0$ where serviced edges from this vehicle are $R_k = \{(v_1,v_2),(v_2,v_3)\}$. Another tour ($T'$) can be obtained by using Algorithm 4 that covers R_k and $L(T'_k) < L(T_k)$. First post-optimization procedure searches for a better tour for each vehicle by using Euler tour construction algorithms. Steps of the first post-optimization algorithm is given in Algorithm 6.

Algorithm 6. First post-optimization procedure.

```
    Input: An undirected connected graph $G=(V,E)$, a required edge set $R$, feasible vehicle tours $T_k$, $k=1,\dots,K$.
    Output: Possibly better vehicle tours $T_k$, $k=1,\dots,K$ on $G$.
 1  for $k \leftarrow 1$ to $K$ do
 2      Determine the set of serviced edges from $k$th vehicle $R_k$
 3      $T'_k \leftarrow$ ConstructTour($G,R_k$)
 4      $Q' \leftarrow$ TotalLoad($T'_k$)
 5      if $(Q' \le Q)$ then
 6          if $(\mathrm{L}(T'_k) < \mathrm{L}(T_k))$ then
 7              $T_k \leftarrow T'_k$
 8          end
 9      end
10  end
11  return $T_k$, $k=1,\dots,K$
```

First post-optimization algorithm determines serviced edges (R_k) on tour T_k where T_k is obtained by using modified Ulusoy algorithm, then constructs an Eulerian tour T'_k that covers R_k by using Algorithm 4. If the total load on T'_k does not exceed vehicle's capacity, and the tour is improved (length of T'_k is strictly less than the length of T_k), T_k is replaced with T'_k. Since the tour construction procedure is called K times, the complexity of the first post-optimization procedure is $O(Kh)$ where h represents the complexity of the tour construction algorithm.

Second post-optimization procedure is based on transferring serviced edges between vehicles. This post-optimization procedure is an insertion type improvement heuristic. Suppose that, list of edges that are serviced on tour k are R_k where $k=1,\dots,K$. The second post-optimization procedure, removes the edge ($(v_i,v_j) \in R_k$) and reassigns to R_l where $l=1,2,\dots,K$ and $l \ne k$. Then, new tours are constructed for tour k and l by using Algorithm 4. If the total lengths of the new tours are improved, then the new tours are approved. As the procedure iteratively removes an edge and reassigns it to another vehicle, some of the tours may not contain any serviced edge. So that, it is possible to reduce not only total distance but also the number of tours that serve R. The steps of the second post-optimization are given in Algorithm 7.

Algorithm 7. Second post-optimization procedure.

```
    Input: An undirected connected graph $G=(V,E)$, a required edge set $R$, feasible vehicle tours $T_k$, $k=1,\dots,K$.
    Output: Possibly better vehicle tours $T_k$, $k=1,\dots,K$ on $G$.
```

```
$length \leftarrow 0$
for $k \leftarrow 1$ to $K$ do
    $length \leftarrow length + \mathrm{L}(T_k)$
    /* Calculate total tour length */
    Determine the set of serviced edges from $k$th vehicle ($R_k$)
end
for ($k \leftarrow 1$ to $K$) do
    foreach $(v_i,v_j) \in R_k$ do
        $R'_k \leftarrow R_k \backslash (v_i,v_j)$
        $T'_k \leftarrow \mathtt{ConstructTour}(G,R'_k)$
        $Q'_k \leftarrow \mathtt{TotalLoad}(T'_k)$
        foreach $l \in \{1,\dots,K\}$ where $l \neq k$ do
            $R'_l \leftarrow R_l \cup (v_i,v_j)$
            $T'_l \leftarrow \mathtt{ConstructTour}(G,R'_l)$
            $Q'_l \leftarrow \mathtt{TotalLoad}(T'_l)$
            if ($Q'_k \le Q$) and ($Q'_l \le Q$) then
                $length' \leftarrow length - \mathrm{L}(T_k) - \mathrm{L}(T_l) + \mathrm{L}(T'_k) + \mathrm{L}(T'_l)$
                if ($length' < length$) then
                    $R_k \leftarrow R_k \backslash (v_i,v_j)$
                    $R_l \leftarrow R_l \cup (v_i,v_j)$

                end
            end
        end
    end
end
return $T_k$, $k = 1,\dots,K$
```

Second post-optimization procedure determines serviced edges (R_k) by vehicle-k, then removes a serviced edge from the vehicle-k and assigns this edge to other vehicles one by one. Thereafter, Eulerian tours are constructed for both vehicles with updated required edge sets by using Algorithm 4. If the total load of both vehicles does not exceed the vehicle capacity (Q), and the total tour length is improved, transferring the service edge and new tours are applied. In this algorithm, required edge assignment is realized for all possible vehicle pairs and $m \ge |R|$ for every graph, so the worst case complexity of the second post-optimization procedure is $O(mK^2h)$ where h is the complexity of the tour construction algorithm.

To sum up, in Sections 4 and 5 the proposed method, which is used to solve CARPDD, is explained in detail. Initially, Eulerian tour construction method is presented that incorporates CPP, RPP solution methods and improvement method named Shorten. Then, modified Ulusoy algorithm is explained that gets the Eulerian tour as an input and generates feasible vehicle tours. Finally, two post-optimization procedures are mentioned that search for better vehicle tours with respect to total tour length without violating capacity constraint. Moreover, worst case complexities of all these algorithms are given, which are bounded with a polynomial function. Since, all mentioned algorithms work in polynomial time, the proposed method is polynomial.

6. Illustrative example

In this section, an illustrative example is presented to clarify the steps of the modified Ulusoy algorithm and post-optimization procedures. Sample undirected graph ($G=(V,E)$) for the illustrative example is given in Fig. 1 where $n=9$ and $m=12$. Required edges and non-required edges are represented by solid line and dashed line in G, respectively. Vertex 1 is the depot ($v_0=1$) of the graph. Edge costs ($c_{ij}\ \forall (v_i,v_j) \in E$) are given in Fig. 1. Edge demands are equal to the edge costs ($q_{ij}\ \forall (v_i,v_j) \in R$), and deadheading demands are 1 ($e_{ij}=1\ \forall (v_i,v_j) \in E$). Fleet is homogeneous, and the capacity of each vehicle is $Q=40$.

Firstly, a giant tour is obtained by using Algorithm 4. As seen in Fig. 1, G_R is a disconnected graph. So, Eulerian tour T that covers R is obtained by using Frederickson's algorithm (T=1,8,2,6,7,9,3,5,7,5,3,8,1,4,1). Then, giant tour is labeled v_1 to v_r where $r=14$ for this example. Giant tour and corresponding labels are given below.

1	8	2	6	7	9	3	5	7	5	3	8	1	4
v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	v_{13}	v_{14}

After labeling the giant tour, it is necessary to obtain $G'=(V',A)$ where each $(v_i,v_j) \in A$ is a feasible vehicle tour. For the given example, possible tours are given in Table 1. In this table, first column represents the arcs of the G'. These arcs correspond to possible partitions of

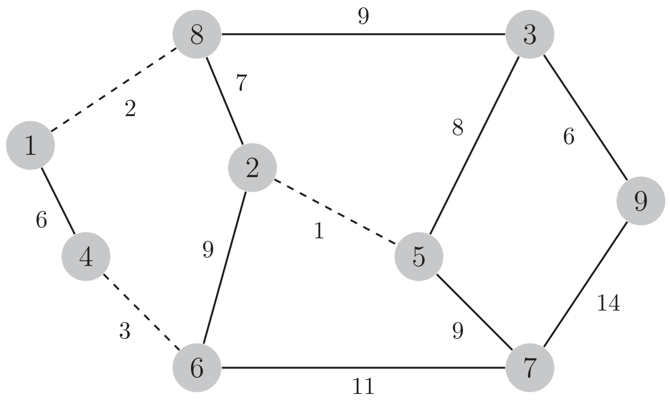

Fig. 1. Original graph *G*.

Table 1
Tour partitions on *G*.

$(v_i,v_j)\in A$	Tour partition	Serviced edges	Tour	Deadheading load	Servicing load	Total load	Cost
(v_2,v_3)	8–2	(2–8)	1–8–2–8–1	4	7	11	18
(v_2,v_4)	8–2–6	(2–6), (2–8)	1–8–2–6–4–1	5	16	21	27
(v_2,v_5)	8–2–6–7	(2–6), (2–8), (6–7)	1–8–2–6–7–5–2–8–1	8	27	35	48
(v_3,v_4)	2–6	(2–6)	1–8–2–6–4–1	5	9	14	27
(v_3,v_5)	2–6–7	(2–6), (6–7)	1–8–2–6–7–5–2–8–1	8	20	28	48
(v_4,v_5)	6–7	(6–7)	1–4–6–7–5–2–8–1	7	11	18	39
(v_4,v_6)	6–7–9	(6–7), (7–9)	1–4–6–7–9–3–8–1	7	25	32	51
(v_4,v_7)	6–7–9–3	(3–9), (6–7), (7–9)	1–4–6–7–9–3–8–1	7	31	38	51
(v_5,v_6)	7–9	(7–9)	1–8–2–5–7–9–3–8–1	8	14	22	50
(v_5,v_7)	7–9–3	(3–9), (7–9)	1–8–2–5–7–9–3–8–1	8	20	28	50
(v_5,v_8)	7–9–3–5	(3–5), (3–9), (7–9)	1–8–2–5–7–9–3–5–2–8–1	10	28	38	57
(v_6,v_7)	9–3	(3–9)	1–8–3–9–3–8–1	6	6	12	34
(v_6,v_8)	9–3–5	(3–5), (3–9)	1–8–3–9–3–5–2–8–1	8	14	22	41
(v_6,v_9)	9–3–5–7	(3–5), (3–9), (5–7)	1–8–3–9–3–5–7–5–2–8–1	10	23	33	59
(v_7,v_8)	3–5	(3–5)	1–8–3–5–2–8–1	6	8	14	29
(v_7,v_9)	3–5–7	(3–5), (5–7)	1–8–3–5–7–5–2–8–1	8	17	25	47
(v_7,v_{12})	3–5–7–5–3–8	(3–5), (3–8), (5–7)	1–8–3–5–7–5–3–8–1	8	26	34	56
(v_8,v_9)	5–7	(5–7)	1–8–2–5–7–5–2–8–1	8	9	17	38
(v_8,v_{12})	5–7–5–3–8	(3–8), (5–7)	1–8–2–5–7–5–3–8–1	8	18	26	47
(v_8,v_{14})	5–7–5–3–8–1–4	(1–4), (3–8), (5–7)	1–4–6–2–5–7–5–3–8–1	9	24	33	56
(v_{11},v_{12})	3–8	(3–8)	1–8–3–8–1	4	9	13	22
(v_{11},v_{14})	3–8–1–4	(3–8), (1–4)	1–4–3–8–1	4	17	21	34
(v_{13},v_{14})	1–4	(1–4)	1–4–1	2	6	8	12

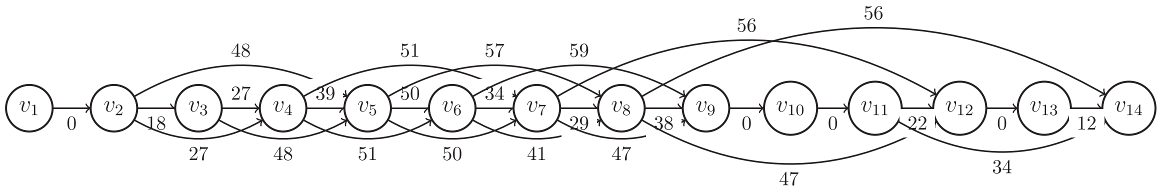

Fig. 2. Directed graph G' that contains feasible vehicle tours.

T, which are given in second column. For each tour partition, some of the edges are servicing some of them deadheading. Third column shows the set of serviced edges. Vehicle tour for each arc in G' is given in fourth column. Columns 5, 6, 7 show deadheading, servicing and total load on the tour partition, respectively. Finally, last column of the table represents the total cost of the tour. $G'=(V',A)$ is also shown in Fig. 2 with arc costs ($d_{ij}\ \forall(v_i,v_j)\in A$).

Shortest path on G' between vertices v_1 and v_{14}, that gives the minimum total cost feasible tours that cover *R*, is as follows: v_1, v_2, v_4, v_7, v_{12}, v_{13}, v_{14}. On this shortest path, only arcs (v_2-v_4), (v_4-v_7), (v_7-v_{12}) and $(v_{13}-v_{14})$ contain serviced edges. So that, it is sufficient to write tours that correspond to these arcs. Tours are as follows where the total cost of all tours is 146.

- Vehicle-1: 1–8–2–6–4–1 (serviced edges: (2–6) and (2–8)).
- Vehicle-2: 1–4–6–7–9–3–8–1 (serviced edges: (3–9), (6–7) and (7–9)).
- Vehicle-3: 1–8–3–5–7–5–3–8–1 (serviced edges: (3–5), (3–8), (5–7)).
- Vehicle-4: 1–4–1 (serviced edge: (1–4)).

After obtaining vehicle tours by using modified Ulusoy algorithm, we apply two post-optimization procedures to these results. Results of the first post-optimization procedure are given below. Total cost after first post-optimization procedure is 137.

- Vehicle-1: 1–4–6–2–8–1 (serviced edges: (2–6)) and (2–8)).
- Vehicle-2: 1–8–3–9–7–6–4–1 (serviced edges: (3–9), (6–7) and (7–9)).
- Vehicle-3: 1–8–3–5–7–5–2–8–1 (serviced edges: (3–5), (3–8), (5–7)).
- Vehicle-4: 1–4–1 (serviced edge: (1–4)).

First post-optimization procedure decreases total cost from 146 to 137. This improvement occurs due to third vehicle tour. Cost of this tour is decreased by using edge (2–5) which is a non-required edge instead of edge (3–5). Thereafter, second post-optimization procedure is applied to the result of the first improvement algorithm and obtained tours are as follows:

- Vehicle-1: 1–4–6–2–8–1 (serviced edges: (2–6) and (2–8)).
- Vehicle-2: 1–8–3–9–7–6–4–1 (serviced edges: (3–9), (6–7) and (7–9)).
- Vehicle-3: 1–4–6–7–5–3–8–1 (serviced edges: (1–4), (3–5), (3–8) and (5–7)).

Second post-optimization procedure decreases the total cost 137 to 126. More importantly, for this example together with total cost, total number of vehicle usage is decreased. This result is obtained by transferring the edge (1–4) from fourth vehicle to third vehicle. Before the second post-optimization procedure, the edge (1–4) was serviced in a separate tour. This procedure assigns the edge to another tour without capacity constraint violation.

7. Computational results

All experiments presented here were performed on an Intel Core2 Duo 2.2 GHz with 2 Gb RAM. All algorithms were implemented in C++. Mathematical models were solved by using GAMS/CPLEX 12.1 [29].

7.1. Test problems

In the literature, three sets of well-known CARP test instances have been introduced so far. The first set (*gdb* tests) contains 23 instances generated by DeArmon [30]. The size of these test instances is in between 7 and 27 vertices, and 11–55 edges all of which are required. The second set (*val* tests) is from Benavent et al. [31] containing 34 instances defined on 10 different graphs to evaluate cutting plane algorithm; for each graph, different instances were created by changing the capacity of the vehicles. Similar to *gdb* test instances, all edges are required in *val* test problems. These instances consist of 24–50 vertices and 34–92 edges. The final set (*eglese* tests) was generated by Eglese based on data from a winter gritting application in Lancashire [3]. There are 24 instances based on two graphs where the different instances have been created by changing the set of required edges and the capacities of the vehicles. Because of the application from which these instances were generated, the demand quantities for the required edges are proportional to their costs. The sizes of these problems are in the range of 77–140 vertices, 98–190 edges and 51–190 required edges. All three sets of test instances can be obtained from http://www.uv.es/~belengue/carp.html.

Unlike the classical CARP, CARPDD considers the deadheading demands, so that deadheading demands are included to the CARP test instances. Deadheading demands (e_{ij}) are obtained for all test instances as follows:

gdb	All edges are required ($R=E$). So, deadheading demands are determined as follows: $e_{ij}=q_{ij}\ \forall(v_i,v_j)\in R$.
val	All edges are required ($R=E$). So, deadheading demands are determined as follows: $e_{ij}=q_{ij}\ \forall(v_i,v_j)\in R$.
eglese	All edges are not required ($R\subset E$) and $q_{ij}=c_{ij}\ \forall(v_i,v_j)\in R$. In a similar manner, deadheading demands are determined as follows: $e_{ij}=c_{ij}\ \forall(v_i,v_j)\in E$.

Adding deadheading demands to CARP test problems makes them CARPDD instances. However, inclusion of deadheading demands turns the instances into capacity infeasible. So, new-vehicle capacities should be determined. Firstly, all test instances are solved by using modified Ulusoy partitioning algorithm with increasing capacity of the vehicle until a feasible solution is obtained. The obtained capacity parameters for the test problems are the minimum vehicle capacity to generate a feasible solution. Since, it is impossible to determine the total deadheading load on G before obtaining the vehicle tours. We use a regression model to predict the total load on G. Total load (dependent variable of the regression model) is affected from the number of edges ($|E|$), the number of required edges ($|R|$), the sum of all deadheading edge demands ($\sum_{(v_i,v_j)\in E} e_{ij}$) and the sum of all servicing edge demands ($\sum_{(v_i,v_j)\in R} q_{ij}$). In the regression model, these four independent variables not only have individual effect but also integrated effect on the dependent variable. Integrated effects are $|E|-\sum_{(v_i,v_j)\in E} e_{ij}$ and $|R|-\sum_{(v_i,v_j)\in R} q_{ij}$. Model parameters are as follows: a^1 represents the number of edges, a^2 is the number of required edges, a^3 shows the sum of all deadheading edge demands and a^4 shows the sum of all servicing edge demands. Decision variables (x_i, $i=1,\ldots,7$) are the coefficients of independent variables.

$$\min\left(\sum_j (y_j^{obs}-y_j^{exp})^2\right) \tag{9}$$

s.t.

$$\sum_{i=1}^{4} a_j^i x_i + \sum_{i=5}^{6} a_j^{(i-4)} a_j^{(i-2)} x_i + x_7 = y_j^{exp} \quad \forall j \tag{10}$$

$$y_j^{exp} \geq 0 \tag{11}$$

Eq. (9) is the objective function of the mathematical model that minimizes the sum of squared errors. Eq. (10) calculates predicted capacity values. In the left-hand side of this equation, first term is for the individual effects, second term is for the integrated effects, and final term is the constant of the regression model. Eq. (11) guarantees that predicted capacities has to be non-negative. Test problem names and corresponding parameters are presented in Table 2.

In Table 2 not the whole "val" and "egl" instances are presented, because while identifying the capacity only the graph and the required edge set are considered. For each group of instances, the vehicle capacity and the number of vehicles in fleet parameters are changing, which have no effect on the regression model. In this table, minimum necessary capacity to obtain a feasible solution by using modified Ulusoy algorithm (Q_{obs}) and required number of vehicles (K) are presented. QK is chosen as an independent variable instead of Q, because we consider total load (deadheading and servicing demand) on all tours.

The regression model, which is a quadratic programming model, is solved by using GAMS/CPLEX 12.1 [29]. Predicted capacity values (Q_{pre}) are obtained by dividing the regression model results to the number of vehicles (K), which were determined before. Goodness-of-fit (R^2) of the model is obtained 96.36% that shows the model is accurate enough to predict the vehicle capacities. Capacity differences between test problems that are in a same group are determined by increasing vehicle capacities and decreasing number of vehicles with the same ratio of original CARP instances. All modified test instances can be obtained from http://home.ku.edu.tr/~gkirlik/routing.htm〉.

7.2. Test results

In this subsection, all test instances, that are generated by modifying CARP test instances, are solved with the mathematical model and the modified Ulusoy algorithm. The results of the algorithm are improved by using two post-optimization procedures. Mathematical model has 10 000 s time limit.

Firstly, "gdb" test instances are solved. In Table 3, first column represents the name of the test instance. Columns between 2 and 6 are the test problem parameters, which are the number of vertices in graph (n), the number edges (m), the number of required edges ($|R|$), vehicle capacities (Q) and the number of available vehicles (K), respectively. Column 7 represents the total number of vehicle usage when

Table 2
Test problem parameters.

File	$\|E\|$	$\|R\|$	$\sum_{(v_i,v_j)\in E} e_{ij}$	$\sum_{(v_i,v_j)\in R} q_{ij}$	Q_{obs}	K	QK	Q_{pre}
gdb1	22	22	22	22	9	11	99	26
gdb2	26	26	26	26	9	11	99	11
gdb3	22	22	22	22	9	10	90	29
gdb4	19	19	19	19	7	13	91	32
gdb5	26	26	26	26	9	13	117	9
gdb6	22	22	22	22	8	10	80	29
gdb7	22	22	22	22	7	11	77	26
gdb8	46	46	249	249	73	16	1168	120
gdb9	51	51	258	258	67	19	1273	93
gdb10	25	25	37	37	11	12	132	27
gdb11	45	45	224	224	51	17	867	97
gdb12	23	23	212	212	80	11	880	248
gdb13	28	28	245	245	68	15	1020	192
gdb14	21	21	89	89	26	12	312	102
gdb15	21	21	112	112	38	10	380	153
gdb16	28	28	116	116	30	13	390	92
gdb17	28	28	168	168	42	13	546	144
gdb18	36	36	153	153	38	12	456	102
gdb19	11	11	66	66	45	6	270	246
gdb20	22	22	107	107	41	11	451	128
gdb21	33	33	154	154	42	12	504	117
gdb22	44	44	205	205	30	24	720	61
gdb23	55	55	266	266	34	30	1020	55
val1	39	39	358	358	106	19	2014	197
val2	34	34	310	310	135	16	2160	212
val3	35	35	137	137	64	13	832	83
val4	69	69	627	627	151	24	3624	230
val5	65	65	614	614	162	22	3564	254
val6	50	50	451	451	151	16	2416	270
val7	66	66	559	559	98	32	3136	151
val8	63	63	566	566	123	27	3321	188
val9	92	92	654	654	116	30	3480	152
val10	92	92	704	704	144	36	5184	144
egl-e1	98	51	2453	1468	863	16	13 808	906
egl-e2	98	72	2453	1879	863	21	18 123	1004
egl-e3	98	87	2453	2188	874	29	25 346	874
egl-e4	98	98	2453	2453	874	31	27 094	880
egl-s1	190	75	4186	1394	959	23	22 057	959
egl-s2	190	147	4186	3174	1236	27	33 372	1236
egl-s3	190	159	4186	3379	1397	25	34 925	1541
egl-s4	190	190	4186	4186	1228	35	42 980	1228

the tours are obtained by using modified Ulusoy algorithm (Ulusoy K). Similarly, next column represents the vehicle usage after the two improvement procedures (Imp. K) applied.

The total tour lengths of the modified Ulusoy algorithm (Ulusoy alg.) are given in column 9. First (Post opt-1) and second (Post opt-2) improvement procedure results are given in columns 10 and 11, respectively. It is noted that, each procedure is used to improve previous algorithm's result. Percentage improvements (Imp.%) after the post-optimization procedures are given in column 12. The percentage improvements are calculated by using (12). Solution times of the modified Ulusoy algorithm (Ulusoy time), first post-optimization (Opt-1 time) procedure, second post-optimization (Opt-2 time) procedure and the total solution time (Total time) are given in columns 13–16.

$$Imp.\% = \frac{\text{UlusoyAlg} - \text{PostOpt2}}{\text{PostOpt2}} \tag{12}$$

Mathematical model results are presented in columns 17–21. Objective value of the best obtained integer feasible solution (Model feas.) is given in column 17. Lower bound value (Model LB) for each problem is given in column 18. Relative difference of the second post-optimization result from the incumbent solution value (Inc. %) and the lower bound value (LB%) is given in columns 19 and 20, respectively. (Inc.%) and (LB%) values are calculated by using (13) where REF corresponds to either incumbent solution's objective value or lower bound. Finally, the solution time of the model is given in column 21.

$$Diff.\% = \frac{\text{PostOpt2} - \text{REF}}{\text{REF}} \tag{13}$$

In Table 3, results of the "gdb" test instances are presented. Effect of the post-optimization procedures can be shown with the total tour length improvement and the number of vehicle usage reduction. The average improvement in "gdb" instances is 7.33%, and the number of vehicle usage is decreased in "gdb5" and "gdb9" instances. Another important remark is the solution time of the whole procedure, which is less than 1 s in all "gdb" instances.

The sizes of the "gdb" test problems are small compared to other instances, so they can be solved optimally by using the mathematical model. As seen from Table 3, most of the model solutions are interrupted due to the time limit. This shows the hardness of the CARPDD even in small-sized instances. Average difference between incumbent solution of the model and heuristics is 1.41%. Although, best obtained integer solution values are slightly better than heuristic objective values, most of instance's solution time is 10 000 s while total solution time of the heuristic approach is less than 1 s.

Another argument to check the solution quality of the mentioned heuristic algorithms is comparing with the lower bounds. These bounds are determined by using the given mathematical model. However, the instances named "gdb8", "gdb9" and "gdb11" are problematic due to the number of vertices ($n \geq 15$) in the graph. Since the number of subtour elimination constraints in the mathematical model is exponential with n, these instances could not be solved. This set of constraints is removed from the model, and relaxed formulation is solved for these instances. Although, the new formulation is a relaxation of the original model, it is still terminated due to the time limit. The solutions after post-optimization procedures are compared with the lower bounds, and the average percentage difference is 17.76%. The average of heuristic results can be seen far from lower bounds, but this situation occurs due to loose lower bounds.

To show the looseness of the lower bounds in Table 3, "gdb1", "gdb6" and "gdb7" instances are chosen to be solved at Koç University high-performance computing laboratory 6-day long. In Table 3, deviations of the heuristic solutions from the lower bounds are reported as 16.67%, 7.31% and 17.94%, respectively. These instances are solved by using the mathematical model in a different computation

Table 3
Results of "gdb" test instances.

File	n	m	$\|R\|$	Q	K	Ulusoy K	Imp. K	Ulusoy alg.	Post opt-1	Post opt-2	Imp. (%)	Ulusoy time	Opt-1 time	Opt-2 time	Total time	Model feas.	Model LB	Inc. (%)	LB (%)	Model time
gdb1	12	22	22	26	11	3	3	294	294	294	0.00	0	0	0.02	0.02	294	252	0.00	16.67	10 000
gdb2	12	26	26	11	11	9	9	413	407	406	1.72	0	0	0.02	0.02	360	291	12.77	39.52	10 000
gdb3	12	22	22	29	10	3	3	316	299	291	8.59	0	0	0.01	0.01	259	233.08	12.36	24.85	10 000
gdb4	11	19	19	32	13	2	2	331	321	266	24.44	0	0	0.01	0.01	266	251.67	0.00	5.69	10 000
gdb5	13	26	26	9	13	13	10	693	663	571	21.37	0	0	0.03	0.03	575	316	−0.70	80.70	10 000
gdb6	12	22	22	29	10	3	3	290	285	279	3.94	0	0	0.01	0.01	279	260	0.00	7.31	10 000
gdb7	12	22	22	26	11	3	3	319	319	309	3.24	0	0	0.01	0.01	304	262	1.64	17.94	10 000
gdb8	27	46	46	120	16	7	7	433	359	323	34.06	0.01	0	0.09	0.11	–	221	–	46.15	10 000
gdb9	27	51	51	93	19	11	10	430	388	335	28.36	0.01	0	0.11	0.13	–	219	–	52.97	10 000
gdb10	12	25	25	27	12	4	4	314	304	295	6.44	0	0	0.02	0.02	275	252	7.27	17.06	10 000
gdb11	22	45	45	97	17	8	8	564	528	473	19.24	0	0	0.11	0.11	–	358	–	32.12	10 000
gdb12	13	23	23	248	11	3	3	438	424	410	6.83	0.01	0	0	0.02	384	336	6.77	22.02	10 000
gdb13	10	28	28	192	15	4	4	548	536	536	2.24	0	0	0.02	0.02	520	514	3.08	4.28	10 000
gdb14	7	21	21	102	12	3	3	96	96	96	0.00	0	0	0	0	96	96	0.00	0.00	0.22
gdb15	7	21	21	153	10	2	2	56	56	56	0.00	0	0	0.01	0.01	56	56	0.00	0.00	0.03
gdb16	8	28	28	92	13	4	4	131	131	127	3.15	0	0	0.01	0.01	125	119	1.60	6.72	10 000
gdb17	8	28	28	144	13	3	3	93	91	91	2.20	0.01	0	0	0.02	91	85	0.00	7.06	10 000
gdb18	9	36	36	102	12	4	4	166	166	164	1.22	0	0	0.02	0.02	158	158	3.80	3.80	0.88
gdb19	8	11	11	246	6	1	1	55	55	55	0.00	0	0	0	0	55	55	0.00	0.00	0.08
gdb20	11	22	22	128	11	3	3	121	121	121	0.00	0	0.01	0	0.01	121	114	0.00	6.14	10 000
gdb21	11	33	33	117	12	4	4	154	154	154	0.00	0	0	0.02	0.02	154	150	0.00	2.67	10 000
gdb22	11	44	44	61	24	10	10	207	207	204	1.47	0	0	0.05	0.05	226	191	−9.73	6.81	10 000
gdb23	11	55	55	55	30	14	14	241	241	241	0.00	0	0	0.08	0.08	270	223	−10.74	8.07	10 000
Average											7.33	0.00	0.00	0.03	0.03			1.41	17.76	8260.92

Table 4
Results of the “val” test instances.

File	n	m	$\|R\|$	Q	K	Ulusoy K	Imp. K	Ulusoy alg.	Post opt-1	Post opt-2	Imp. (%)	Ulusoy time	Opt-1 time	Opt-1 time	Total time
val1A	24	39	39	876	5	2	2	229	215	194	18.04	0	0	0.04	0.04
val1B	24	39	39	526	8	2	2	235	215	197	19.29	0.01	0	0.03	0.05
val1C	24	39	39	197	19	7	6	305	283	223	36.77	0	0	0.07	0.07
val2A	24	34	34	954	4	1	1	286	286	286	0.00	0	0	0.01	0.01
val2B	24	34	34	636	6	3	3	316	298	298	6.04	0	0	0.07	0.07
val2C	24	34	34	212	16	7	6	520	465	393	32.32	0	0	0.09	0.09
val3A	24	35	35	332	4	2	2	98	98	93	5.38	0.01	0	0.04	0.06
val3B	24	35	35	208	6	2	2	110	109	95	15.79	0	0	0.08	0.08
val3C	24	35	35	83	13	8	6	174	157	137	27.01	0	0	0.06	0.06
val4A	41	69	69	690	8	3	3	578	547	474	21.94	0.01	0.01	0.61	0.64
val4B	41	69	69	552	11	5	5	612	561	530	15.47	0.01	0	0.43	0.45
val4C	41	69	69	399	14	6	6	632	581	537	17.69	0.02	0	0.21	0.25
val4D	41	69	69	230	24	12	10	844	780	649	30.05	0.01	0	0.3	0.32
val5A	34	65	65	746	8	3	3	597	569	507	17.75	0.01	0.01	0.38	0.41
val5B	34	65	65	559	10	4	4	621	619	503	23.46	0.01	0	0.35	0.37
val5C	34	65	65	441	13	6	5	653	629	516	26.55	0.01	0	0.36	0.38
val5D	34	65	65	254	22	10	9	841	783	623	34.99	0.01	0	0.27	0.29
val6A	31	50	50	918	5	2	2	278	278	278	0.00	0.01	0	0.05	0.07
val6B	31	50	50	648	7	2	2	278	278	278	0.00	0.01	0	0.05	0.07
val6C	31	50	50	270	16	6	6	346	317	279	24.01	0	0	0.16	0.16
val7A	40	66	66	465	11	5	5	418	403	349	19.77	0.02	0	0.3	0.34
val7B	40	66	66	349	14	5	5	419	408	340	23.24	0.01	0	0.37	0.39
val7C	40	66	66	151	32	16	15	540	510	455	18.68	0.01	0	0.21	0.23
val8A	30	63	63	579	9	3	3	519	494	442	17.42	0.01	0	0.24	0.26
val8B	30	63	63	434	12	5	5	547	507	481	13.72	0	0.01	0.23	0.24
val8C	30	63	63	188	27	13	12	825	762	700	17.86	0.01	0	0.21	0.23
val9A	50	92	92	511	9	5	4	464	451	418	11.00	0.03	0	0.77	0.83
val9B	50	92	92	380	12	6	6	482	461	422	14.22	0.02	0	0.6	0.64
val9C	50	92	92	304	15	7	7	520	506	437	18.99	0.02	0	0.58	0.62
val9D	50	92	92	152	30	20	16	735	665	574	28.05	0.01	0.01	0.58	0.61
val10A	50	92	92	480	11	5	5	606	568	564	7.45	0.02	0.01	0.5	0.55
val10B	50	92	92	365	15	7	7	644	603	574	12.20	0.02	0	0.53	0.57
val10C	50	92	92	288	18	10	10	700	639	605	15.70	0.02	0	0.53	0.57
val10D	50	92	92	144	36	36	25	1345	1274	952	41.28	0.02	0	0.68	0.72
Average											18.59	0.01	0.00	0.29	0.32

environment with a longer period of time. At the termination, although the optimal solutions cannot be attained, lower bound values are increased while incumbent solutions are still same. As a result, better bounds are obtained compared to former results, which are 5.38%, 4.49% and 6.02%, respectively. Since, the lower bounds are increased, and incumbent solutions remain same for these instances, it can be interpreted as the gaps between heuristic solutions and lower bounds are much tighter than in Table 3.

Results of the “val” test instance are given in Table 4. These test problems are medium and large-sized instances. Since the sizes of the “val” instances are larger than “gdb” instances, the effects of the improvement procedures can be seen more clearly. Average improvement by using post-optimization procedures is 18.59% which is more than two fold of “gdb” instances. Besides, in the 11 of the instances total number of vehicle usage is decreased owing to second post-optimization procedure. Another point related to post-optimization procedures is the second post-optimization procedure is more beneficial than the first one. The improvement amount of the first post-optimization procedure (Ulusoy alg.–Post opt-1) is 5.69% while the second one (Post opt-1–Post opt-2) is 12.16%. This result is not an unexpected, because the second post-optimization searches for a large-sized neighborhood of the current solution where the current solution is Ulusoy result for the first post-optimization and first post-optimization procedure result for the second post-optimization procedure. As in “gdb” instances, solution times of the instances are less than 1 s.

In Table 5, results of the “eglese” test instances are presented. All “eglese” test problems are the large-sized instances, so the importance of the post-optimization procedures is especially observed in these instances. The average improvement of the post-optimization procedures is 63.22% which is much larger than both “val” and “gdb” instances. Moreover, in almost all instances number of vehicle usage is decreased. Improvement amounts of the first and second post-optimization procedures are 26.18% and 29.24%, respectively.

In “eglese” instances solution time of the whole procedure is increased considerably compared to former instance types, because of the second post-optimization procedure. In the average, 87.74% of the total time are passed during this procedure due to large neighborhood size. Since, the number of required edges is increased, number of neighbors of the current solution becomes too much. So that, searching in a large neighborhood is time consuming. However, it should be noted that solution times of all instances are less than 15 s.

8. Conclusions

In this paper, a new CARP variation was introduced, which incurs deadheading edge demands differently from classical CARP. This problem is observed in multi-robot sensor-based coverage problem where a mobile-robot consumes energy while servicing and traversing during the coverage task.

Table 5
Results of the "eglese" test instances.

File	n	m	$\|R\|$	Q	K	Ulusoy K	Imp. K	Ulusoy alg.	Post opt-1	Post opt-2	Imp. (%)	Ulusoy time	Opt-1 time	Opt-1 time	Total time
egl-e1-A	77	98	51	1728	16	5	5	5560	5119	4636	19.93	0.03	0	0.35	0.41
egl-e1-B	77	98	51	1246	16	8	7	6800	5588	4862	39.86	0.03	0	0.23	0.29
egl-e1-C	77	98	51	906	16	14	12	9748	8795	7152	36.30	0.03	0	0.37	0.43
egl-e2-A	77	98	72	2008	21	6	6	8375	7037	5821	43.88	0.05	0.01	0.79	0.9
egl-e2-B	77	98	72	1435	21	10	8	8805	8019	6692	31.58	0.06	0	1.23	1.35
egl-e2-C	77	98	72	1004	21	15	13	10 191	9913	8298	22.81	0.05	0.01	0.62	0.73
egl-e3-A	77	98	87	1813	29	9	9	11 460	9021	7714	48.56	0.06	0.01	1.33	1.46
egl-e3-B	77	98	87	1231	29	14	12	13 217	10 563	8318	58.90	0.06	0.01	1.56	1.69
egl-e3-C	77	98	87	874	29	29	21	18 878	18 238	13 716	37.63	0.06	0.01	1.25	1.38
egl-e4-A	77	98	98	1896	31	10	8	12 158	9578	7628	59.39	0.07	0.01	2.08	2.23
egl-e4-B	77	98	98	1219	31	17	14	14 858	12 370	9736	52.61	0.07	0	1.26	1.4
egl-e4-C	77	98	98	880	31	30	20	19 357	18 853	13 405	44.40	0.07	0	1.18	1.32
egl-s1-A	140	190	75	1956	23	6	5	9330	6829	5114	82.44	0.14	0	1.32	1.6
egl-s1-B	140	190	75	1397	23	13	8	12 193	9758	6709	81.74	0.14	0	0.85	1.13
egl-s1-C	140	190	75	959	23	23	16	17 075	16 429	11 472	48.84	0.13	0.01	0.92	1.19
egl-s2-A	140	190	147	2421	27	12	11	19 976	12 821	9774	104.38	0.37	0.01	8.81	9.56
egl-s2-B	140	190	147	1648	27	21	16	23 876	17 587	12 795	86.60	0.37	0	6.74	7.48
egl-s2-C	140	190	147	1236	27	27	19	25 414	20 047	14 550	74.67	0.37	0.01	4.23	4.98
egl-s3-A	140	190	159	3082	25	9	8	19 181	12 983	9606	99.68	0.41	0.01	13.94	14.77
egl-s3-B	140	190	159	2055	25	15	13	22 692	15 080	11 441	98.34	0.42	0.01	7.7	8.55
egl-s3-C	140	190	159	1541	25	22	17	23 916	18 120	13 351	79.13	0.42	0	7.53	8.37
egl-s4-A	140	190	190	2354	35	14	14	22 622	14 116	11 717	93.07	0.52	0	12.97	14.01
egl-s4-B	140	190	190	1638	35	22	18	25 228	17 339	13 758	83.37	0.51	0.01	6.52	7.55
egl-s4-C	140	190	190	1228	35	35	21	31 235	23 980	16 515	89.13	0.52	0.01	8.46	9.51
Average											63.22	0.21	0.01	3.84	4.26

The new CARP variation, which is called CARPDD, is the generalized form of the CARP. A mathematical model was presented for the problem. CARPDD is an NP-hard problem, so it is computationally infeasible to solve large-size instances optimally. In this study, one of the two-phase constructive (route first cluster second) heuristics, Ulusoy-partitioning algorithm, was modified to solve CARPDD instances. Modified Ulusoy algorithm is a constructive type heuristic, so it may generate unnecessary long tours. So that, two post-optimization procedures were used to improve the result of the modified Ulusoy algorithm. Additionally, computational complexity of all mentioned algorithms were presented, and we showed that the whole procedure works in polynomial time.

The performance of the proposed mathematical model, modified heuristic and post-optimization procedures were tested by using test problems, which were obtained by adding deadheading edge demands to CARP test instances. Tests showed that some of the "gdb" instances (small-sized) can be solved optimally by using the mathematical model. Some of the instances were not solved optimally due to the time limit. For these instances, lower bounds were obtained by relaxing subtour elimination constraints into the model. Another point about the test results is the benefit of post-optimization procedures. Average improvements after the post-optimization procedures for "gdb", "val" and "eglese" instances were 7.33%, 18.59% and 63.22%, respectively. As a future work, we will obtain tighter lower bounds for the modified "gdb" instances, and we will generate lower bounds for the modified "val" and "eglese" instances.

References

[1] Golden BL, Wong RT. Capacitated arc routing problems. Networks 1981;11(3):305–15.
[2] Hertz A. Recent trends in arc routing. In: Golumbic MC, Hartman IB-A, editors. Graph theory, combinatorics and algorithms, vol. 34. MA, USA: Kluwer Academic Publishers; 2005. p. 215–36 [Chapter 9].
[3] Eglese RW. Routing winter gritting vehicles. Discrete Applied Mathematics 1994;48(3):231–44.
[4] Hertz A, Mittaz M. Heuristic algorithms. In: Dror M, editor. Arc routing: theory, solutions and applications. MA, USA: Kluwer Academic Publishers; 2000. p. 327–86. [Chapter 9].
[5] Brandão J, Eglese R. A deterministic tabu search algorithm for the capacitated arc routing problem. Computers & Operations Research 2008;35(4):1112–26.
[6] Lacomme P, Prins C, Ramdane-Cherif W. Competitive memetic algorithms for arc routing problems. Annals of Operations Research 2004;131:159–85.
[7] Polacek M, Doerner KF, Hartl RF, Maniezzo V. A variable neighborhood search for the capacitated arc routing problem with intermediate facilities. Journal of Heuristics 2008;14(5):405–23.
[8] Ghiani G, Laganá D, Laporte G, Mari F. Ant colony optimization for the arc routing problem with intermediate facilities under capacity and length restrictions. Journal of Heuristics 2010;16(2):211–33.
[9] Wøhlk S. A decade of capacitated arc routing. In: Golden B, Raghavan S, Wasil E, editors. The vehicle routing problem: latest advances and new challenges. New York, USA: Springer; 2008. p. 29–48.
[10] Ulusoy G. The fleet size and mix problem for capacitated arc routing. European Journal of Operational Research 1985;22(3):329–37.
[11] Choset H. Coverage for robotics—a survey of recent results. Annals of Mathematics and Artificial Intelligence 2001;31(1):113–26.
[12] Prassler E, Ritter A, Schaeffer C, Fiorini P. A short history of cleaning robots. Autonomous Robots 2000;9(3):211–26.
[13] Arkin EM, Fekete SP, Mitchell JSB. Approximation algorithms for lawn mowing and milling. Computational Geometry Theory and Applications 2000;17(1–2):25–50.
[14] Trevai C, Ota J, Arai T. Multiple mobile robot surveillance in unknown environments. Advanced Robotics 2007;21(7):729–49.
[15] Acar EU, Choset H, Lee JY. Sensor-based coverage with extended range detectors. IEEE Transactions on Robotics 2006;22(1):189–98.
[16] Choset H. Sensor based motion planning: the hierarchical generalized voronoi graph. PhD thesis. Pasadena, CA, USA: California Institute of Technology; 1996.
[17] Edmonds J, Johnson EL. Matching Euler, tours and the Chinese postman. Mathematical Programming 1973;5(1):88–124.
[18] M.B. Dias. Traderbots: a new paradigm for robust and efficient multirobot coordination in dynamic environments. PhD thesis. Pittsburgh, PA, USA: Carnegie Mellon University; 2004.
[19] Sipahioglu A, Kirlik G, Parlaktuna O, Yazici A. Energy constrained multi-robot sensor-based coverage path planning using capacitated arc routing approach. Robotics and Autonomous Systems 2010;58(5):529–38.
[20] Assad AA, Golden BL. Arc routing methods and applications. In: Ball M, Magnanti T, Monma C, Nemhauser G, editors. Network routing. Handbooks in operations research and management science, vol. 8. Elsevier; 1995. p. 375–483.
[21] Eiselt HA, Gendreau M, Laporte G. Arc routing problems. Part I. The Chinese postman problem. Operations Research 1995;43(2):231–42.

[22] Gross JL, Yellen J. Handbook of graph theory.CRC Press; 2004.
[23] Kolmogorov V. Blossom V: a new implementation of a minimum cost perfect matching algorithm. Mathematical Programming Computation 2009;1(1):43–67.
[24] Eiselt HA, Gendreau M, Laporte G. Arc routing problems. Part II. The rural postman problem. Operations Research 1995;43(3):399–414.
[25] Lenstra JK, Rinnoy Kan AHG. On general routing problems. Networks 1974;6(3):273–80.
[26] Frederickson GN. Approximation algorithms for some postman problems. Journal of the ACM (JACM) 1979;26(3):538–54.
[27] Ahuja R, Magnanti T, Orlin J. Network flows: theory algorithms and applications. Prentice Hall; 1993.
[28] Hertz A, Laporte G, Hugo PN. Improvement procedures for the undirected rural postman problem. INFORMS Journal on Computing 1999;11(1):53–62.
[29] IBM ILOG CPLEX 12.1 reference manual. URL ⟨http://www.ilog.com⟩; 2010.
[30] DeArmon J. A comparison of heuristics for the capacitated chinese postman problem. Master's thesis. MD, USA: University of Maryland; 1981.
[31] Belenguer JM, Benavent E. A cutting plane algorithm for the capacitated arc routing problem. Computers & Operations Research 2003;30(5):705–28.