Computers and Operations Research 103 (2019) 288–295

Contents lists available at ScienceDirect

Computers and Operations Research

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

An efficiency-based path-scanning heuristic for the capacitated arc routing problem

Rafael Kendy Arakaki*, Fábio Luiz Usberti

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

ARTICLE INFO

Article history:
Received 30 April 2018
Revised 8 October 2018
Accepted 26 November 2018
Available online 27 November 2018

Keywords:
Capacitated arc routing problem
Constructive heuristic
Path-scanning

ABSTRACT

The capacitated arc routing problem (CARP) is an important combinatorial optimization problem that has been extensively studied in the last decades. The objective is to optimize routes that service demands located on the edges of a graph, given a fleet of homogeneous vehicles with limited capacity that starts and ends its routes at a specific node (depot). This work proposes a new path-scanning heuristic for the CARP which introduces the concept of efficiency rule. Given the current vehicle location, its traversed distance and the amount of serviced demand, the efficiency rule selects the most promising edges to service next. Computational experiments conducted on a set of benchmark instances reveal that the proposed heuristic substantially outperformed all previous path-scanning heuristics from literature.

1. Introduction

Given scattered demands in the edges of a network that must be serviced by a fleet of identical vehicles with limited capacity all gathered in a central depot. How to service these demands within the shortest distance and without exhausting the vehicles capacities? This question is addressed by the *capacitated arc routing problem* (CARP) proposed by Golden and Wong (1981). Since then CARP attracted the attention of many researchers mainly due to its optimization difficulty and numerous applications.

The CARP has been extensively studied and we refer the reader to references (Belenguer et al., 2014; Mourão and Pinto, 2017; Prins, 2014) for a comprehensive survey. Applications of CARP include but are not restricted to waste collection (Ghiani et al., 2014), meter reading (Eglese et al., 2014) and snow plows (Liu et al., 2014). Moreover, many real-world complex operations encouraged the emergence of CARP variants, where additional constraints and special characteristics are modeled. Examples of CARP variants include *CARP with time windows* (Afsar, 2010; Vansteenwegen et al., 2010), *periodic CARP* (Zhang et al., 2017), *open CARP* (Arakaki and Usberti, 2018; Usberti et al., 2011), and others (Muyldermans and Pang, 2014). The improvement in methods for CARP can often be extended to these variants.

CARP is an NP-hard problem for which many exact and heuristic methodologies have been proposed (Mourão and Pinto, 2017). Optimal solutions have been reported only for relatively small instances involving around 100 edges with positive demand (Bode and Irnich, 2014), while the best results for larger instances are provided by heuristic methods (Chen et al., 2016).

Constructive heuristics start building a solution from scratch and progressively include elements into the solution until feasibility is obtained. The first constructive heuristic proposed for CARP was the *construct-strike* presented by Golden and Wong in the seminal paper (Golden and Wong, 1981). Two years later, Golden et al. (1982) proposed two other constructive heuristics called *path-scanning* and *augment-merge*. Since then, constructive heuristics with better performance have been proposed, such as *improved merge* (Belenguer et al., 2006) and *path-scanning with ellipse rule* (Santos et al., 2009).

Our contribution. This work proposes a constructive heuristic called *path-scanning with efficiency rule* (PS-Efficiency) for CARP, which extends the ideas of the *path-scanning with ellipse rule* (PS-Ellipse) proposed by Santos et al. (2009). In PS-Ellipse, an ellipse rule is triggered when the current vehicle capacity is below a given static threshold. Once activated, the rule restricts the next candidate edges to be within the ellipse containing the depot and the current node as focal points. The PS-Efficiency improves these ideas by: (i) proposing a dynamic threshold for activating the rule; (ii) the set of candidate edges are selected based on a route efficiency index with respect to the ratio of demand over distance.

Computational experiments conducted on a set of benchmark instances were performed to compare the proposed method to other path-scanning heuristics from literature. Experiments reveal that PS-Efficiency substantially outperformed PS-Ellipse (Santos et al., 2009).

* Corresponding author.
E-mail address: rafael.arakaki@ic.unicamp.br (R. Kendy Arakaki).

https://doi.org/10.1016/j.cor.2018.11.018
0305-0548/

This paper is organized as follows. The CARP is formally defined in Section 2. Section 3 presents a review of constructive heuristics for CARP. Section 4 describes the PS-Efficiency heuristic. Section 5 shows the computational experiments and the performance analysis of the methodologies. Section 6 provides the final remarks.

2. Problem definition

The Capacitated Arc Routing Problem (CARP) (Golden and Wong, 1981) can be formally defined as follows. Let $G(V, E)$ be an undirected graph with a non-negative cost or length $c(v_i, v_j)$ and a non-negative demand $d(v_i, v_j)$ assigned to each edge $(v_i, v_j) \in E$. Let $E_R \subseteq E$ be the set of edges with positive demand, called *required edges*. A fleet of identical vehicles with limited capacity D is used to service all required edges. While traversing an edge (v_i, v_j), a vehicle might (i) service (v_i, v_j), which deducts its capacity by $d(v_i, v_j)$ and increases the solution cost by $c(v_i, v_j)$ or (ii) deadhead (v_i, v_j), which only increases the solution cost by $c(v_i, v_j)$.

A route is a trajectory of a vehicle in G, represented by the sequence of traversed edges, which starts and finishes at a distinguished node v_0 called *depot*. A set of routes is defined feasible if the sum of demands serviced in each route does not exceed D and each required edge is serviced by exactly one vehicle. The objective is to find a feasible set of routes with minimum cost.

3. Constructive heuristics for CARP

The literature of CARP heuristics can be generally classified in three major categories: (i) constructive heuristics, which start from an empty solution and iteratively grow it into a feasible solution; (ii) local search methods, that start from a feasible solution and attempt to improve it by exploring a neighbourhood; (iii) metaheuristics, which usually comprise a combination of constructive heuristics, local search methods and likely some additional optimization techniques, such as short and long-term memory, evolutionary framework, diversification and intensification. Constructive heuristics are usually simple and fast. On the other hand, metaheuristics have higher complexity and require more processing time while often providing the best solutions. The decision of which method is more suitable for a given application depends on the available computing time and the characteristics of the instances.

Constructive heuristics remain an important area of research for CARP. As described by Santos et al. (2009), there are several reasons for investigating these heuristics: (i) fast processing time, which is crucial for many real-time applications; (ii) warm starting solutions for metaheuristics; (iii) less fine-tuning of parameters; (iv) more easily adapted to the numerous CARP variants. For example, PS-Ellipse was used to construct initial solutions for at least three metaheuristics (Santos et al., 2010; Shang et al., 2016; Usberti et al., 2013) and inspired heuristics for at least four variants of CARP (Bosco et al., 2013; Eydi and Javazi, 2012; Grandinetti et al., 2012; Usberti et al., 2011).

The first constructive heuristics for CARP have been proposed by Golden and Wong (1981) and Golden et al. (1982): *path-scanning* (PS), *augment-merge* (AM) and *construct-strike* (CS). The CS heuristic have not attracted much attention (except for *modified-CS* from Pearn (1989)), possibly because of its complexity. On the other hand, PS and AM have been extensively used as benchmark heuristics for CARP and have given rise to new improved methods based on their classical versions. Special attention will be given to the review of path-scanning heuristics in Section 3.1.

Belenguer et al. (2006) proposed the *improved merge* (IM) heuristic based on AM, achieving very good results for large instances (more than 100 required edges) with fairly little computational cost. Ulusoy (1984) proposed an innovative *route-first cluster-second* method where first a giant tour including all required edges is built which is then partitioned into feasible tours by a *Split* algorithm. Later, Prins et al. (2009) proposed improving methods for Ulusoy's algorithm. Other constructive heuristics for CARP include *parallel-insert* (Chapleau et al., 1984), *augment-insert* (Pearn, 1991) and the four heuristics proposed by Wøhlk (2005) in her thesis: *modified path-scanning, double outer scan, node duplication heuristic* and *A-ALG*.

3.1. Path-scanning heuristics

The original *path-scanning* (PS) proposed by Golden et al. (1982) is a greedy heuristic that starts with the vehicle at the depot node, then progressively services the nearest unserviced edge. The vehicle stops and returns to the depot when the remaining vehicle capacity is not sufficient to service more edges. Then, a new vehicle is employed to service the remaining edges and the process is repeated. Often there are multiple required edges with minimum distance. In this case the heuristic employs five different criteria to decide which unserviced edge $(v_i, v_j) \in E_R$ will be serviced next: (1) minimize $c(v_i, v_j)/d(v_i, v_j)$; (2) maximize $c(v_i, v_j)/d(v_i, v_j)$; (3) minimize cost of node v_j back to depot node; (4) maximize cost of node v_j back to depot node; (5) use criterion 4 if vehicle has more than half capacity, otherwise use criterion 3. An instance is solved five times, using the five different criteria, then the best solution is returned. Using the five different criteria often generates five very different solutions, therefore increasing the chance of finding a good solution.

A modified version of PS was proposed by Pearn (1989) by selecting one of the original five criteria at random as tie-breaking for each time multiple unserviced edges with minimum distance are identified. This heuristic will be called *path-scanning with random criteria* (PS-RC) from now on. One of the main advantages of PS-RC is to produce much more than five different solutions because of its stochastic behavior. Results with number of executions $k = 30$ were published and obtained better performance than the original path-scanning for all instances experimented.

Later, Belenguer et al. (2006) proposed another modified version of *path-scanning*, here called *path-scanning with random edge* (PS-RE), which simplifies the tie-breaking decision by just randomly selecting one of the tied edges. The authors compared the performance of PS-RE and PS-RC on benchmark instances and found the performance of both heuristics for $k = 20$ and $k = 50$ to be fairly similar.

Finally, Santos et al. (2009) proposed the *path-scanning with ellipse rule* (PS-Ellipse) that works similarly to PS-RE but also considers an "ellipse rule". This rule triggers when the vehicle remaining capacity is low enough and restricts the vehicle to service only a subset of edges that are within an ellipse. The PS-Ellipse produced far better solutions than its precedents PS-RE and PS-RC with a minor increase in computational time. A detailed description of PS-Ellipse is provided next.

Let $ned = |E_R|$ be the number of required edges, $td = \sum_{(v_i,v_j)\in E_R} d(v_i, v_j)$ be the total demand, $tc = \sum_{(v_i,v_j)\in E_R} c(v_i, v_j)$ be the total cost of required edges, α a real parameter, (v_g, v_h) the last serviced edge in the route, $SP(v_p, v_q)$ the shortest path cost from node v_p to node v_q, rvc the remaining vehicle capacity and v_0 the depot node. The heuristic starts with an empty vehicle at the depot and iteratively inserts in the route the closest unserviced edge (v_i, v_j) that does not violate the rvc. Tie-breaking rule selects one of the tied edges at random. The *ellipse rule* is activated right after rvc satisfies the *triggering criteria* (1). After the *ellipse rule* is triggered, the vehicle can only service edges that are inside the ellipse with focal points v_h and v_0. An edge $(v_i, v_j) \in E_R$ is inside the ellipse if in equation (2) holds. The vehicle returns to the depot when its

capacity is insufficient to service any edge inside the ellipse. A new empty vehicle starts at depot node and the process is repeated until all required edges are serviced.

$$rvc \leq \alpha \times td/ned \tag{1}$$

$$SP(v_h, v_i) + c(v_i, v_j) + SP(v_j, v_0) \leq tc/ned + SP(v_h, v_0) \tag{2}$$

The rationale behind PS-Ellipse is that if the remaining vehicle capacity is low then only edges that are close to the path from the current node to the depot should be considered for servicing. The authors performed computational experiments comparing PS-RC, PS-RE and PS-Ellipse on a set of benchmark instances. The results revealed that PS-Ellipse reduced the average deviation from lower bounds by about 44% in comparison to PS-RC and PS-RE.

4. Path-scanning with efficiency rule

The *path-scanning with efficiency rule* (PS-Efficiency) is a heuristic based on the *path-scanning with ellipse rule* (PS-Ellipse) proposed by Santos et al. (2009) described in Section 3.1. It constructs a set of feasible routes in a greedy fashion, where each route is created starting from the depot and then sequentially selecting the nearest unserviced edge $(v_i, v_j) \in E_R$ for which the demand $d(v_i, v_j)$ does not exceed the remaining vehicle capacity. Similarly to PS-Ellipse, when the vehicle reaches a certain load threshold it activates an *efficiency rule* which restricts the vehicle to service only edges considered "cost-efficient".

4.1. Terminology and definitions

The terminology adopted by the algorithm is given next: $ned(E')$ is the number of required edges in the set E', $td(E') = \sum_{(v_i,v_j)\in E'} d(v_i, v_j)$ is the total demand of set E', $tc(E') = \sum_{(v_i,v_j)\in E'} c(v_i, v_j)$ is the total cost of edges in set E', rvc is the remaining vehicle capacity, α is a real parameter, (v_g, v_h) is the last serviced edge in the route and v_0 is the depot.

The function $near(v_h)$ is given by (3) and defines the set of unserviced edges close enough to v_h. The function $SP(v_i, v_j)$ gives the shortest path cost from node v_i to node v_j.

$$\begin{aligned} near(v_h) = \{(v_i, v_j) \in E_R | min\{SP(v_h, v_i), SP(v_h, v_j)\} \\ \leq tc(E_R)/ned(E_R) \textbf{ and } (v_i, v_j) \text{is unserviced}\} \end{aligned} \tag{3}$$

Let a route R be defined by the sequence $((u_1, v_1), (u_2, v_2), \ldots (u_n, v_n))$, where each (u_i, v_i) is a serviced edge. The route efficiency index $eff(R)$ is given by Eq. (4), where the numerator is the demand serviced and the denominator is the distance traversed including the return to the depot. In summary, $eff(R)$ is the ratio of the demand serviced by distance traversed of route R.

$$eff(R) = \frac{\sum_{i=1}^{n} d(u_i, v_i)}{SP(v_0, u_1) + \sum_{i=1}^{n} c(u_i, v_i) + \sum_{i=1}^{n-1} SP(v_i, u_{i+1}) + SP(v_n, v_0)} \tag{4}$$

4.2. Efficiency rule

The load threshold that activates the *efficiency rule* is given by the *triggering criteria* (5). If $near(v_h)$ is empty, the algorithm adopts in equation (1) instead of (5).

$$rvc \leq \alpha \times td(near(v_h))/ned(near(v_h)) \tag{5}$$

Once the vehicle achieves the load threshold, the vehicle is constrained to service only edges $(v_i, v_j) \in E_R$ satisfying the *efficiency rule* (6).

$$\frac{d(v_i, v_j)}{SP(v_h, v_i) + c(v_i, v_j) + SP(v_j, v_0) - SP(v_h, v_0)} \geq eff(R) \tag{6}$$

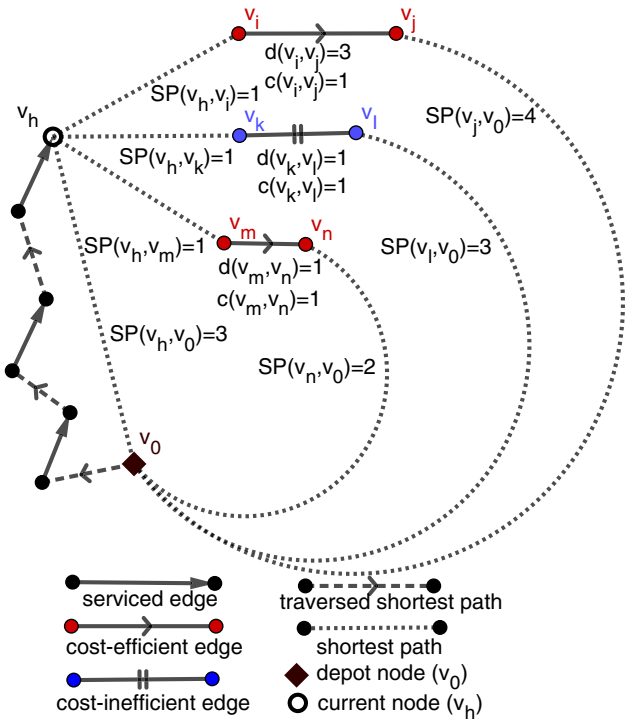

Fig. 1. Example of efficiency rule for a route R with $eff(R) = 1$.

The *efficiency rule* only allows edges to be serviced by the route if their demand compensates for the increase of the route cost; we call these edges cost-efficient. More precisely, the route efficiency index $eff(R)$ is not allowed to decrease once the *efficiency rule* is activated. Fig. 1 gives an example with three required edges (v_i, v_j), (v_k, v_l) and (v_m, v_n). In this example, only the edges (v_i, v_j) and (v_m, v_n) are satisfying the *efficiency rule*.

After the determination of the cost-efficient edges, the algorithm selects the closest one to the vehicle that does not exceed the rvc to be serviced next. Ties are broken randomly. If the vehicle capacity is insufficient to service any cost-efficient edge then the vehicle returns to the depot. If there are still unserviced edges, a new vehicle is employed and the process is repeated. The PS-Efficiency pseudocode is given by Algorithm 1 .

A comparison between PS-Ellipse and PS-Efficiency is provided next:

1. Comparing the *triggering criteria* (1) and (5), the latter considers only unserviced edges that are close to the current node while the former considers all edges. The reasoning for adopting the dynamic criteria (5) instead of the static criteria (1) is that edges which are distant from the current node are often less relevant to decision making.
2. Comparing the *restricting rules* (2) and (6), the *efficiency rule* (6) considers dynamic information about the current route expressed by the efficiency index $eff(R)$. On the other hand, *ellipse rule* (2) relies solely on static information of the instance. The rationale of the *efficiency rule* is that optimal CARP solutions are likely composed of routes with high efficiency indices.

Regarding computational complexity, PS-Efficiency can be implemented in the same worst-case time complexity as PS-Ellipse. From Algorithm 1, the most expensive computational step relies is the determination of the set F of candidate edges, which has $O(|E_R|)$ linear complexity on the number of required edges. That step is present in both PS-Ellipse and PS-Efficiency, and since it is

Algorithm 1: PS-Efficiency.

Input: $G(V, E)$: instance graph; D: vehicle capacity; **c**: cost vector; **d**: demand vector; **SP**: matrix of shortest path costs; α: real parameter; k: number of iterations;
Output: *bestSol*: best solution found;
begin
  $bestSol \leftarrow \emptyset$;
  for $it = 1$ to k **do**
    $sol \leftarrow \emptyset$; $R \leftarrow \{v_0\}$; $rvc \leftarrow D$; $v_h \leftarrow v_0$; $sumDist \leftarrow 0$;
    $rule \leftarrow$ false;
    for $n = 1$ to ned **do**
      if $near(v_h) \neq \emptyset$ **and** inequation (5) is satisfied **then**
        $rule \leftarrow$ true;
      else if $near(v_h) = \emptyset$ **and** inequation (1) is satisfied **then**
        $rule \leftarrow$ true;
      if $rule$ is false **then**
        Determine set of candidate edges $F \subseteq E_R$;
      else
        Determine set of candidate edges $F \subseteq E_R$ satisfying (6);
      if $F = \emptyset$ **then**
        $sol \leftarrow sol \cup (R \cup \{v_0\})$; // end of route
        $R \leftarrow \emptyset$; $rvc \leftarrow D$; $v_h \leftarrow v_0$;
        $sumDist \leftarrow 0$; $rule \leftarrow$ false;
      else
        Select (v_i, v_j) randomly from F and add it to R; // edge is serviced
        $sumDist \leftarrow sumDist + SP(v_h, v_i) + c(v_i, v_j)$;
        $rvc \leftarrow rvc - d(v_i, v_j)$; $v_h \leftarrow v_j$;
        $eff(R) \leftarrow (D - rvc)/(sumDist + SP(v_h, v_0))$;
    if $(cost(sol) < cost(bestSol))$ **or** $bestSol = \emptyset$ **then**
      // If current solution is the better so far
      $bestSol \leftarrow sol$;
  return $(bestSol)$;

repeated $|E_R|$ times for each solution and k solutions are generated, the whole procedure has $O(k|E_R|^2)$ time complexity.

5. Computational experiments

5.1. Settings and instances

The heuristics in Table 1 were implemented in C++ using the LEMON (Dezs et al., 2011) library for graph algorithms and data structures. The experiments were executed in an Intel Core i7-6700K 4.0 GHz with 8 GB of RAM and Linux 64-bit operating system. Full experimental data, instances and the algorithms source codes are available on-line[1].

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

Table 2
Benchmark instances for CARP.

group	$\|V\|$	$\|E\|$	$\|E_R\|$
gdb	7–27	11–55	11–55
val	24–50	34–97	34–97
egl	77–140	98–190	51–190
C	32–97	42–140	32–107
D	32–97	42–140	32–107
E	26–97	35–142	28–107
F	26–97	35–142	28–107
egl-large	255	375	347–375

Table 2presents the data concerning the instances benchmark considered in the experiments, which includes 23 *gdb*(Golden et al., 1982), 34 *val* (Benavent et al., 1991), 24 *egl* (Li and Eglese, 1996), 25 *C* (Beullens et al., 2002), 25 *D* (Beullens et al., 2002), 25 *E* (Beullens et al., 2002), 25 *F* (Beullens et al., 2002), and 10 *egl-large* (Brandão and Eglese, 2008) instances, totaling 191 instances.

5.2. Comparative performance analysis

The results presented in Table 3 compare the PS-Efficiency with three path-scanning methods from literature: PS-RC (Pearn, 1989), PS-RE (Belenguer et al., 2006) and PS-Ellipse (Santos et al., 2009). Each heuristic was evaluated considering multiple values of number of iterations k. Regarding PS-Ellipse and PS-Efficiency, multiple values of parameter α were considered. The results are shown in percentage of the average deviation from lower bounds *Gap(%)*, defined as: $Gap(\%) = 100(UB - LB)/LB$, where UB is the solution cost and LB is the lower bound given by Bode and Irnich (2014). Comparing PS-Efficiency and PS-Ellipse for each k and α, the best deviations are highlighted in bold. Over all heuristics and parameters, the best results were underlined.

Overall comparison. Table 3 shows that the heuristics PS-Efficiency and PS-Ellipse clearly outperformed by a great margin the PS-RC and PS-RE heuristics. For all sets of instances, the average deviations obtained by PS-Efficiency and PS-Ellipse with $k = 1000$ are better than those obtained by PS-RC and PS-RE with $k = 20,000$.

The best overall value of α for PS-Ellipse was $\alpha = 1.5$ and for PS-Efficiency was $\alpha = 3.0$. Considering $k = 20,000$, the heuristics PS-RE, PS-Ellipse($\alpha = 1.5$) and PS-Efficiency($\alpha = 3.0$) reduced the PS-RC overall *Gap(%)* by 0.70%, 42.23% and 53.08%, respectively.

PS-Efficiency vs. PS-Ellipse. In comparison to PS-Ellipse($\alpha = 1.5$), the PS-Efficiency($\alpha = 3.0$) reduced the overall *Gap(%)* by 14.22%, 16.12% and 18.79% for k 1000, 10,000 and 20,000, respectively.

The most significant reduction of deviations from PS-Ellipse($\alpha = 1.5$) to PS-Efficiency($\alpha = 3.0$), considering the results with $k = 20,000$, was reported for the *F* instances where the *Gap(%)* was reduced from 6.04% to 4.18% (reduction of 30.7%) and the least significant reduction was observed for the *gdb* instances from 0.88% to 0.84% (reduction of 4.5%).

Consistency analysis. The results of PS-Efficiency and PS-Ellipse for each value of k and α show that PS-Efficiency performed bet-

Table 1
Path-scanning algorithms.

Algorithm	Triggering criteria	Restricting rule	Tie-breaking
PS-RC (Section 3.1)	–	–	random criteria
PS-RE (Section 3.1)	–	–	random edge
PS-Ellipse (Section 3.1)	static (1)	*ellipse rule* (2)	random edge
PS-Efficiency (Section 4)	dynamic (5)	*efficiency rule* (6)	random edge
PS-Alt1 (Section 5.3)	static (1)	*efficiency rule* (6)	random edge
PS-Alt2 (Section 5.3)	dynamic (5)	*ellipse rule* (2)	random edge

Table 3
Computational experiment results.

group	k	PS-RC(k)	PS-RE(k)		PS-Ellipse(k, α)							PS-Efficiency(k, α)					
				α:	1.0	1.5	2.0	2.5	3.0	3.5	α:	1.0	1.5	2.0	2.5	3.0	3.5
gdb	1000	3.95	3.77		**1.63**	1.50	1.98	1.92	4.83	7.82		1.70	**1.44**	**1.19**	**1.24**	**1.83**	**1.85**
	10,000	2.65	2.15		**1.10**	1.02	1.34	1.22	3.52	6.44		1.16	**0.74**	**0.78**	**0.80**	**1.03**	**1.16**
	20,000	2.21	2.06		1.04	0.88	1.25	1.22	3.46	6.18		**1.02**	**0.74**	**0.75**	**0.71**	**0.84**	**1.05**
val	1000	8.42	8.35		**6.12**	5.38	5.10	5.26	6.03	8.17		6.20	**5.29**	**4.85**	**4.83**	**4.65**	**4.36**
	10,000	5.85	6.29		4.37	3.70	3.42	3.53	4.23	6.31		**4.28**	**3.65**	**3.33**	**3.27**	**2.75**	**3.15**
	20,000	5.60	5.81		3.82	3.38	3.12	3.26	3.98	5.88		**3.79**	**3.18**	**2.98**	**2.80**	**2.53**	**2.78**
egl	1000	16.64	16.43		8.59	9.02	12.04	15.05	19.96	25.32		**8.50**	**7.49**	**7.51**	**7.74**	**7.73**	**8.03**
	10,000	15.32	15.29		7.30	7.67	10.22	13.32	18.62	23.67		**7.07**	**6.49**	**6.41**	**6.59**	**6.80**	**6.47**
	20,000	14.92	15.00		7.05	7.51	10.01	12.62	18.20	23.33		**6.89**	**6.31**	**6.06**	**6.28**	**6.41**	**6.25**
C	1000	16.08	16.27		10.40	9.28	9.50	10.03	10.45	12.65		**9.91**	**9.27**	**8.90**	**8.61**	**8.88**	**8.61**
	10,000	13.39	12.99		7.40	7.27	7.28	7.29	8.64	9.99		**7.44**	**6.83**	**6.96**	**6.34**	**6.69**	**6.61**
	20,000	12.76	12.22		7.26	6.71	6.64	6.60	8.20	9.77		**6.74**	**6.27**	**6.36**	**5.96**	**6.09**	**6.34**
D	1000	12.15	12.30		10.16	9.40	8.48	7.74	7.68	7.20		**9.55**	**8.13**	**7.26**	**7.18**	**6.49**	**6.86**
	10,000	9.23	9.45		6.74	6.28	5.91	5.66	5.35	4.97		**6.34**	**5.74**	**4.75**	**4.86**	**4.93**	**4.48**
	20,000	8.77	8.85		6.26	5.71	5.33	5.03	4.92	4.45		**5.73**	**5.15**	**4.40**	**4.20**	**4.13**	**4.30**
E	1000	16.10	16.18		10.90	10.06	9.91	10.05	11.56	12.62		**9.57**	**9.54**	**8.26**	**8.28**	**8.20**	**8.27**
	10,000	13.09	12.60		8.04	7.68	7.72	7.98	9.25	10.25		**7.45**	**6.93**	**6.29**	**5.97**	**6.00**	**5.82**
	20,000	12.57	11.96		7.34	7.16	6.70	7.38	8.55	9.77		**6.96**	**6.49**	**5.55**	**5.62**	**5.53**	**5.57**
F	1000	12.34	11.53		10.04	8.92	8.96	8.47	8.64	7.75		**9.48**	**8.27**	**8.50**	**7.78**	**7.40**	**6.66**
	10,000	9.20	9.07		7.43	6.53	6.00	6.50	6.05	5.81		**6.64**	**6.40**	**5.80**	**5.47**	**4.77**	**4.64**
	20,000	8.56	8.47		6.83	6.04	5.54	5.84	5.48	5.27		**6.00**	**5.69**	**5.29**	**4.98**	**4.18**	**4.44**
egl-large	1000	28.14	27.61		17.78	16.97	16.98	17.91	18.25	19.53		**17.06**	**16.45**	**16.42**	**16.07**	**16.59**	**16.26**
	10,000	25.50	26.37		16.23	15.65	16.03	16.49	16.79	17.81		**16.12**	**15.47**	**14.76**	**15.19**	**15.22**	**15.08**
	20,000	25.09	26.12		16.01	15.16	15.45	16.09	16.10	17.27		**15.65**	**15.05**	**14.69**	**14.50**	**14.23**	**14.95**
overall	1000	12.96	12.82		8.73	8.09	8.37	8.75	10.14	11.86		**8.31**	**7.53**	**7.12**	**6.99**	**6.94**	**6.84**
	10,000	10.50	10.45		6.55	6.20	6.42	6.90	8.23	9.87		**6.28**	**5.75**	**5.38**	**5.27**	**5.20**	**5.12**
	20,000	10.04	9.97		6.15	5.80	5.94	6.41	7.81	9.46		**5.81**	**5.33**	**4.98**	**4.85**	**4.71**	**4.89**

In **bold**: the best results for each α comparing PS-Efficiency and PS-Ellipse. In underline: the best results regarding all algorithms.

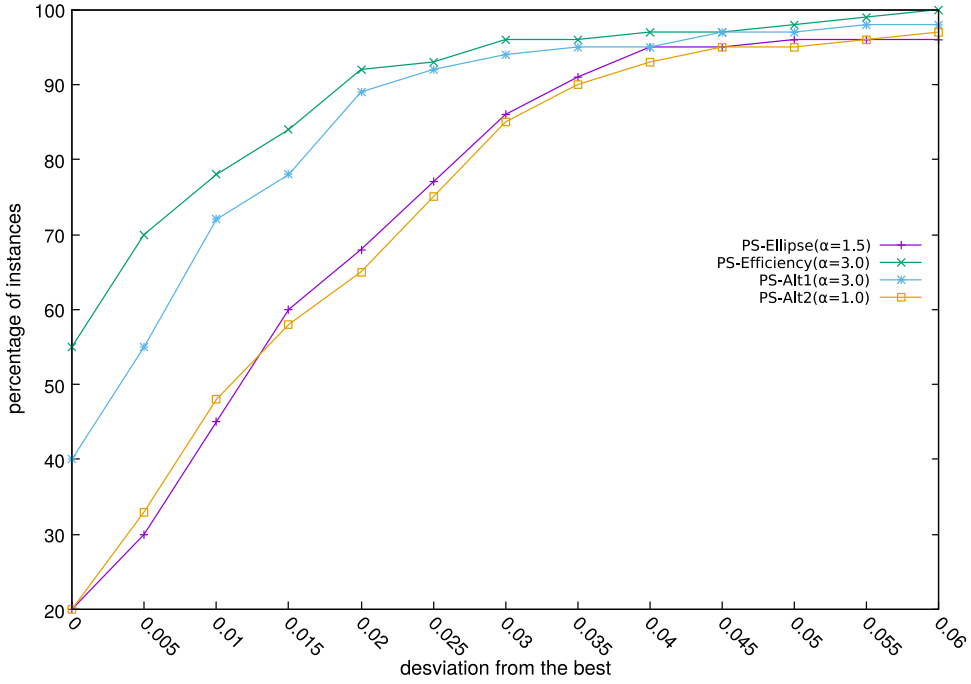

Fig. 2. Comparative experiment of path-scanning algorithms.

ter for every set of instances with the only exceptions being the *gdb* instances with $\alpha = 1.0$ and $k \in \{1000, 10,000\}$; and *val* instances with $\alpha = 1.0$ and $k = 1000$. Therefore, the performance of PS-Efficiency was consistently better for almost all instances and parameters considered.

Robustness analysis. Table 3 shows PS-Efficiency as much more parameter-robust than PS-Ellipse for all sets of instances. Let a variation factor for each heuristic be given by: $100 \times (max_{GAP} - min_{GAP})/min_{GAP}$, where max_{GAP} is the largest deviation obtained in the experiment considering all values of α and min_{GAP} the smallest. The overall variation factor with $k = 20,000$ for PS-Ellipse is 63.10% while the same factor for PS-Efficiency is 23.35%. In special, the set of instances *egl* with $k = 20,000$ presents the highest difference between variation factors, where PS-Ellipse obtained 230.92% and PS-Efficiency obtained 13.69%.

5.3. Effect of triggering criteria and restricting rule

PS-Ellipse and PS-Efficiency are two path-scanning heuristics distinguished by their (i) *triggering criteria* that determine the conditions to activate a (ii) *restricting rule* which restrain the set of edges that can be serviced. To evaluate the effectiveness of these features we have compared PS-Ellipse and PS-Efficiency with two new path-scanning heuristics, called PS-Alt1 and PS-Alt2. These new heuristics implement alternative combinations of the *triggering criteria* and *restricting rules* as shown by Table 1.

The experiment aims to show the impact of each feature embedded in PS-Efficiency. To achieve that, the experiment is executed for the four path-scanning heuristics: PS-Ellipse, PS-Efficiency, PS-Alt1 and PS-Alt2. Each heuristic was executed for all instances in Table 2 with $k = 20,000$ and using its overall best value of parameter α. The performance profiles introduced by Dolan and Moré (2002) are adopted as comparison method.

Fig. 2shows the results. The experiment works by first computing a deviation factor from the best using the following formula for each heuristic and each instance: $(UB - UB_{best})/UB_{best}$, where UB is the solution cost obtained by the heuristic and UB_{best} is the best solution cost obtained among all heuristics. From that, it is calculated a percentage of the instances (y-axis) that fall below certain thresholds of deviation factors from the best (x-axis) for each method.

Fig. 2 shows that PS-Efficiency obtained the best solutions for 60% of the instances while PS-Alt1 obtained the best solutions for 40% of instances. Tied in the last place, PS-Ellipse and PS-Alt2 obtained the best solutions for 20% of the instances.

Considering that PS-Efficiency and PS-Alt1 both use the same *restricting rule* and diverge only by their *triggering criteria*, it is possible to infer from the results that the *efficiency rule* is a key factor for the better performance of these methods in comparison to PS-Ellipse and PS-Alt2.

Observing the performance of PS-Alt2 in Fig. 2, it is revealed that the dynamic *triggering criteria* was ineffective when used in combination with the *ellipse rule*. This indicates that there is a synergy between the dynamic *triggering criteria* and the *efficiency rule*, thus both features should be employed together so as to achieve the best outcome.

5.4. Computational times

Table 4 shows the CPU average processing times required by the heuristics for each set of instances. Each method was executed for $k = 20,000$ iterations and for six different values of the parameter $\alpha \in \{1.0, 1.5, 2.0, 2.5, 3.0, 3.5\}$. The PS-RC and PS-RE are shown having the overall smallest processing times, while PS-Ellipse consumed less computing time than PS-Efficiency. This can be explained by the fact that the dynamic information used by PS-Efficiency requires often updates, such as identifying nearby unserviced edges and computing the route efficiency index. On the other hand, PS-Ellipse uses only static information from the graph thereby requiring less computation. The PS-Alt1 and PS-Alt2 processing times are positioned between PS-Ellipse and PS-Efficiency as expected since they combine features of both methods.

Another important pattern is that the processing time of PS-Efficiency is approximately 1.5 times the processing time of PS-Ellipse for all sets of instances. In other words, the observed time increase seems bounded by a constant factor. This was expected

Table 4
Average processing times for $k = 20,000$.

group	PS-RC	PS-RE	PS-Ellipse	PS-Efficiency	PS-Alt1	PS-Alt2
gdb	0.76	0.70	0.74	1.19	0.91	1.08
val	2.16	2.04	2.18	3.08	2.52	3.04
egl	6.02	5.94	7.70	11.82	9.53	10.12
C	2.34	2.27	2.63	3.90	3.03	3.60
D	2.20	2.12	2.27	3.15	2.55	3.12
E	2.20	2.16	2.45	3.69	2.88	3.38
F	2.08	1.98	2.06	2.91	2.38	2.88
egl-large	59.34	58.04	71.05	97.52	81.04	87.02
overall	5.49	5.34	6.36	9.07	7.41	8.20

Average processing time in seconds (s).

since PS-Efficiency has the same worst-case asymptotic time complexity as PS-Ellipse.

5.5. Comparison to metaheuristic approaches

Constructive heuristics are often incorporated by metaheuristics because they are fast procedures that produce feasible solutions with potentially good quality and high variability. For example: TSA2 (*tabu search algorithm*) (Brandão and Eglese, 2008), GRASP (Usberti et al., 2013) and HMA (Chen et al., 2016) used the path-scanning heuristics PS, PS-Ellipse and PS-RE respectively to warm start its initial solutions.

The CARP literature contains some reports on the effects of incorporating constructive heuristics into metaheuristics frameworks. Lacomme et al. (2004) proposed a *memetic algorithm* (MA) and reported that initializing the MA initial population with three different constructive heuristics influenced positively for the MA final solution quality. Brandão and Eglese (2008) proposed two versions of a deterministic *tabu search algorithm*: TSA1 and TSA2, with the difference being that TSA2 is the TSA1 executed 5 times starting with 5 different initial solutions obtained by 5 different constructive heuristics. Brandão and Eglese argued that "the diversity provided by the different starting solutions was found to be useful in ultimately finding high quality solutions". Polacek et al. (2008) proposed a *variable neighbourhood search* (VNS) and reported results on initializing their method using different solutions obtained by a constructive heuristic. It was reported that the initial solution cost did not influenced much in terms of final solution quality but influenced greatly for shortening the VNS processing time.

Moreover, some recent metaheuristics such as HGA (Arakaki and Usberti, 2018) and MAENS (Fu et al., 2010; Tang et al., 2009) employed constructive heuristics as internal components of its local search methods. In these local search methods, a subset of the routes of a feasible solution is selected and reconstructed using constructive heuristics and, if the reconstructed routes have lower costs than the original ones, then the solution is updated. Authors of MAENS stated that their local search method is likely to generate high-quality solutions due the adoption of constructive heuristics (PS and Ulusoy's) that are known to produce relatively good solutions.

In summary, good constructive heuristics are commonly incorporated by CARP metaheuristics. Nevertheless, a comparison between constructive heuristics and metaheuristics is interesting considering the trade-off involving solution costs and processing times. Table 5 presents results comparing the PS-Efficiency ($\alpha = 3.0$) to two high performance metaheuristics (MAENS (Tang et al., 2009) and HMA (Chen et al., 2016)). Table 5 upper segment presents the average deviation from lower bounds *Gap(%)*, calculated as in Section 5.2. Table 5 lower segment shows the average processing time *CPU(s)* for each method.

The processing times for the metaheuristics were obtained from Chen et al. (2016). Following the practice of literature (Chen et al., 2016; Santos et al., 2010; Usberti et al., 2013), a CPU time conversion factor, based on CPU frequency, was adopted. Therefore,

Table 5
Comparison of constructive heuristics and metaheuristics.

group	PS-Efficiency ($\alpha = 3.0$)			Metaheuristics	
	$k = 1000$	$k = 10,000$	$k = 20,000$	MAENS	HMA
***Gap(%)*(average deviation from lower bounds, in percentage)**					
gdb	1.83	1.03	0.84	0.01	0.00
val	4.65	2.75	2.53	0.22	0.06
egl	7.73	6.80	6.41	0.92	0.39
C	8.88	6.69	6.09	0.46	0.07
D	6.49	4.93	4.13	0.26	0.09
E	8.20	6.00	5.53	0.61	0.05
F	7.40	4.77	4.18	0.30	0.03
egl-large	16.59	15.22	14.23	3.54	1.97
overall	6.94	5.20	4.71	0.55	0.19
***CPU(s)*(average processing time, in seconds)**					
gdb	0.06	0.63	1.26	3.13	0.83*
val	0.21	2.10	3.29	33.85	18.70*
egl	0.69	6.75	13.49	348.94	452.22*
C	0.21	2.13	4.20	115.85	37.95*
D	0.16	1.64	3.36	153.67	24.59*
E	0.20	2.02	4.04	112.62	81.58*
F	0.16	1.55	3.10	116.79	5.91*
egl-large	5.53	55.28	110.40	1706.11	2872.80*
overall	0.51	5.09	10.14	204.88	230.30*

*: HMA is stopped when it achieves a best known lower bound (obtained from literature).

the processing times reported for the metaheuristics (Chen et al., 2016) were scaled by a factor of 0.7. It is worth mentioning that differently from other methods, HMA is stopped when its best incumbent cost achieves a known lower bound (obtained from literature). This characteristic may substantially favor the HMA processing time in comparison to the other methods.

Table 5 shows that metaheuristics obtained better solutions but at the cost of higher processing times. For example, the overall *Gap(%)* for PS-Efficiency ($k = 20,000$) is 4.71% while for the metaheuristics vary from 0.19% to 0.55%. On the other hand, the average processing times for PS-Efficiency is 10.14s while for the metaheuristics vary from 204.88 s to 230.30 s.

6. Conclusion

This paper proposed a constructive heuristic for the capacitated arc routing problem called *path-scanning with efficiency rule* (PS-Efficiency). This heuristic uses a dynamically activated rule which restricts the search towards promising candidate edges to be serviced next by the current route. The *efficiency rule* is based on how each edge would affect the current route efficiency calculated as the ratio of serviced demand by traversed distance.

Computational experiments conducted on a set of benchmark instances revealed that the proposed heuristic outperformed all previous path-scanning heuristics by a considerable margin and was found to be more parameter-robust. Moreover, the impact of the *efficiency rule* was investigated and found to be a key factor for the high performance of the PS-Efficiency heuristic.

Future research can focus on assessing the effectiveness of PS-Efficiency when applied to other routing problems, such as the vehicle routing problem (Toth and Vigo, 2014). Another possible line of research is to investigate the speed-ups that can be achieved by implementing a parallelized PS-Efficiency.

Acknowledgment

This work was supported by grants #2016/00315-0, São Paulo Research Foundation (FAPESP) and 307472/2015-9, National Counsel of Technological and Scientific Development (CNPq). Also, we would like to thank the anonymous referees for their helpful comments.

References

Afsar, H.M., 2010. A branch-and-price algorithm for capacitated arc routing problem with flexible time windows. Electron. Notes Discret. Math. 36, 319–326.
Arakaki, R.K., Usberti, F.L., 2018. Hybrid genetic algorithm for the open capacitated arc routing problem. Comput. Oper. Res. 90, 221–231.
Belenguer, J.M., Benavent, E., Irnich, S., 2014. The Capacitated Arc Routing Problem: Exact Algorithms. In: Corberán, Á., Laporte, G. (Eds.), Arc Routing: Problems, Methods, and Applications. SIAM Publications, Philadelphia, PA, pp. 183–221. Ch. 9
Belenguer, J.-M., Benavent, E., Lacomme, P., Prins, C., 2006. Lower and upper bounds for the mixed capacitated arc routing problem. Comput. Oper. Res. 33 (12), 3363–3383.
Benavent, E., Campos, V., Corberán, A., Mota, E., 1991. The capacitated arc routing problem: lower bounds. Networks 22, 669–690.
Beullens, P., Muyldermans, L., Cattrysse, D., Van Oudheusden, D., 2002. A guided local search heuristic for the capacitated arc routing problem. Eur. J. Oper. Res. 147, 629–643.
Bode, C., Irnich, S., 2014. The shortest-path problem with resource constraints with (k,2)-loop elimination and its application to the capacitated arc-routing problem. Eur. J. Oper. Res. 238 (2), 415–426.
Bosco, A., Laganà, D., Musmanno, R., Vocaturo, F., 2013. Modeling and solving the mixed capacitated general routing problem. Optim. Lett. 7 (7), 1451–1469.
Brandão, J., Eglese, R., 2008. A deterministic tabu search algorithm for the capacitated arc routing problem. Comput. Oper. Res. 35, 1112–1126.
Chapleau, L., Ferland, J.A., Lapalme, G., Rousseau, J.-M., 1984. A parallel insert method for the capacitated arc routing problem. Oper. Res. Lett. 3 (2), 95–99.
Chen, Y., Hao, J.-K., Glover, F., 2016. A hybrid metaheuristic approach for the capacitated arc routing problem. Eur. J. Oper. Res. 253 (1), 25–39.
Dezs, B., Jüttner, A., Kovács, P., 2011. Lemon - an open source c++ graph template library. Electron. Notes Theor. Comput. Sci. 264 (5), 23–45.
Dolan, E.D., Moré, J.J., 2002. Benchmarking optimization software with performance profiles. Math. Program. 91 (2), 201–213.
Eglese, R., Golden, B., Wasil, E., 2014. Route Optimization for Meter Reading and Salt Spreading. In: Corberán, Á., Laporte, G. (Eds.), Arc Routing: Problems, Methods, and Applications. SIAM Publications, pp. 303–320. Ch. 13
Eydi, A., Javazi, L., 2012. Model and solution approach for multi objective-multi commodity capacitated arc routing problem with fuzzy demand. J. Ind. Syst. Eng. 5 (4), 208–229.
Fu, H., Mei, Y., Tang, K., Zhu, Y., 2010. Memetic algorithm with heuristic candidate list strategy for capacitated arc routing problem. In: Proceedings of the IEEE Congress on Evolutionary Computation, pp. 1–8. Barcelona, Spain
Ghiani, G., Mourão, C., Pinto, L., Vigo, D., 2014. Routing in Waste Collection Applications. In: Corberán, Á., Laporte, G. (Eds.), Arc Routing: Problems, Methods, and Applications. SIAM Publications, Philadelphia, PA, pp. 351–370. Ch. 15
Golden, B.L., Wong, R.T., 1981. Capacitated arc routing problems. Networks 11, 305–315.
Golden, L.B., DeArmon, S.J., Baker, K.E., 1982. Computational experiments with algorithms for a class of routing problems. Comput. Oper. Res. 10, 47–59.
Grandinetti, L., Guerriero, F., Laganà, D., Pisacane, O., 2012. An optimization-based heuristic for the multi-objective undirected capacitated arc routing problem. Comput. Oper. Res. 39 (10), 2300–2309.
Lacomme, P., Prins, C., Ramdane-Cherif, W., 2004. Competitive memetic algorithms for arc routing problems. Ann. Oper. Res. 131 (1–4), 159–185.
Li, Y.L., Eglese, W.R., 1996. An interactive algorithm for vehicle routeing for winter-gritting. J. Oper. Res. Soc. 217–228.
Liu, G., Ge, Y., Qiu, T.Z., Soleymani, H.R., 2014. Optimization of snow plowing cost and time in an urban environment: a case study for the city of edmonton. Can. J. Civ. Eng. 41 (7), 667–675.
Mourão, M.C., Pinto, L.S., 2017. An updated annotated bibliography on arc routing problems. Networks 70 (3), 144–194.
Muyldermans, L., Pang, G., 2014. Variants of the Capacitated Arc Routing Problem. In: Corberán, Á., Laporte, G. (Eds.), Arc Routing: Problems, Methods, and Applications. SIAM Publications, Philadelphia, PA, pp. 223–253. Ch. 10
Pearn, W.L., 1989. Approximate solutions for the capacitated arc routing problem. Comput. Oper. Res. 16 (6), 589–600.
Pearn, W.L., 1991. Augment-insert algorithms for the capacitated arc routing problem. Comput. Oper. Res. 18, 189–198.
Polacek, M., Doerner, K.F., Hartl, R.F., Maniezzo, V., 2008. A variable neighborhood search for the capacitated arc routing problem with intermediate facilities. J. Heuristics 14 (5), 405–423.
Prins, C., 2014. The Capacitated Arc Routing Problem: Heuristics. In: Corberán, Á., Laporte, G. (Eds.), Arc Routing: Problems, Methods, and Applications. SIAM Publications, Philadelphia, PA, pp. 131–157. Ch. 7
Prins, C., Labadi, N., Reghioui, M., 2009. Tour splitting algorithms for vehicle routing problems. Int. J. Prod. Res. 47 (2), 507–535.
Santos, L., Coutinho-Rodrigues, J., Current, J.R., 2009. An improved heuristic for the capacitated arc routing problem. Comput. Oper. Res. 36 (9), 2632–2637.
Santos, L., Coutinho-Rodrigues, J., Current, J.R., 2010. An improved ant colony optimization based algorithm for the capacitated arc routing problem. Transp. Res. Part B 44 (2), 246–266.
Shang, R., Ma, H., Wang, J., Jiao, L., Stolkin, R., 2016. Immune clonal selection algorithm for capacitated arc routing problem. Soft comput. 20 (6), 2177–2204.
Tang, K., Mei, Y., Yao, X., 2009. Memetic algorithm with extended neighborhood search for capacitated arc routing problems. IEEE Trans. Evol. Comput. 13 (5), 1151–1166.
Toth, P., Vigo, D., 2014. Vehicle routing: Problems, methods, and applications. SIAM Publications, Philadelphia, PA.
Ulusoy, G., 1984. The fleet size and mix problem for capacitated arc routing. Eur. J. Oper. Res. 22, 329–337.
Usberti, F.L., França, P.M., França, A.L.M., 2011. The open capacitated arc routing problem. Comput. Oper. Res. 38 (11), 1543–1555.
Usberti, F.L., França, P.M., França, A.L.M., 2013. Grasp with evolutionary path-relinking for the capacitated arc routing problem. Comput. Oper. Res. 40 (12), 3206–3217.
Vansteenwegen, P., Souffriau, W., Sörensen, K., 2010. Solving the mobile mapping van problem: a hybrid metaheuristic for capacitated arc routing with soft time windows. Comput.Oper.Res. 37 (11), 1870–1876.
Wøhlk, S., 2005. Contributions to arc routing. University of Southern Denmark, Aarhus, Denmark Ph.D. thesis.
Zhang, Y., Mei, Y., Tang, K., Jiang, K., 2017. Memetic algorithm with route decomposing for periodic capacitated arc routing problem. Appl. Soft Comput. 52, 1130–1142.