Computers & Operations Research 40 (2013) 2145–2160

Contents lists available at SciVerse ScienceDirect

Computers & Operations Research

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

Improved bounds for large scale capacitated arc routing problem

Rafael Martinelli [a,*], Marcus Poggi [a], Anand Subramanian [b]

[a] Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio) – Departamento de Informática – Rua Marquês de São Vicente, 225 – RDC, 4º andar – Gávea – Rio de Janeiro, RJ 22451-900, Brazil

[b] Universidade Federal da Paraíba (UFPB) – Departamento de Engenharia de Produção – Centro de Tecnologia, Campus I – Bloco G, Cidade Universitária – João Pessoa, PB 58051-970, Brazil

ARTICLE INFO

Available online 28 February 2013

Keywords:
Arc routing
Integer programming
Dual ascent
Capacity cuts
Iterated local search

ABSTRACT

The Capacitated Arc Routing Problem (CARP) stands among the hardest combinatorial problems to solve or to find high quality solutions. This becomes even more true when dealing with large instances. This paper investigates methods to improve on lower and upper bounds of instances on graphs with over 200 vertices and 300 edges, dimensions that, today, can be considered of large scale. On the lower bound side, we propose to explore the speed of a dual ascent heuristic to generate capacity cuts. These cuts are next improved with a new exact separation enchained to the linear program resolution that follows the dual heuristic. On the upper bound, we implement a modified Iterated Local Search procedure to Capacitated Vehicle Routing Problem (CVRP) instances obtained by applying a transformation from the CARP original instances. Computational experiments were carried out on the set of large instances generated by Brandão and Eglese and also on the regular size sets. The experiments on the latter allow for evaluating the quality of the proposed solution approaches, while those on the former present improved lower and upper bounds for all instances of the corresponding set.

1. Introduction

The Capacitated Arc Routing Problem (CARP) can be defined as follows. Let $G=(V,E)$ be an undirected graph, where V and E are the vertex and edge set respectively. There is a special vertex called *depot* (usually vertex 0) where a set I of identical vehicles with capacity Q is located. Each edge in E has a cost $c: E \rightarrow \mathbb{Z}^+$ and a demand $d: E \rightarrow \mathbb{Z}_0^+$. Let $E_R = \{e \in E : d_e > 0\}$ be the set of required edges. The objective is to find a set of routes, one for each available vehicle, which minimizes the total traversal cost satisfying the following constraints: (i) every route starts and ends at the depot; (ii) each required edge must be visited exactly once; (iii) the total load of each vehicle must not exceed Q.

This problem can arise in many real life situations. According to Wølhk [1], some of the applications studied in the literature are garbage collection, street sweeping, winter gritting, electric meter reading and airline scheduling.

The CARP is $\mathcal{NP}$-hard and it was first proposed by Golden and Wong in 1981 [2]. Since then, several solution approaches were proposed in the literature involving algorithms based on heuristics, metaheuristics, cutting plane, column generation, branch-and-bound, among others.

In 2003, Belenguer and Benavent [3] proposed a mathematical formulation for the CARP which makes use of two families of cuts as constraints, the *odd-edge cutset cuts* and the *capacity cuts*. With this formulation and other families of cuts, they devised a cutting plane algorithm in order to obtain good lower bounds for well-known CARP instance datasets. Before this work, the best known CARP lower bounds were found mainly by heuristic algorithms.

Since the work of Belenguer and Benavent, the best known lower bounds were found using exact algorithms. In 2004, Ahr [4] devised a mixed-integer formulation using an exact separation of capacity cuts. However, due to memory limitations, the author did not manage to apply his algorithm in all known instances, which illustrates the difficulty in separating such cuts.

The main drawback of the exact approaches is the fact of being prohibitive on larger instances. Up to this date, the larger instance solved to optimality is the *egl-s*3-*c* from the *eglese* instance dataset, proposed almost 20 years ago by Li [5] and Li and Eglese [6]. This instance has 140 vertices and 190 edges, 159 of these required ones, and it was solved for the first time by Bartolini et al. in 2011 [7] using a cut-and-column based technique combined with a set partitioning approach. Other recent works using exact approaches which solved to optimality instances from *eglese* instance dataset are those of Bode and Irnich [8], which used a cut-first branch-and-price-second exploiting the sparsity of the instances, and Martinelli et al. [9], which used a branch-cut-and-price with non-elementary routes.

In their work of 2008, Brandão and Eglese [10] proposed a new set of CARP instances, called *egl-large*, containing 255 vertices,

* Corresponding author.
E-mail addresses: rmartinelli@inf.puc-rio.br (R. Martinelli), poggi@inf.puc-rio.br (M. Poggi), anand@ct.ufpb.br (A. Subramanian).

0305-0548/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.cor.2013.02.013

375 edges and 347 or 375 required edges. They ran the path-scanning heuristic from Golden [11] and compared the results with their deterministic tabu search, giving the first upper bounds for this instance dataset. In 2009, Mei et al. [12] improved these upper bounds using a repair-based tabu search algorithm. To the best of our knowledge, there are no lower bounds reported in the literature for this instance dataset.

The contributions of this paper are twofold: (i) provide a methodology capable of obtaining good lower bounds and (ii) improve the existing upper bounds by means of a heuristic algorithm; both approaches with emphasis on large scale instances. In order to find the first lower bounds for the *egl-large* instance dataset, we devise a *dual ascent heuristic* to speed up a cutting plane algorithm which uses a new exact separation of the capacity cuts and a known exact separation of the odd edge cutset cuts. The upper bounds are found using a known transformation to the Capacitated Vehicle Routing Problem (CVRP) and then applying an *Iterated Local Search* (ILS) based heuristic. We report new improved upper bounds for all 10 instances of the *egl-large* set.

The remainder of the paper is organized as follows. Section 2 presents the mathematical formulation needed for the dual ascent heuristic and the known exact separation algorithms. Section 3 introduces a new exact separation for the capacity cuts. Section 4 describes our dual ascent heuristic and how it generates cuts to hot-start the cutting plane algorithm. Section 5 explains the known transformation to the CVRP and the ILS heuristic. Section 6 presents extensive computational experiments. Finally, conclusions are given in Section 7.

2. Mathematical formulation

2.1. The one-index formulation

In their work, Belenguer and Benavent [3] developed a CARP formulation, usually referred as the *One-Index Formulation* [13]. In contrast to other approaches, this formulation only makes use of variables representing the deadheading of an edge. An edge is deadheaded when a vehicle traverses this edge without servicing it. In addition, all vehicles are aggregated. Due to these simplifications, this formulation is not complete, i.e., it may result in an infeasible solution for the problem. Moreover, even when a given solution is feasible, it is a very hard task to find a complete solution. Nevertheless, these issues do not prevent such formulation of giving very good lower bounds in practice.

For each deadheaded edge e, there is an integer variable z_e representing the number of times the edge e was deadheaded by *any* vehicle. Let $S \subseteq V \backslash \{0\}$ be a subset of vertices not including the depot. We can define $\delta(S) = \{(i,j) \in E : i \in S \wedge j \notin S\}$ as being the set of edges which have one endpoint inside S and the other outside S. Similarly, $\delta_R(S) = \{(i,j) \in E_R : i \in S \wedge j \notin S\}$ is the set of *required* edges which have one endpoint inside S and the other outside S. Analogously, $E(S) = \{(i,j) \in E : i \in S \wedge j \in S\}$ and $E_R(S) = \{(i,j) \in E_R : i \in S \wedge j \in S\}$ are the sets of edges with both endpoints inside S.

Given a vertex set S, with $|\delta_R(S)|$ odd, it is easy to conclude that at least one edge in $\delta(S)$ must be deadheaded because each vehicle entering the set S must leave and return to the depot. This is the principle of the *odd-edge cutset cuts*:

$$\sum_{e \in \delta(S)} z_e \geq 1 \quad \forall S \subseteq V \backslash \{0\},\ |\delta_R(S)| \text{odd} \tag{1}$$

Furthermore, we can define a lower bound on the number of vehicles needed to meet the demands in $\delta_R(S) \cup E_R(S)$ as $k(S) = \lceil \sum_{e \in \delta_R(S) \cup E_R(S)} d_e / Q \rceil$. These $k(S)$ vehicles must enter and leave the set S, in such a way that at least $2k(S) - |\delta_R(S)|$ times an edge in $\delta(S)$ will be deadheaded. If this value is positive, we can define the following *capacity cut*:

$$\sum_{e \in \delta(S)} z_e \geq 2k(S) - |\delta_R(S)| \quad \forall S \subseteq V \backslash \{0\} \tag{2}$$

Since the left-hand side of both (1) and (2) are the same, they can be represented in the formulation by only using a single constraint. This can be done by introducing $\alpha(S)$, which is defined as follows:

$$\alpha(S) = \begin{cases} \max\{2k(S) - |\delta_R(S)|, 1\} & \text{if } |\delta_R(S)| \text{is odd}, \\ \max\{2k(S) - |\delta_R(S)|, 0\} & \text{if } |\delta_R(S)| \text{is even} \end{cases} \tag{3}$$

These two families of cuts define the one-index formulation:

$$\text{Min} \quad \sum_{e \in E} c_e z_e \tag{4}$$

$$\text{s.t.} \quad \sum_{e \in \delta(S)} z_e \geq \alpha(S) \quad \forall S \subseteq V \backslash \{0\} \tag{5}$$

$$z_e \in \mathbb{Z}_0^+ \quad \forall e \in E \tag{6}$$

The objective function (4) minimizes the cost of the deadheaded edges. Constraints (5) combine cuts (1) and (2). In order to obtain the total cost for the problem, one needs to add the costs of the required edges ($\sum_{e \in E_R} c_e$) to the solution cost.

2.2. Exact odd-degree cutset cuts separation

The exact separation of the odd-degree cutset cuts (1) can be done in polynomial time using the *Odd Minimum Cutset Algorithm* of Padberg and Rao [14]. We believe that the application of the algorithm is not immediate and therefore we decided to provide a brief description of the separation routine, which is as follows.

The odd minimum cutset algorithm creates a *Gomory-Hu Tree* [15] using just the vertices with odd $|\delta_R(\{v\})|$, called *terminals*. This tree represents a *maximum flow tree*, i.e., the maximum flow of any pair of vertices is represented on this tree. In order to obtain the maximum flow between a pair of vertices, one only needs to find the least cost edge on the unique path between these two vertices. This edge also represents the minimum cut between them. Hence, to determine a violated odd-degree cutset cut, one needs to find any edge with a value less than one. This can be done during the execution of the algorithm, but we prefer to run it until the end to find as many violated cuts as possible.

This whole operation can be done running at most $|V|-1$ times any maximum flow algorithm. In this work we use the *Edmonds–Karp Algorithm* [16], which takes $\mathcal{O}(|V| \cdot |E|^2)$, resulting in a total complexity of $\mathcal{O}(|V|^2 \cdot |E|^2)$.

2.3. Ahr's exact capacity cut separation

The only exact separation routine for the capacity cuts available in the CARP literature was proposed by Ahr [4] in 2004. This algorithm runs a mixed-integer formulation several times, one for each possible number of vehicles. This approach was inspired on the exact separation of the capacity cuts for the CVRP proposed by Fukasawa et al. [17]. In Ahr's work, this separation was used to identify violated cuts on a complete formulation for the CARP. As we only wish to separate the cuts, we changed the objective function of the mixed-integer formulation to use it with the one-index formulation.

The formulation is composed by three types of variables. The first one is the binary variable h_e, $\forall e \in E$, which is 1 when exactly one endpoint of e is inside the cut (what we call *cut edge*) and 0 otherwise. The second variable is the binary variable f_e, $\forall e \in E$, which is 1 when both endpoints of e are inside the cut (called *inner edge*) and 0 otherwise. The last variable is the binary variable s_i, $\forall i \in V$, which is 1 if vertex i is inside the cut and

0 otherwise. These variables are sufficient to describe a capacity cut. Thus, the following formulation is created for each possible number of vehicles $k = 0 \ldots \lceil \sum_{e \in E_R} d_e/Q \rceil - 1$:

$$\text{Min} \quad \sum_{e \in E} \tilde{z}_e h_e + \sum_{e \in E_R} h_e \tag{7}$$

$$\text{s.t.} \quad h_e - s_i + s_j \geq 0 \quad \forall e = \{i,j\} \in E \tag{8}$$

$$h_e + s_i - s_j \geq 0 \quad \forall e = \{i,j\} \in E \tag{9}$$

$$-h_e + s_i + s_j \geq 0 \quad \forall e = \{i,j\} \in E \tag{10}$$

$$s_i - f_e \geq 0 \quad \forall e = \{i,j\} \in E \tag{11}$$

$$s_j - f_e \geq 0 \quad \forall e = \{i,j\} \in E \tag{12}$$

$$s_i + s_j - f_e \leq 1 \quad \forall e = \{i,j\} \in E \tag{13}$$

$$\sum_{e \in \delta(\{i\})} (h_e + f_e) - s_i \geq 0 \quad \forall i \in V \tag{14}$$

$$h_e + f_e \leq 1 \quad \forall e \in E \tag{15}$$

$$\sum_{e \in E_R} d_e(h_e + f_e) \geq kQ + 1 \tag{16}$$

$$s_0 = 0 \tag{17}$$

$$h_e, f_e \in \{0,1\} \quad \forall e \in E \tag{18}$$

$$s_i \in [0,1] \quad \forall i \in V \backslash \{0\} \tag{19}$$

The objective function (7) uses a solution of the one-index formulation $\tilde{z}_e$ and minimizes the total value of the cut edges plus the number of cut edges that are required. Constraints (8)–(10) bind the variables s_i and h_e. Analogously, constraints (11)–(13) bind the variables s_i and f_e. The constraints (14) assure that if a vertex i is inside the cut, at least one edge adjacent to i is a cut edge or an inner edge. Constraints (15) assure that an edge e cannot be a cut edge and an inner edge at the same time. Constraints (16) ensure that the total demand of the cut found is at least $kQ+1$. Constraint (17) forbids the inclusion of the depot in a cut. Notice that due to the association of s_i with h_e and f_e, the variables s_i need not to be integral.

Given the value of the objective function Z^* associated to a solution in a given iteration k, the cut which can be generated using the s_i variables is a violated capacity cut if $Z^* < 2(k-1)$. Therefore, the problem needs to be solved to optimality only when we aim at finding the most violated capacity cut.

This separation routine has the disadvantage of running several MIPs, one for every possible number of vehicles. Depending on the instance, this number may be up to 42. Nevertheless, in his work, Ahr could not manage to run this separation for all CARP instances due to memory limitations.

3. A new exact capacity cut separation

The exact separation suggested by Ahr requires solving several MIPs because it is not possible to build a mixed-integer formulation that directly represents the *ceiling function* ($\lceil \cdot \rceil$) of the capacity cut. In order to deal with this issue, we developed a new formulation which is capable of separating a capacity cut in an exact fashion considering any possible number of vehicles. Our approach was inspired by the exact separation of the *Chvátal-Gomory cuts* proposed by Fischetti and Lodi in 2007 [18].

Our mixed-integer formulation uses the same three variables presented in Ahr's formulation, that is, h_e, f_e and s_i. In addition, we also consider an integer variable κ indicating the value of $k(S)$ in the formulation and a continuous slack variable γ representing the fractional difference of applying the ceiling function to obtain κ. This difference must be within the range [0, 1).

Furthermore, we use constraints (8)–(15) and (17) from Ahr's formulation. These constraints are required to depict a capacity cut. We write our complete formulation as follows:

$$\text{Max} \quad 2\kappa - \sum_{e \in E_R} h_e - \sum_{e \in E} \tilde{z}_e h_e \tag{20}$$

$$\text{s.t.} \quad h_e - s_i + s_j \geq 0 \quad \forall e = \{i,j\} \in E \tag{21}$$

$$h_e + s_i - s_j \geq 0 \quad \forall e = \{i,j\} \in E \tag{22}$$

$$-h_e + s_i + s_j \geq 0 \quad \forall e = \{i,j\} \in E \tag{23}$$

$$s_i - f_e \geq 0 \quad \forall e = \{i,j\} \in E \tag{24}$$

$$s_j - f_e \geq 0 \quad \forall e = \{i,j\} \in E \tag{25}$$

$$s_i + s_j - f_e \leq 1 \quad \forall e = \{i,j\} \in E \tag{26}$$

$$\sum_{e \in \delta(\{i\})} (h_e + f_e) - s_i \geq 0 \quad \forall i \in V \tag{27}$$

$$h_e + f_e \leq 1 \quad \forall e \in E \tag{28}$$

$$\kappa = \sum_{e \in E} \frac{d_e(h_e + f_e)}{Q} + \gamma \tag{29}$$

$$s_0 = 0 \tag{30}$$

$$h_e, f_e \in \{0,1\} \quad \forall e \in E \tag{31}$$

$$s_i \in [0,1] \quad \forall i \in V \backslash \{0\} \tag{32}$$

$$\kappa \in \mathbb{Z}_0^+ \tag{33}$$

$$\gamma \in [0,1) \tag{34}$$

The objective function (20) maximizes the violation of the capacity cut, while constraint (29) limits the difference between κ and the fractional value using the slack variable γ. As mentioned, constraints (21)–(28) and (30) are from Ahr's formulation. We will further show in the computational experiments that this formulation can perform better in practice than Ahr's formulation.

4. Dual ascent heuristic

Even with the improvement on the exact separation of the capacity cuts, the separation routine still takes a long time when applied to large instances. However, if we use a heuristic approach to generate valid cuts to be used as a hot-start for the separation algorithm, the number of iterations of the separation routine could reduce drastically. In view of this, we propose a dual ascent heuristic.

A dual ascent heuristic is usually devised to obtain good lower bounds for a problem. A good example of this type of approach can be found in the work of Wong [19] on the Steiner Tree Problem. When this heuristic is applied over the CARP one-index formulation, it can generate several cuts on each iteration. If good cuts are found during these iterations, they can be very helpful for the exact separation.

4.1. Main algorithm

The main algorithm of the dual ascent heuristic works on the dual of the linear relaxation of the one-index formulation:

$$\text{Max} \quad \sum_{S \subseteq V \backslash \{0\}} \alpha(S)\pi_S \tag{35}$$

$$\text{s.t.} \sum_{S \subseteq V \setminus \{0\}: e \in \delta(S)} \pi_S \le c_e \quad \forall e \in E \tag{36}$$

$$\pi_S \in \mathbb{R}_0^+ \quad \forall S \subseteq V \setminus \{0\} \tag{37}$$

In this formulation, the variables π_S are associated with constraints (5) and constraints (36) are associated with z_e variables. These latter constraints impose a limit on the dual variables. The sum of the dual variables associated with the cuts which have an edge $e \in \delta(S)$ must not exceed the cost of this edge e. This is the base of our dual ascent heuristic.

As already mentioned, the objective of our dual ascent heuristic is to find a lower bound for the CARP. Therefore, it starts with the trivial lower bound $LB = \sum_{e \in E_R} c_e$. At each iteration, several cuts are generated using a strategy that will be further discussed. Among these cuts, only one is chosen using an arbitrary criterion. A good cut is one with a large $\alpha(S)$ or, in case of a tie, one with a large contribution to the objective function. The contribution of a cut can be calculated as presented in (38).

$$\sigma(S) = \alpha(S) \cdot \min\{c_e : e \in \delta(S)\} \tag{38}$$

Given the selected cut S^*, the heuristic updates its lower bound ($LB = LB + \sigma(S^*)$) and it also changes the dual formulation to reflect the use of this cut. Knowing the value of the variable $\pi_{S^*} = \min\{c_e : e \in \delta(S^*)\}$ associated with the cut, each constraint of the dual formulation where $e \in \delta(S^*)$ must have its right-hand side modified to $c_e - \pi_{S^*}$. As a result, the variable π_{S^*} is removed from the formulation.

This latter operation has a direct effect on the graph G. The update of the right-hand side of the constraints (36) is the same of reducing the costs of the edges $e \in \delta(S)$. When an edge $e = (i,j)$ is saturated, i.e., the edge has its cost reduced to 0, the heuristic contracts the vertices i and j as shown in Fig. 1. This contraction guarantees that no saturated edges appear as cut edges on future iterations of the heuristic.

The next iteration of the heuristic is then applied over the new graph. When the graph has only one vertex (the depot), the heuristic stops. Notice that at each iteration, at least one edge is saturated. Due to this fact, the heuristic performs at most $|V|-1$ iterations.

4.2. Cut generation

As pointed before, the dual ascent heuristic can only give good lower bounds if good cuts are chosen. Therefore, the cut generation strategies are the most important part of the heuristic. Any strategy can be used within our heuristic. After some preliminary experiments, we decided to turn attention to four different strategies. When one of the strategies generates a previously generated cut or a cut S with $\alpha(S) = 0$, this new cut is discarded.

4.2.1. Simple cuts

In the *simple cuts* strategy, we create a set of cuts $S = \{v\}$, $\forall v \in V \setminus \{0\}$, containing only one vertex. Such vertex cannot be the depot. It is noteworthy to mention that as the graph is modified during the iterations of the heuristic, a vertex at some iteration might not be a single vertex on the original graph. An example of this strategy is shown in Fig. 2a. This strategy takes time $\mathcal{O}(|V|)$ and generates at most $|V|-1$ cuts.

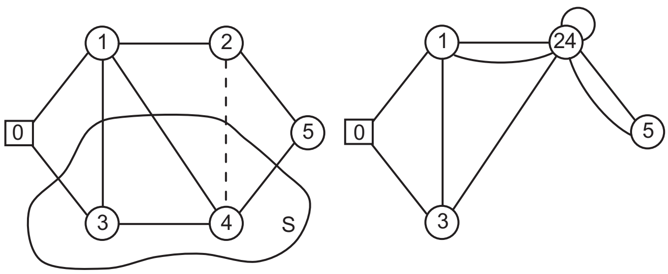

Fig. 1. Example of vertex contraction: vertices 2 and 4 are contracted, becoming one vertex.

4.2.2. Complete cuts

In the *complete cuts* strategy, we create a set of cuts $S = V \setminus \{0, v\}$, $\forall v \in V$, which, for each vertex $v \in V$ (including the depot), contains all the vertices of the graph except v and the depot. Analogously to the previous strategy, the vertex left out of the cut might not be a single vertex at a given iteration of the heuristic. An example of this strategy is shown in Fig. 2b. This strategy takes time $\mathcal{O}(|V|^2)$ and generates at most $|V|$ cuts.

4.2.3. Connected cuts

The *connected cuts* strategy inserts vertices in the cut using a *breadth-first search* approach. Firstly, it chooses a random size for the cut between 2 and $|V|-2$, as all the cuts of size 1, $|V|-1$ and $|V|$ are generated in the first two strategies. Secondly, it chooses a random vertex (excluding the depot) to start the search. Each time the breadth-first search finds a new vertex, this vertex is added to the cut. The search stops when the size of the cut is equal to the desired size. This operation is repeated $|E|$ times. The whole operation takes time $\mathcal{O}(|E|(|V|+|E|))$ and generates at most $|E|$ cuts.

4.2.4. MST cuts

The *MST cuts* strategy starts by generating the *Minimum Spanning Tree* (MST) of the graph. Each edge of the MST defines two vertex set on the graph. Those which do not contain the depot are then generated as cuts (see Fig. 3). Using the *Kruskal's Algorithm* [20] for MST, along with any search algorithm, this strategy takes time $\mathcal{O}(|E| log |V|)$ and generates at most $|V|-1$ cuts.

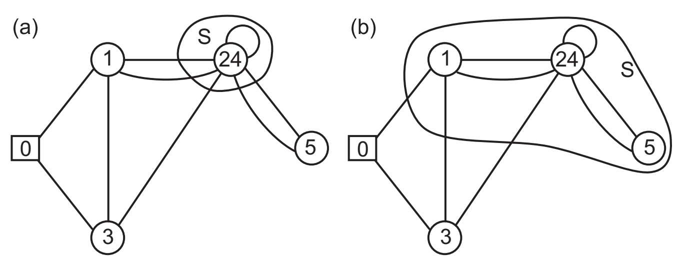

Fig. 2. (a) Example of the simple cuts strategy and (b) example of the complete cut strategy.

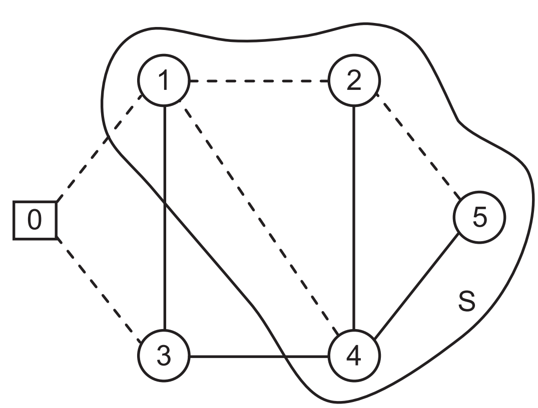

Fig. 3. Example of a MST cut defined by edge (0,1). The minimum spanning tree is shown by dashed edges.

Table 1
Exact separation results for *kshs* and *gdb* datasets.

Ins	$\|V\|$	$\|E\|$	$\|E_R\|$	$\|I\|$	Opt	$Cost_1$	$Cost_2$	Ahr's exact sep						Our exact sep					
								Cap_1	Odd_1	$Time_1$	Cap_2	Odd_2	$Time_2$	Cap_1	Odd_1	$Time_1$	Cap_2	Odd_2	$Time_2$
kshs1	8	15	15	4	14,661	**14,661**	**14,661**	2	5	0.091	2	5	0.136	2	5	0.046	2	5	0.072
kshs2	10	15	15	4	9863	**9863**	**9863**	4	6	0.124	4	6	0.162	4	6	0.075	4	6	0.099
kshs3	6	15	15	4	9320	**9320**	**9320**	1	6	0.196	1	6	0.292	1	6	0.038	1	6	0.070
kshs4	8	15	15	4	11,498	11,098	11,098	3	6	0.163	3	6	0.230	3	6	0.048	3	6	0.066
kshs5	8	15	15	3	10,957	**10,957**	**10,957**	1	8	0.182	1	8	0.261	1	8	0.043	1	8	0.060
kshs6	9	15	15	3	10,197	**10,197**	**10,197**	0	17	0.031	0	17	0.061	0	17	0.016	0	17	0.031
gdb1	12	22	22	5	316	**316**	**316**	2	17	0.228	2	17	0.353	2	17	0.058	2	17	0.085
gdb2	12	26	26	6	339	**339**	**339**	1	7	0.233	1	7	0.372	1	7	0.031	1	7	0.052
gdb3	12	22	22	5	275	**275**	**275**	2	14	0.311	2	14	0.420	2	14	0.077	2	14	0.103
gdb4	11	19	19	4	287	**287**	**287**	4	8	0.244	4	8	0.346	4	8	0.065	4	8	0.083
gdb5	13	26	26	6	377	**377**	**377**	3	15	0.289	3	15	0.428	3	15	0.076	3	15	0.100
gdb6	12	22	22	5	298	**298**	**298**	2	13	0.308	2	13	0.415	2	13	0.062	2	13	0.093
gdb7	12	22	22	5	325	**325**	**325**	2	23	0.218	2	23	0.317	2	23	0.049	2	23	0.078
gdb8	27	46	46	10	348	344	344	14	33	1.400	14	33	1.804	21	33	0.760	21	33	0.825
gdb9	27	51	51	10	303	**303**	**303**	14	28	1.690	14	28	2.192	11	28	0.316	11	28	0.438
gdb10	12	25	25	4	275	**275**	**275**	0	7	0.451	0	7	0.898	0	7	0.049	0	7	0.096
gdb11	22	45	45	5	395	**395**	**395**	1	21	0.453	1	21	0.692	1	21	0.066	1	21	0.116
gdb12	13	23	23	7	458	450	450	5	11	0.328	5	11	0.472	3	11	0.054	3	11	0.082
gdb13	10	28	28	6	536	**536**	**536**	1	11	1.717	1	11	2.349	1	11	0.089	1	11	0.132
gdb14	7	21	21	5	100	**100**	**100**	1	0	0.687	1	0	1.103	1	0	0.020	1	0	0.032
gdb15	7	21	21	4	58	**58**	**58**	1	0	0.444	1	0	0.689	1	0	0.024	1	0	0.039
gdb16	8	28	28	5	127	**127**	**127**	1	7	1.459	1	7	2.077	1	7	0.045	1	7	0.081
gdb17	8	28	28	5	91	**91**	**91**	0	8	0.790	0	8	1.581	0	8	0.016	0	8	0.032
gdb18	9	36	36	5	164	**164**	**164**	1	0	1.542	1	0	2.462	1	0	0.056	1	0	0.101
gdb19	8	11	11	3	55	**55**	**55**	0	10	0.023	0	10	0.046	0	10	0.016	0	10	0.032
gdb20	11	22	22	4	121	**121**	**121**	0	14	0.198	0	14	0.391	0	14	0.019	0	14	0.057
gdb21	11	33	33	6	156	**156**	**156**	1	7	1.972	1	7	2.529	1	7	0.079	1	7	0.114
gdb22	11	44	44	8	200	**200**	**200**	1	33	4.657	1	33	6.538	1	33	0.115	1	33	0.176
gdb23	11	55	55	10	233	**233**	**233**	2	0	4.569	2	0	6.824	2	0	0.086	2	0	0.138

Table 2
Exact separation results for *bccm* dataset.

Ins	$\|V\|$	$\|E\|$	$\|E_R\|$	$\|I\|$	Opt	$Cost_1$	$Cost_2$	Ahr's exact sep						Our exact sep					
								Cap_1	Odd_1	$Time_1$	Cap_2	Odd_2	$Time_2$	Cap_1	Odd_1	$Time_1$	Cap_2	Odd_2	$Time_2$
1A	24	39	39	2	173	**173**	**173**	0	48	0.115	0	48	0.216	0	48	0.057	0	48	0.106
1B	24	39	39	3	173	**173**	**173**	0	48	0.117	0	48	0.226	0	48	0.051	0	48	0.093
1C	24	39	39	8	245	235	235	23	48	1.458	23	48	1.772	13	48	0.277	13	48	0.325
2A	24	34	34	2	227	**227**	**227**	3	36	0.280	3	36	0.399	3	36	0.132	3	36	0.169
2B	24	34	34	3	259	257	257	6	44	0.436	6	44	0.552	6	44	0.209	6	44	0.250
2C	24	34	34	8	457	455	455	24	31	1.446	24	31	1.701	20	31	0.477	20	31	0.519
3A	24	35	35	2	81	**81**	**81**	2	26	0.198	2	26	0.264	2	26	0.106	2	26	0.146
3B	24	35	35	3	87	**87**	**87**	9	26	0.580	9	26	0.724	8	25	0.221	8	25	0.262
3C	24	35	35	7	138	135	135	22	23	1.481	22	23	1.731	18	23	0.365	18	23	0.405
4A	41	69	69	3	400	**400**	**400**	5	32	0.918	5	32	1.327	4	34	0.307	4	34	0.403
4B	41	69	69	4	412	**412**	**412**	5	32	1.097	5	32	1.517	4	34	0.272	4	34	0.362
4C	41	69	69	5	428	**428**	**428**	9	34	2.173	9	34	2.666	9	34	0.463	9	34	0.554
4D	41	69	69	9	530	519.5	521	39	63	7.764	39	63	8.489	31	62	2.022	31	63	2.140
5A	34	65	65	3	423	**423**	**423**	2	57	0.905	2	57	1.182	2	57	0.225	2	57	0.305
5B	34	65	65	4	446	443	443	4	63	1.114	4	63	1.466	4	63	0.303	4	63	0.384
5C	34	65	65	5	474	467	467	6	77	1.825	6	77	2.463	7	77	0.348	7	77	0.430
5D	34	65	65	9	577	571	571	17	56	3.208	17	56	3.971	15	56	0.693	15	56	0.775
6A	31	50	50	3	223	**223**	**223**	2	36	0.281	2	36	0.423	2	36	0.115	2	36	0.166
6B	31	50	50	4	233	229	229	3	38	0.659	3	38	0.846	3	38	0.217	3	38	0.269
6C	31	50	50	10	317	307	307	30	36	2.690	30	36	3.207	22	36	0.950	22	36	1.044
7A	40	66	66	3	279	**279**	**279**	0	30	0.132	0	30	0.261	0	30	0.077	0	30	0.151
7B	40	66	66	4	283	**283**	**283**	1	30	0.377	1	30	0.551	1	30	0.098	1	30	0.176
7C	40	66	66	9	334	327	327	16	112	2.300	16	112	2.881	11	106	0.666	11	106	0.775
8A	30	63	63	3	386	**386**	**386**	1	31	0.603	1	31	0.813	1	31	0.165	1	31	0.231
8B	30	63	63	4	395	**395**	**395**	4	31	0.917	4	31	1.228	4	31	0.228	4	31	0.303
8C	30	63	63	9	521	509	509	25	64	3.844	25	64	4.570	19	65	0.659	19	65	0.728
9A	50	92	92	3	323	**323**	**323**	0	112	0.367	0	112	0.690	0	112	0.154	0	112	0.266
9B	50	92	92	4	326	**326**	**326**	1	122	1.671	1	122	2.158	1	122	0.320	1	122	0.427
9C	50	92	92	5	332	**332**	**332**	2	126	2.026	2	126	2.619	2	126	0.339	2	126	0.459
9D	50	92	92	10	391	378	378	23	112	5.241	23	112	6.175	15	112	1.213	15	112	1.344
10A	50	97	97	3	428	**428**	**428**	1	66	0.830	1	66	1.245	1	66	0.203	1	66	0.335
10B	50	97	97	4	436	**436**	**436**	2	68	2.400	2	68	3.249	2	68	0.346	2	68	0.464
10C	50	97	97	5	446	**446**	**446**	6	69	3.508	6	69	4.453	7	70	0.652	7	70	0.778
10D	50	97	97	10	526	521.5	522	47	73	15.851	48	73	17.376	28	74	2.592	28	74	2.719

5. Iterated local search heuristic

With a view of improving the existing upper bounds for the CARP large-scale instances, we implemented an ILS [21] based heuristic which was originally proposed by Penna et al. [22] for solving the Heterogeneous Fleet Vehicle Routing Problem (HFVRP). However, instead of completely redesigning the algorithm to solve CARP instances, we applied a procedure that transforms a CARP instance into a CVRP instance. Some transformation routines are available in the literature (see for example Pearn et al. [23], Longo et al. [24], Baldacci and Maniezzo [25]). In this work we decided to make use of the one developed in [25]. Since the HFVRP includes the CVRP as a special case when all vehicles are identical, we only had to perform minor adaptations in the original heuristic.

5.1. The ILS-RVND heuristic

The multi-start heuristic, called ILS-RVND, combines the ILS approach with a local search procedure based on the Variable Neighborhood Descent [26] with Random neighborhood ordering (RVND) [27]. The two main parameters of this heuristic are the number of iterations (*MaxIter*) and the number of consecutive perturbations without improvements (*MaxIterILS*).

The initial solutions are generated using two insertion strategies, namely: (i) Sequential Insertion Strategy, in which a single route is considered at a time; and (ii) Parallel Insertion Strategy, in which all routes are considered at once. Two insertion criteria were adopted, specifically: (i) Modified Cheapest Insertion Criterion, in which the insertion cost g of customer k between customers i and j in route u is given by $g(k)=(c^u_{ik}+c^u_{kj}-c^u_{ij})-\gamma(c^u_{0k}+c^u_{k0})$, where $\gamma \in \{0.00, 0.05, \ldots, 1.70\}$ is a parameter whose interval was empirically calibrated in [27]; and (ii) Cheapest Insertion Criterion, where the insertion cost g is given by $g(k)=c^u_{ik}$.

The transformed instances contain a subset of edges with artificial negative costs that must be in any feasible solution. The constructive procedure does not necessarily impose the inclusion of such edges when generating an initial solution. Hence, initial infeasible solutions are often generated. Nevertheless, these solutions eventually become feasible during the local search.

The RVND procedure is composed by the following four inter-route neighborhood structures. **Shift(1,0)**, a customer k is transferred from a route r_1 to a route r_2. **Shift(2,0)**, two adjacent customers, k and l, are transferred from a route r_1 to a route r_2. This move can also be seen as an arc transferring. In this case, the move examines the transferring of both arcs (k,l) and (l,k). **Swap(2,2)**, permutation between two adjacent customers, k and l, from a route r_1 by another two adjacent customers k' and l', belonging to a route r_2. We consider the four possible combinations of exchanging arcs (k,l),(l,k),(k',l') and (l',k'). **Cross**, the arc between adjacent customers k and l, belonging to a route r_1, and the one between k' and l', from a route r_2, are both removed. Next, an arc is inserted connecting k and l' and another is inserted linking k' and l. In case of improvement we perform a intensification in the modified routes using the following three classical Traveling Salesman Problem neighborhood structures. **2-opt**, two non-adjacent arcs are deleted and another two are added in such a way that a new route is generated. **Reinsertion**, one customer is removed and inserted in another position of the route. **Or-opt2**, two adjacent customers are removed and inserted in another position of the route. The solution spaces of all neighborhoods are exhaustively explored and their computational complexity is $\mathcal{O}(n^2)$, where n is the number of customers. We only consider those moves that do not violate the vehicle capacity.

It is noteworthy to mention that in the work of Penna et al. [22] four other CVRP neighborhood structures were considered in the local search, namely Swap(1,1), Swap(2,1), Exchange and Or-opt3. We disregarded such neighborhoods because they revealed to be ineffective when applied to the transformed instances.

Two simple perturbation mechanisms were adopted. The first one is **Multiple-Swap(1,1)**, where multiple Swap(1,1) moves are performed randomly, and the second one is **Multiple-Shift(1,1)**, where multiple Shift(1,1) moves are performed randomly. In Swap(1,1), a customer k from a route r_1 is exchanged with a customer l, from a route r_2. The Shift(1,1) consists in transferring a customer k from a route r_1 to a route r_2, whereas a customer

Table 3
Exact separation results for C dataset.

Ins	$\|V\|$	$\|E\|$	$\|E_R\|$	$\|I\|$	LB	$Cost_1$	$Cost_2$	Ahr's exact sep						Our exact sep					
								Cap_1	Odd_1	$Time_1$	Cap_2	Odd_2	$Time_2$	Cap_1	Odd_1	$Time_1$	Cap_2	Odd_2	$Time_2$
C01	69	98	79	9	4105	4070	4075	99	80	23.401	99	80	24.693	95	92	12.037	96	92	12.482
C02	48	66	53	7	3135	**3135**	**3135**	65	69	7.108	65	69	7.717	47	48	3.125	47	48	3.215
C03	46	64	51	6	2575	2525	2525	51	66	5.306	51	66	5.734	66	66	4.506	66	66	4.607
C04	60	84	72	8	3478	3455	3455	67	54	9.970	67	54	10.650	44	54	3.224	44	54	3.407
C05	56	79	65	10	5365	5305	5305	154	46	27.472	154	46	28.515	80	49	6.516	80	49	6.683
C06	38	55	51	6	2535	2495	2495	16	40	1.385	16	40	1.727	12	40	0.450	12	40	0.522
C07	54	70	52	8	4075	4015	4015	119	54	14.855	119	54	15.531	86	54	6.004	86	54	6.178
C08	66	88	63	8	4090	4000	4000	123	27	24.239	123	27	25.164	86	27	8.249	86	27	8.518
C09	76	117	97	12	5233	5215	5215	131	215	42.724	131	215	44.829	56	189	7.966	56	189	8.238
C10	60	82	55	9	4700	4597.5	4620	131	78	19.331	134	80	21.279	141	73	17.857	148	73	19.215
C11	83	118	94	10	4583	4550	4550	200	234	60.606	200	234	62.023	79	248	12.758	79	248	13.079
C12	62	88	72	9	4209	4140	4140	209	66	43.984	209	66	45.038	111	84	15.603	111	84	15.801
C13	40	60	52	7	2955	2895	2895	23	28	2.216	23	28	2.650	26	28	1.363	26	28	1.472
C14	58	79	57	8	4030	3970	3970	73	80	10.182	73	80	11.040	54	80	3.719	54	80	3.911
C15	97	140	107	11	4912	4845	4845	144	110	55.379	144	110	58.131	123	110	41.717	123	110	42.664
C16	32	42	32	3	1475	1470	1470	26	21	1.476	26	21	1.643	31	21	1.076	31	21	1.130
C17	43	56	42	7	3555	3535	3535	71	40	6.818	71	40	7.317	91	40	5.941	91	40	6.044
C18	93	133	121	11	5577	5550	5550	148	79	59.036	148	79	61.354	104	81	21.444	104	81	22.346
C19	62	84	61	6	3096	3065	3065	99	78	12.728	99	78	13.376	60	75	5.052	60	75	5.218
C20	45	64	53	5	2120	**2120**	**2120**	24	55	2.702	24	55	3.010	11	55	0.480	11	55	0.562
C21	60	84	76	8	3960	3950	3950	65	38	9.727	65	38	10.418	50	38	3.714	50	38	3.884
C22	56	76	43	4	2245	**2245**	**2245**	35	51	3.518	35	51	3.845	35	51	1.848	35	51	1.921
C23	78	109	92	8	4032	4012.5	4040	149	169	41.487	155	169	45.886	99	193	16.280	102	193	17.604
C24	77	115	84	7	3384	3370	3370	118	97	29.205	118	97	30.421	73	97	10.006	73	97	10.241
C25	37	50	38	5	2310	**2310**	**2310**	48	106	3.420	48	106	3.648	33	110	1.620	33	110	1.700

l from r_2 is transferred to r_1. As in the local search, only those moves that do not violate the vehicle capacity are admitted.

The main steps of the ILS-RVND heuristic are described as follows:

Step 0: Let *iter* be the current iteration. If $iter \le MaxIter$ then generate an initial solution by choosing an insertion strategy and an insertion criterion at random. Otherwise, stop.

Step 1: If the current solution is infeasible then perform a local search using the RVND procedure considering all neighborhood structures. Otherwise, apply RVND without Shift(1,0).

Step 2: Let *iterILS* be the current number of perturbations without improvements. If $iterILS \le MaxIterILS$ then apply one of the perturbation mechanisms at random and go to Step 1. Otherwise, update the incumbent solution (if necessary) and go to Step 0.

Table 4
Exact separation results for D dataset.

Ins	$\|V\|$	$\|E\|$	$\|E_R\|$	$\|I\|$	LB	$Cost_1$	$Cost_2$	Ahr's exact sep						Our exact sep					
								Cap_1	Odd_1	$Time_1$	Cap_2	Odd_2	$Time_2$	Cap_1	Odd_1	$Time_1$	Cap_2	Odd_2	$Time_2$
D01	69	98	79	5	3215	**3215**	**3215**	13	57	2.367	13	57	2.972	11	57	0.877	11	57	1.023
D02	48	66	53	4	2520	**2520**	**2520**	22	30	1.414	22	30	1.632	17	30	0.823	17	30	0.896
D03	46	64	51	3	2065	**2065**	**2065**	7	73	0.943	7	73	1.196	6	73	0.380	6	73	0.462
D04	60	84	72	4	2785	**2785**	**2785**	28	71	3.961	28	71	4.458	19	65	1.470	19	65	1.596
D05	56	79	65	5	3935	**3935**	**3935**	30	47	2.980	30	47	3.295	25	42	1.026	25	42	1.111
D06	38	55	51	3	2125	**2125**	**2125**	1	40	0.311	1	40	0.478	1	40	0.094	1	40	0.162
D07	54	70	52	4	3115	3015	3015	9	50	1.159	9	50	1.575	9	50	0.512	9	50	0.611
D08	66	88	63	4	2995	2975	2975	22	27	3.113	22	27	3.629	36	27	3.994	36	27	4.139
D09	76	117	97	6	4120	**4120**	**4120**	21	59	6.020	21	59	6.960	13	59	1.411	13	59	1.569
D10	60	82	55	5	3340	3330	3330	8	53	1.346	8	53	1.777	11	53	0.974	11	53	1.072
D11	83	118	94	5	3745	**3745**	**3745**	18	281	6.068	18	281	6.880	18	277	2.026	18	277	2.187
D12	62	88	72	5	3310	**3310**	**3310**	50	64	7.498	50	64	8.004	39	64	2.111	39	64	2.230
D13	40	60	52	4	2535	**2535**	**2535**	5	54	0.877	5	54	1.107	3	54	0.202	3	54	0.273
D14	58	79	57	4	3272	3270	3270	38	81	4.502	38	81	4.837	42	81	3.998	42	81	4.093
D15	97	140	107	6	3990	**3990**	**3990**	11	110	3.647	11	110	4.777	18	110	1.611	18	110	1.833
D16	32	42	32	2	1060	**1060**	**1060**	3	20	0.287	3	20	0.405	5	20	0.234	5	20	0.278
D17	43	56	42	4	2620	**2620**	**2620**	23	48	1.426	23	48	1.603	14	44	0.668	14	44	0.737
D18	93	133	121	6	4165	**4165**	**4165**	40	87	10.832	40	87	11.905	39	86	2.972	39	86	3.173
D19	62	84	61	3	2393	2370	2370	18	66	3.018	18	66	3.394	29	63	2.609	29	63	2.743
D20	45	64	53	3	1870	**1870**	**1870**	1	55	0.475	1	55	0.636	1	55	0.191	1	55	0.272
D21	60	84	76	4	2985	2940	2940	18	38	1.951	18	38	2.397	20	38	1.435	20	38	1.650
D22	56	76	43	2	1865	**1865**	**1865**	15	51	0.660	15	51	0.802	19	51	0.820	19	51	0.929
D23	78	109	92	4	3114	3110	3110	10	94	3.221	10	94	3.931	12	94	1.186	12	94	1.364
D24	77	115	84	4	2676	2660	2660	25	95	4.754	25	95	5.223	19	95	1.814	19	95	1.973
D25	37	50	38	3	1815	**1815**	**1815**	13	75	0.859	13	75	0.992	17	75	1.045	17	75	1.098

Table 5
Exact separation results for E dataset.

Ins	$\|V\|$	$\|E\|$	$\|E_R\|$	$\|I\|$	LB	$Cost_1$	$Cost_2$	Ahr's exact sep						Our exact sep					
								Cap_1	Odd_1	$Time_1$	Cap_2	Odd_2	$Time_2$	Cap_1	Odd_1	$Time_1$	Cap_2	Odd_2	$Time_2$
E01	73	105	85	10	4885	4830	4830	104	81	28.878	104	81	30.342	57	81	5.990	57	81	6.245
E02	58	81	58	8	3990	3960	3960	94	161	12.502	94	161	13.160	32	161	2.578	32	161	2.679
E03	46	61	47	5	2015	**2015**	**2015**	26	56	2.063	26	56	2.382	25	56	0.927	25	56	1.009
E04	70	99	77	9	4155	4125	4125	98	81	18.284	98	81	19.375	55	81	6.819	55	81	7.111
E05	68	94	61	9	4585	4555	4555	80	79	16.531	80	79	17.533	38	79	3.136	38	79	3.291
E06	49	66	43	5	2055	**2055**	**2055**	28	39	2.853	28	39	3.188	22	39	1.233	22	39	1.312
E07	73	94	50	8	4155	4035	4035	121	126	20.459	121	126	21.300	58	126	5.066	58	126	5.254
E08	74	98	59	9	4710	4640	4640	260	169	56.820	260	169	57.811	129	169	20.040	129	169	20.270
E09	93	141	103	12	5780	5745	5745	236	237	105.287	236	237	108.340	116	237	29.640	116	237	30.296
E10	56	76	49	7	3605	**3605**	**3605**	87	90	10.077	87	90	10.702	48	90	3.195	48	90	3.315
E11	80	113	94	10	4637	4620	4630	216	247	65.394	216	249	67.176	103	273	20.861	103	273	21.130
E12	74	103	67	9	4180	4065	4065	177	348	42.218	177	348	43.436	50	348	5.548	50	348	5.790
E13	49	73	52	7	3345	3305	3320	126	59	17.960	126	59	18.578	70	52	6.109	70	54	6.280
E14	53	72	55	8	4115	4085	4085	57	92	6.859	57	92	7.551	27	92	1.379	27	92	1.504
E15	85	126	107	9	4189	4170	4170	64	496	17.723	64	496	19.276	84	496	14.160	84	496	14.391
E16	60	80	54	7	3755	3735	3735	104	44	13.644	104	44	14.278	100	42	8.183	100	42	8.325
E17	38	50	36	5	2740	**2740**	**2740**	82	40	6.688	82	40	6.994	55	43	2.490	55	43	2.557
E18	78	110	88	8	3825	**3825**	**3825**	111	343	26.712	111	343	27.614	58	343	5.014	58	343	5.165
E19	77	103	66	6	3222	3192.5	3200	148	97	28.123	150	99	31.857	84	87	7.646	86	87	8.363
E20	56	80	63	7	2802	2785	2785	44	232	6.091	44	232	6.668	35	232	2.189	35	232	2.300
E21	57	82	72	7	3728	3725	3725	38	56	5.614	38	56	6.216	39	61	2.077	39	61	2.191
E22	54	73	44	5	2470	2440	2440	62	160	6.352	62	160	6.648	50	160	3.873	50	160	3.954
E23	93	130	89	8	3686	3675	3675	190	246	58.855	190	246	60.477	151	209	32.832	151	209	33.221
E24	97	142	86	8	4001	3930	3930	154	261	66.954	154	261	68.266	80	261	12.624	80	261	12.972
E25	26	35	28	4	1615	**1615**	**1615**	13	60	0.652	13	60	0.789	9	60	0.229	9	60	0.268

6. Computational experiments

For the sake of comparison, we applied our algorithms to all well-known CARP instance datasets, namely: *kshs* [28], *gdb* [29,11], *bccm* [30], *eglese* [5,6], *beullens* (*C*, *D*, *E* and *F*) [31] and *egl-large* [10]. The first four are known as the classical CARP instance datasets and have been widely used in the literature over the past 20 years. The last two were created more recently and only some recent works have attempted to solve them.

The datasets *kshs*, *gdb* and *bccm* were artificially generated and have no non-required edges. On the other hand, the *eglese* and *egl-large* datasets were constructed using as underlying graph regions of the road network of the county of Lancashire (UK). Analogously, the *beullens* dataset was constructed based on the intercity road network in Flanders (Belgium). The instances belonging to these last three datasets have costs and demands proportional to the length of the edges and most of them have non-required edges.

As mentioned before, the objective of this work is focused on solving the large scale CARP instances. The instances we consider as

Table 6
Exact separation results for *F* dataset.

Ins	$\|V\|$	$\|E\|$	$\|E_R\|$	$\|I\|$	LB	Cost_1	Cost_2	Ahr's exact sep						Our exact sep					
								Cap_1	Odd_1	Time_1	Cap_2	Odd_2	Time_2	Cap_1	Odd_1	Time_1	Cap_2	Odd_2	Time_2
F01	73	105	85	5	4040	**4040**	**4040**	20	81	5.465	20	81	6.343	15	81	1.995	15	81	2.145
F02	58	81	58	4	3300	**3300**	**3300**	14	163	2.577	14	163	2.994	12	165	1.780	12	165	1.866
F03	46	61	47	3	1665	**1665**	**1665**	10	56	0.468	10	56	0.570	12	56	0.355	12	56	0.419
F04	70	99	77	5	3476	3475	3475	34	88	6.511	34	88	7.048	23	88	2.042	23	88	2.151
F05	68	94	61	5	3605	**3605**	**3605**	22	79	4.196	22	79	4.683	13	79	0.636	13	79	0.738
F06	49	66	43	3	1875	**1875**	**1875**	15	40	1.081	15	40	1.232	7	40	0.502	7	40	0.571
F07	73	94	50	4	3335	**3335**	**3335**	38	126	6.297	38	126	6.795	29	126	3.780	29	126	3.901
F08	74	98	59	5	3690	**3690**	**3695**	66	183	11.802	66	183	12.232	50	202	3.534	50	202	3.701
F09	93	141	103	6	4730	**4730**	**4730**	22	235	7.745	22	235	9.033	17	235	3.154	17	235	3.427
F10	56	76	49	4	2925	**2925**	**2925**	22	109	2.080	22	109	2.364	30	116	2.097	30	116	2.175
F11	80	113	94	5	3835	**3835**	**3835**	12	327	5.069	12	327	5.905	13	300	1.606	13	300	1.756
F12	74	103	67	5	3390	3385	3385	8	348	2.040	8	348	2.689	4	348	0.437	4	348	0.586
F13	49	73	52	4	2855	**2855**	**2855**	6	49	0.894	6	49	1.132	6	49	0.240	6	49	0.332
F14	53	72	55	4	3330	**3330**	**3330**	20	92	1.982	20	92	2.316	9	92	0.566	9	92	0.653
F15	85	126	107	5	3560	**3560**	**3560**	6	494	2.265	6	494	2.908	6	494	0.778	6	494	0.932
F16	60	80	54	4	2725	**2725**	**2725**	17	42	0.941	17	42	1.192	17	42	0.582	17	42	0.674
F17	38	50	36	3	2055	**2055**	**2055**	15	29	0.897	15	29	1.040	12	29	0.495	12	29	0.582
F18	78	110	88	4	3063	3060	3060	12	343	2.044	12	343	2.533	14	343	1.373	14	343	1.518
F19	77	103	66	3	2500	2485	2485	35	64	5.859	35	64	6.274	52	64	7.246	52	64	7.406
F20	56	80	63	4	2445	**2445**	**2445**	5	232	0.763	5	232	1.037	5	232	0.391	5	232	0.475
F21	57	82	72	4	2930	**2930**	**2930**	33	54	2.926	33	54	3.132	48	54	3.510	48	54	3.606
F22	54	73	44	3	2075	**2075**	**2075**	27	160	1.807	27	160	1.964	20	160	0.927	20	160	1.000
F23	93	130	89	4	2994	2985	2985	28	108	9.256	28	108	10.096	19	102	3.168	19	102	3.330
F24	97	142	86	4	3210	**3210**	**3210**	18	267	5.954	18	267	6.592	23	267	3.350	23	267	3.560
F25	26	35	28	2	1390	**1390**	**1390**	5	60	0.179	5	60	0.227	5	60	0.192	5	60	0.235

Table 7
Exact separation results for *eglese* dataset.

Ins	$\|V\|$	$\|E\|$	$\|E_R\|$	$\|I\|$	LB	Cost_1	Cost_2	Ahr's exact sep						Our exact sep					
								Cap_1	Odd_1	Time_1	Cap_2	Odd_2	Time_2	Cap_1	Odd_1	Time_1	Cap_2	Odd_2	Time_2
e1-A	77	98	51	5	3548	3527	3527	167	81	25.487	167	81	26.017	123	81	9.698	123	81	9.850
e1-B	77	98	51	7	4498	4463.7	4468	274	82	44.833	274	82	45.568	229	82	25.598	229	82	25.781
e1-C	77	98	51	10	5595	5513	5513	280	81	50.179	280	81	51.190	208	81	23.685	208	81	23.927
e2-A	77	98	72	7	5018	4995	4995	106	101	16.478	106	101	17.192	105	101	7.826	105	101	7.966
e2-B	77	98	72	10	6305	6271	6273	168	101	26.946	169	101	28.087	140	101	14.638	140	101	14.804
e2-C	77	98	72	14	8335	8160.5	8165	248	101	44.159	250	101	46.252	194	101	25.035	194	101	25.353
e3-A	77	98	87	8	5898	5893.8	**5898**	111	209	20.363	111	209	21.169	87	213	9.230	87	213	9.377
e3-B	77	98	87	12	7729	7648.7	7649	149	161	33.012	149	161	34.498	110	175	13.252	110	175	13.561
e3-C	77	98	87	17	10,244	10124.5	10138	170	138	37.693	171	139	41.034	144	139	15.081	148	141	16.074
e4-A	77	98	98	9	6408	6378	6378	75	304	15.082	75	304	16.157	48	298	3.501	48	298	3.685
e4-B	77	98	98	14	8935	8838	8838	126	280	24.165	126	280	25.891	104	310	9.656	104	310	10.201
e4-C	77	98	98	19	11,493	11,376	11383	176	270	35.119	176	270	37.468	127	279	13.885	127	279	14.300
s1-A	140	190	75	7	5018	5010	5010	571	215	265.508	571	215	267.202	410	215	123.690	410	215	124.410
s1-B	140	190	75	10	6388	6368	6368	865	215	461.099	865	215	463.965	507	215	140.192	507	215	141.378
s1-C	140	190	75	14	8518	8404	8404	801	215	394.296	801	215	398.753	533	215	152.893	533	215	154.139
s2-A	140	190	147	14	9825	9737	9737	240	234	182.092	240	234	187.452	164	315	72.018	164	315	76.212
s2-B	140	190	147	20	13,017	12901	12901	357	171	240.034	357	171	247.175	215	171	68.024	215	171	71.968
s2-C	140	190	147	27	16,425	16247.3	16,274	525	171	617.949	526	171	632.157	330	171	426.016	347	171	452.645
s3-A	140	190	159	15	10146	10082.5	10083	263	545	186.441	263	545	191.255	210	370	144.411	210	370	145.777
s3-B	140	190	159	22	13,648	13,568	13,568	399	240	276.988	399	240	284.552	269	240	165.409	269	240	168.352
s3-C	140	190	159	29	17,188	17,006.4	17,019	467	240	716.157	469	240	738.009	322	240	612.498	328	240	637.911
s4-A	140	190	190	19	12,144	12,026	12,026	181	139	114.597	181	139	120.558	136	139	31.896	136	139	33.040
s4-B	140	190	190	27	16,103	15,984	16,001	396	139	322.022	399	139	337.803	232	139	178.946	239	139	186.743
s4-C	140	190	190	35	20,430	20,235.3	20,256	462	139	368.004	466	139	387.719	278	139	235.180	294	139	246.744

Table 8
Dual ascent results for *kshs* and *gdb* datasets.

Ins	Opt	Dual ascent									Single cuts			Complete cuts			Connected cuts			MST cuts		
		Cost	Time	Cuts	SGL	CMP	CON	MST	Int	Time	Cost	Time	Cuts	Cost	Time	Cuts	Cost	Time	Cuts	Cost	Time	Cuts
kshs1	14,661	**14,661**	< 0.01	42	11	19	19	5	**14,661**	< 0.01	11277	< 0.01	10	**14,661**	< 0.01	20	10,542	< 0.01	17	**14,661**	< 0.01	8
kshs2	9863	**9863**	< 0.01	70	25	34	34	3	**9863**	< 0.01	8099	< 0.01	12	9325	< 0.01	33	8160	< 0.01	40	9275	< 0.01	7
kshs3	9320	**9320**	< 0.01	21	11	9	5	0	**9320**	< 0.01	9045	< 0.01	8	8813	< 0.01	7	8114	< 0.01	5	9045	< 0.01	5
kshs4	11,498	11,098	< 0.01	46	9	23	15	6	11,098	< 0.01	8680	< 0.01	5	11,098	< 0.01	22	8998	< 0.01	16	10,774	< 0.01	6
kshs5	10,957	**10,957**	< 0.01	43	12	16	22	4	**10,957**	< 0.01	10,353	< 0.01	8	10,921	< 0.01	17	9934	< 0.01	18	**10,957**	< 0.01	9
kshs6	10,197	**10,197**	< 0.01	59	9	22	28	6	**10,197**	< 0.01	**10,197**	< 0.01	11	**10,197**	< 0.01	28	9932	< 0.01	33	10,192	< 0.01	12
best											2			4			0			3		
gdb1	316	311	< 0.01	109	32	43	57	20	**316**	0.01	**316**	< 0.01	18	299	< 0.01	36	280	< 0.01	46	308	< 0.01	22
gdb2	339	**339**	< 0.01	102	18	40	59	11	**339**	< 0.01	315	< 0.01	9	332	< 0.01	36	310	< 0.01	45	**339**	< 0.01	11
gdb3	275	**275**	< 0.01	84	21	35	51	9	**275**	< 0.01	259	< 0.01	13	272	< 0.01	35	258	< 0.01	55	267	< 0.01	10
gdb4	287	**287**	< 0.01	82	24	30	41	4	**287**	< 0.01	266	< 0.01	11	283	< 0.01	29	265	< 0.01	50	282	< 0.01	13
gdb5	377	371	< 0.01	150	21	48	81	20	**377**	< 0.01	346	< 0.01	17	369	< 0.01	42	339	< 0.01	49	346	< 0.01	20
gdb6	298	**298**	< 0.01	66	7	32	36	5	**298**	< 0.01	279	< 0.01	6	**298**	< 0.01	44	279	< 0.01	29	**298**	< 0.01	11
gdb7	325	**325**	< 0.01	105	27	38	62	13	**325**	< 0.01	304	< 0.01	17	317	< 0.01	32	291	< 0.01	52	309	< 0.01	22
gdb8	348	329	< 0.01	485	119	196	240	94	344	0.02	275	< 0.01	36	323	< 0.01	179	335	< 0.01	183	324	< 0.01	57
gdb9	303	**303**	0.01	538	111	182	333	81	**303**	0.02	240	< 0.01	22	289	< 0.01	159	289	< 0.01	273	286	< 0.01	64
gdb10	275	**275**	< 0.01	87	18	30	43	11	**275**	< 0.01	**275**	< 0.01	7	266	< 0.01	28	273	< 0.01	43	**275**	< 0.01	12
gdb11	395	**395**	< 0.01	305	84	86	188	26	**395**	0.01	387	< 0.01	21	381	< 0.01	65	380	< 0.01	147	387	< 0.01	27
gdb12	458	450	< 0.01	146	32	52	79	8	450	0.01	384	< 0.01	11	446	< 0.01	58	406	< 0.01	72	423	< 0.01	24
gdb13	536	**536**	< 0.01	66	12	21	60	5	**536**	< 0.01	520	< 0.01	6	531	< 0.01	25	520	< 0.01	31	532	< 0.01	5
gdb14	100	**100**	< 0.01	1	0	1	0	0	**100**	< 0.01	96	< 0.01	0	**100**	< 0.01	1	96	< 0.01	0	96	< 0.01	0
gdb15	58	**58**	< 0.01	1	0	1	0	0	**58**	< 0.01	56	< 0.01	0	**58**	< 0.01	1	56	< 0.01	0	56	< 0.01	0
gdb16	127	**127**	< 0.01	27	15	5	12	2	**127**	< 0.01	125	< 0.01	8	125	< 0.01	3	121	< 0.01	13	125	< 0.01	12
gdb17	91	87	< 0.01	16	7	1	8	0	**91**	< 0.01	**91**	< 0.01	7	87	< 0.01	1	85	< 0.01	9	**91**	< 0.01	7
gdb18	164	**164**	< 0.01	2	0	1	0	1	**164**	< 0.01	158	< 0.01	0	**164**	< 0.01	1	158	< 0.01	0	**164**	< 0.01	1
gdb19	55	**55**	< 0.01	35	11	18	13	0	**55**	< 0.01	**55**	< 0.01	6	**55**	< 0.01	15	**55**	< 0.01	11	**55**	< 0.01	6
gdb20	121	**121**	< 0.01	63	16	25	30	4	**121**	< 0.01	**121**	< 0.01	10	117	< 0.01	18	116	< 0.01	21	**121**	< 0.01	13
gdb21	156	**156**	< 0.01	51	6	17	34	2	**156**	< 0.01	154	< 0.01	5	**156**	< 0.01	20	153	< 0.01	21	154	< 0.01	5
gdb22	200	199	< 0.01	42	7	11	26	3	**200**	< 0.01	196	< 0.01	8	198	< 0.01	10	193	< 0.01	26	196	< 0.01	8
gdb23	233	**233**	< 0.01	11	0	11	0	0	**233**	< 0.01	223	< 0.01	0	**233**	< 0.01	11	223	< 0.01	0	223	< 0.01	0
best											7			5			3			10		

large scale are those of the *egl-large* dataset. These instances have 255 vertices and up to 375 required edges. As far as we know, only metaheuristics were used to solve these instances, which explains our lack of knowledge of lower bounds for them.

6.1. Exact separation

The exact separation algorithms were implemented in C++, using Windows Vista 32-bits, Visual C++ 2010 Express Edition and IBM Cplex 12.4. Tests were conducted on an Intel Core 2 Duo 2.8 GHz, with 4 GB of RAM and using only one core (IBM Cplex 12.4 uses both cores when running the branch-and-cut for the mixed-integer program). We compare the execution of both exact separation algorithms, the one from Section 2.3 proposed by Ahr [4] and our new algorithm from Section 3, executed together with the exact separation of the odd-degree cutset cuts from Section 2.2.

For both algorithms, we first apply the separation on the linear relaxation of the one-index formulation. Once the linear optimum is found, the z_e variables are then shifted to integer and the separation continues until the integer optimum is obtained. For our new exact separation, in order to model the γ limits on Eq. (34), we use a constant $\delta = 0.001$ and set $\gamma \in [0, 1-\delta]$. Results are shown in Tables 1–7.

Columns `Ins`, $|V|$, $|E_R|$, $|E|$ and $|I|$ show the name, number of vertices, required edges, total edges and number of vehicles of each instance, respectively. When the optimal value of all instances of a dataset is known, the column `Opt` displays this value. Otherwise, the known lower bounds are shown in column `LB`. For each following column `X`, $\texttt{X}_1$ shows the results obtained at the end of the first part of the experiment, when just the linear relaxation of the one-index formulation is used. Furthermore, $\texttt{X}_2$ shows the results of the complete experiment, i.e., after the solution of the integer one-index formulation. Column `Cost` shows the cost of the separation of (1) and (2) cuts, which is the same for all algorithms. For each algorithm, columns `Cap`, `Odd` and `Time` show the total number of capacity cuts, the total number of odd-degree cutset cuts and the total time in seconds. Optimal or best known values are highlighted in boldface.

From Tables 1–7, it can be observed that our algorithm performs better in nearly every instance tested. On average, it was faster for all datasets: 60.82% for *kshs*, 83.32% for *gdb*, 69.96% for *bccm*, 52.01% for *eglese*, 54.01% for *C*, 44.00% for *D*, 65.14% for *E* and 44.92% for *F*, in a total of 59.41% improvement overall.

Notice that the algorithms were not tested on the large scale instance dataset because the complete separation of the capacity cuts does not run in reasonable time without some hot-start technique, as shown next.

6.2. Dual ascent heuristic

The dual ascent heuristic was implemented using the same configuration of the exact separation algorithms. In order to show the benefit of each strategy, we tested each one separately. In addition, a complete test was also performed as follows.

Table 9
Dual ascent results for *bccm* dataset.

Ins	Opt	Dual ascent									Single cuts			Complete cuts			Connected cuts			MST cuts		
		Cost	Time	Cuts	SGL	CMP	CON	MST	Int	Time	Cost	Time	Cuts	Cost	Time	Cuts	Cost	Time	Cuts	Cost	Time	Cuts
1A	173	170	< 0.01	270	64	71	130	46	**173**	0.02	**173**	< 0.01	44	162	< 0.01	62	166	< 0.01	92	171	< 0.01	52
1B	173	171	< 0.01	278	75	82	150	40	**173**	0.01	**173**	< 0.01	44	162	< 0.01	62	167	< 0.01	116	171	< 0.01	52
1C	245	231	< 0.01	341	88	131	184	56	232	0.01	177	< 0.01	44	216	< 0.01	113	216	< 0.01	215	226	< 0.01	66
2A	227	**227**	< 0.01	211	74	78	117	36	**227**	0.01	217	< 0.01	27	221	< 0.01	83	207	< 0.01	116	225	< 0.01	23
2B	259	257	< 0.01	291	86	122	156	52	257	0.01	217	< 0.01	27	249	< 0.01	106	219	< 0.01	177	235	< 0.01	22
2C	457	449	< 0.01	541	114	227	305	93	455	0.01	282	< 0.01	39	445	< 0.01	208	360	< 0.01	273	427	< 0.01	45
3A	81	77	< 0.01	192	54	59	94	42	**81**	0.01	77	< 0.01	13	78	< 0.01	71	74	< 0.01	89	80	< 0.01	57
3B	87	85	< 0.01	265	64	82	150	49	**87**	0.01	77	< 0.01	13	84	< 0.01	78	78	< 0.01	120	84	< 0.01	57
3C	138	135	< 0.01	361	75	144	207	78	135	0.01	79	< 0.01	14	134	< 0.01	141	106	< 0.01	141	123	< 0.01	48
4A	400	395	0.01	874	205	223	573	141	396	0.03	385	< 0.01	39	382	< 0.01	216	379	0.01	494	395	< 0.01	66
4B	412	405	0.01	973	188	350	553	142	**412**	0.03	385	< 0.01	39	396	< 0.01	265	388	0.01	422	407	< 0.01	82
4C	428	419	0.01	877	212	317	518	148	424	0.03	385	< 0.01	39	413	< 0.01	346	407	0.01	460	419	< 0.01	82
4D	530	511	0.01	1200	229	458	704	189	515	0.03	385	< 0.01	39	489	< 0.01	418	494	0.01	583	472	< 0.01	94
5A	423	420	0.01	670	117	191	394	79	**423**	0.02	410	< 0.01	62	408	< 0.01	202	410	< 0.01	402	**423**	< 0.01	85
5B	446	440	0.01	678	123	200	416	92	441	0.02	412	< 0.01	62	426	< 0.01	193	404	< 0.01	317	441	< 0.01	85
5C	474	459	0.01	774	144	247	460	109	467	0.02	416	< 0.01	62	450	< 0.01	224	420	0.01	407	451	< 0.01	76
5D	577	569	0.01	887	163	340	506	105	569	0.02	430	< 0.01	62	558	< 0.01	304	503	0.01	479	528	< 0.01	90
6A	223	222	0.01	474	97	166	249	72	**223**	0.01	220	< 0.01	51	208	< 0.01	122	217	< 0.01	290	**223**	< 0.01	53
6B	233	228	0.01	483	107	151	310	74	229	0.01	220	< 0.01	51	214	< 0.01	134	221	< 0.01	279	223	< 0.01	53
6C	317	296	0.01	548	129	230	335	106	300	0.01	220	< 0.01	51	279	< 0.01	216	278	< 0.01	265	248	< 0.01	54
7A	279	278	0.01	485	129	171	268	64	**279**	0.02	**279**	< 0.01	36	264	< 0.01	130	276	< 0.01	287	278	< 0.01	38
7B	283	282	0.01	527	145	180	304	76	**283**	0.02	279	< 0.01	36	264	< 0.01	130	273	< 0.01	214	282	< 0.01	39
7C	334	323	0.01	528	172	245	252	121	323	0.02	279	< 0.01	36	301	< 0.01	236	314	0.01	357	317	< 0.01	44
8A	386	385	0.01	536	126	122	331	72	**386**	0.02	383	< 0.01	45	375	< 0.01	144	367	< 0.01	276	383	< 0.01	45
8B	395	**395**	0.01	600	136	148	379	64	**395**	0.02	383	< 0.01	45	386	< 0.01	156	382	< 0.01	297	385	< 0.01	41
8C	521	503	0.01	660	130	222	423	57	508	0.02	383	< 0.01	45	499	< 0.01	242	468	< 0.01	375	458	< 0.01	61
9A	323	319	0.02	1172	299	320	732	187	**323**	0.04	321	< 0.01	81	299	< 0.01	178	303	0.01	526	320	< 0.01	147
9B	326	320	0.02	1091	285	296	671	171	**326**	0.04	321	< 0.01	81	305	< 0.01	182	303	0.01	580	321	< 0.01	159
9C	332	325	0.02	1068	242	313	638	158	**332**	0.04	321	< 0.01	81	311	< 0.01	200	302	0.01	543	321	< 0.01	159
9D	391	374	0.02	993	267	335	567	148	377	0.04	325	< 0.01	83	359	< 0.01	261	351	0.01	555	337	< 0.01	163
10A	428	418	0.02	1096	249	250	727	150	**428**	0.04	420	< 0.01	73	400	< 0.01	218	399	0.01	542	424	< 0.01	149
10B	436	429	0.02	1240	286	321	786	172	**436**	0.05	420	< 0.01	73	406	< 0.01	247	406	0.01	686	431	< 0.01	149
10C	446	437	0.02	1294	288	354	817	188	444	0.05	420	< 0.01	73	415	< 0.01	264	416	0.01	675	439	< 0.01	149
10D	526	509	0.02	1487	314	475	931	201	517	0.05	420	< 0.01	73	490	< 0.01	388	470	0.01	788	491	< 0.01	149
best											8			9			1			20		

At each iteration of the dual ascent heuristic, we generate a cut pool using the strategies in the following order: complete cuts, single cuts, connected cuts and MST cuts. Next, the best cut is chosen from this pool, the graph is updated as described in Section 4.1 and all cuts found in this iteration are added to another pool of cuts, the resulting pool. At the end of the

Table 10
Dual ascent results for *C* dataset.

Ins	Opt	Dual ascent									Single cuts			Complete cuts			Connected cuts			MST cuts		
		Cost	Time	Cuts	SGL	CMP	CON	MST	Int	Time	Cost	Time	Cuts	Cost	Time	Cuts	Cost	Time	Cuts	Cost	Time	Cuts
C01	4105	3760	0.04	2447	695	1003	1370	381	3865	0.09	2965	< 0.01	157	3625	0.01	910	3700	0.03	1644	3750	0.01	266
C02	3135	3090	0.02	1019	283	487	506	76	3095	0.03	2340	< 0.01	43	2830	< 0.01	377	2935	0.01	534	2850	< 0.01	85
C03	2575	2490	0.01	1033	329	394	581	150	2525	0.02	1985	< 0.01	59	2265	< 0.01	352	2340	0.01	557	2455	< 0.01	96
C04	3478	3335	0.03	1600	494	660	852	247	3410	0.04	2645	< 0.01	155	3025	0.01	595	3210	0.02	1239	3305	< 0.01	205
C05	5365	5015	0.02	1671	331	691	872	292	5225	0.04	3885	< 0.01	66	4825	0.01	655	4945	0.01	973	4850	< 0.01	252
C06	2535	2445	0.01	828	283	311	457	148	2485	0.02	2155	< 0.01	46	2275	< 0.01	322	2240	0.01	480	2440	< 0.01	86
C07	4075	3815	0.02	1885	425	721	937	277	3915	0.04	2945	< 0.01	118	3485	0.01	687	3630	0.01	980	3575	< 0.01	194
C08	4090	3885	0.04	2320	360	882	1219	252	3955	0.08	2675	< 0.01	72	3635	0.01	890	3715	0.02	1264	3795	< 0.01	142
C09	5233	5105	0.06	3182	565	1228	1820	385	5165	0.11	3845	< 0.01	145	4740	0.02	1019	4925	0.03	1771	4775	0.01	222
C10	4700	4285	0.03	2446	343	921	1252	321	4515	0.08	3060	< 0.01	138	3860	0.01	763	3975	0.02	1215	4050	< 0.01	262
C11	4583	4345	0.07	3661	869	1280	2175	420	4420	0.24	3465	< 0.01	161	4015	0.02	1285	4220	0.04	1832	4005	0.01	361
C12	4209	4000	0.03	2142	418	869	1180	412	4070	0.08	3060	< 0.01	125	3830	0.01	777	3830	0.02	1436	3655	< 0.01	232
C13	2955	2795	0.01	1051	254	406	555	202	2875	0.02	2320	< 0.01	43	2590	< 0.01	395	2645	0.01	528	2615	< 0.01	126
C14	4030	3890	0.04	2483	538	1008	1345	377	3950	0.07	2990	< 0.01	140	3515	0.01	831	3735	0.02	1288	3750	< 0.01	171
C15	4912	4675	0.12	4736	1138	1866	2613	698	4810	0.19	3920	< 0.01	153	4020	0.03	1624	4480	0.06	2275	4425	0.01	430
C16	1475	1410	0.01	654	176	235	341	93	1470	0.02	1020	< 0.01	22	1220	< 0.01	201	1320	< 0.01	336	1270	< 0.01	56
C17	3555	3340	0.02	1436	321	520	825	239	3535	0.07	2380	< 0.01	55	2975	< 0.01	422	3210	0.01	861	3175	< 0.01	145
C18	5577	5415	0.09	4557	1266	1693	2447	745	5485	0.14	3730	< 0.01	213	5045	0.02	1407	5160	0.05	2410	5005	0.01	460
C19	3096	2875	0.04	2494	704	1026	1336	493	2960	0.18	2275	< 0.01	76	2600	0.01	854	2795	0.02	1296	2835	0.01	322
C20	2120	2020	0.02	1350	370	476	709	201	2035	0.03	1860	< 0.01	62	1810	< 0.01	397	1980	0.01	534	1960	< 0.01	165
C21	3960	3670	0.03	2015	423	801	1104	191	3710	0.05	2640	< 0.01	59	3435	0.01	726	3520	0.02	1210	3785	< 0.01	160
C22	2245	2225	0.03	1651	342	565	898	316	2235	0.04	1885	< 0.01	111	1825	0.01	541	2105	0.01	895	2100	< 0.01	187
C23	4032	3645	0.06	3501	797	1190	1909	577	3895	0.11	3115	< 0.01	174	3220	0.02	1008	3515	0.04	1874	3595	0.01	397
C24	3384	3240	0.05	2547	528	750	1448	374	3350	0.09	2435	< 0.01	136	2910	0.01	851	3185	0.04	1674	3265	0.01	337
C25	2310	2160	0.01	912	170	382	477	174	2290	0.02	1740	< 0.01	77	2075	< 0.01	361	2095	< 0.01	422	2095	< 0.01	122
best											0			1			14			12		

Table 11
Dual ascent results for *D* dataset.

Ins	Opt	Dual ascent									Single cuts			Complete cuts			Connected cuts			MST cuts		
		Cost	Time	Cuts	SGL	CMP	CON	MST	Int	Time	Cost	Time	Cuts	Cost	Time	Cuts	Cost	Time	Cuts	Cost	Time	Cuts
D01	3215	3115	0.04	1757	540	604	1006	196	3145	0.08	2965	< 0.01	157	2925	0.01	635	3095	0.03	1284	3120	0.01	221
D02	2520	**2520**	0.01	873	233	375	457	60	**2520**	0.03	2340	< 0.01	43	2365	< 0.01	323	2395	0.01	389	2400	< 0.01	27
D03	2065	2045	0.01	793	268	283	446	94	**2065**	0.02	1985	< 0.01	59	1875	< 0.01	244	1965	0.01	493	2045	< 0.01	66
D04	2785	2695	0.03	1875	507	650	970	276	2740	0.06	2645	< 0.01	155	2470	0.01	466	2555	0.02	883	2720	< 0.01	241
D05	3935	3815	0.03	1801	374	584	1054	271	3855	0.05	3455	< 0.01	58	3495	0.01	533	3660	0.01	840	3735	< 0.01	168
D06	2125	2115	0.01	820	194	257	456	119	**2125**	0.02	2075	< 0.01	46	1905	< 0.01	178	1915	0.01	369	2120	< 0.01	101
D07	3115	3015	0.02	1397	402	490	755	169	3015	0.04	2945	< 0.01	118	2685	0.01	498	2855	0.01	756	2975	< 0.01	155
D08	2995	2895	0.02	1269	223	587	603	114	2975	0.04	2675	< 0.01	72	2795	0.01	648	2825	0.02	808	2895	< 0.01	102
D09	4120	4095	0.05	2484	502	881	1497	254	4100	0.11	3845	< 0.01	145	3880	0.01	744	3980	0.03	1342	3970	0.01	162
D10	3340	3280	0.03	1752	226	718	902	238	3330	0.05	2950	< 0.01	138	3060	0.01	618	3110	0.02	961	3050	< 0.01	174
D11	3745	3620	0.07	3323	700	1077	1843	447	3710	0.15	3465	< 0.01	161	3275	0.02	901	3525	0.04	1611	3585	0.01	438
D12	3310	3210	0.03	1681	363	691	806	341	3260	0.06	3060	< 0.01	125	3020	0.01	610	3110	0.02	1178	3135	< 0.01	176
D13	2535	2470	0.01	982	260	313	529	158	**2535**	0.03	2320	< 0.01	43	2240	< 0.01	280	2290	0.01	448	2405	< 0.01	147
D14	3272	3170	0.03	1973	561	698	933	290	3220	0.05	2980	< 0.01	137	2795	0.01	647	3030	0.02	1023	3130	< 0.01	217
D15	3990	3935	0.11	4120	1019	1374	2407	562	3970	0.17	3865	< 0.01	151	3475	0.02	1076	3790	0.06	2140	3930	0.01	336
D16	1060	1050	< 0.01	390	101	142	200	51	1050	0.04	1020	< 0.01	22	1050	< 0.01	132	**1060**	< 0.01	175	1050	< 0.01	39
D17	2620	2600	0.01	845	247	302	431	111	**2620**	0.02	2370	< 0.01	55	2395	< 0.01	342	2540	0.01	451	2470	< 0.01	94
D18	4165	4115	0.09	4008	1229	1470	2172	649	**4165**	0.15	3730	< 0.01	213	3940	0.02	1125	3855	0.04	1684	3930	0.01	498
D19	2393	2370	0.03	2079	585	671	1106	365	2370	0.06	2215	< 0.01	82	2010	0.01	538	2175	0.02	1025	2360	0.01	282
D20	1870	1860	0.01	993	282	298	574	129	**1870**	0.02	1860	< 0.01	62	1660	< 0.01	301	1725	0.01	521	1860	< 0.01	115
D21	2985	2930	0.03	1443	379	442	856	185	2940	0.04	2640	< 0.01	59	2665	0.01	449	2655	0.01	697	2935	< 0.01	129
D22	1865	1835	0.02	1422	304	476	766	258	1845	0.04	1725	< 0.01	107	1545	0.01	383	1760	0.01	765	**1865**	< 0.01	220
D23	3114	3005	0.06	2559	633	913	1393	326	3080	0.09	2955	< 0.01	173	2640	0.01	642	2830	0.03	1423	3015	0.01	282
D24	2676	2605	0.05	2291	461	669	1327	282	2660	0.09	2435	< 0.01	136	2460	0.01	610	2510	0.03	1238	2610	0.01	213
D25	1815	**1815**	0.01	640	166	239	329	112	**1815**	0.02	1740	< 0.01	77	1680	< 0.01	232	1655	< 0.01	268	**1815**	< 0.01	106
best											1			1			4			20		

Table 12
Dual ascent results for *E* dataset.

Ins	Opt	Dual ascent									Single cuts			Complete cuts			Connected cuts			MST cuts		
		Cost	Time	Cuts	SGL	CMP	CON	MST	Int	Time	Cost	Time	Cuts	Cost	Time	Cuts	Cost	Time	Cuts	Cost	Time	Cuts
E01	4885	4660	0.06	3464	987	1222	1994	480	4785	0.14	3800	< 0.01	239	4320	0.02	1089	4435	0.03	1865	4440	0.01	424
E02	3990	3885	0.03	1790	419	661	890	299	3935	0.05	3120	< 0.01	152	3500	0.01	556	3805	0.02	1297	3495	< 0.01	177
E03	2015	2005	0.01	902	301	386	441	146	**2015**	0.02	1585	< 0.01	79	1735	< 0.01	322	1860	0.01	461	1945	< 0.01	90
E04	4155	4015	0.05	2746	874	1025	1529	445	4105	0.08	3270	< 0.01	203	3680	0.01	855	3805	0.03	1727	3840	0.01	334
E05	4585	4395	0.04	2229	466	843	1174	372	4535	0.07	3410	< 0.01	190	4175	0.01	807	4410	0.02	1363	4180	< 0.01	253
E06	2055	2045	0.02	1055	228	433	528	140	**2055**	0.03	1720	< 0.01	46	1840	< 0.01	413	1965	0.01	597	1980	< 0.01	103
E07	4155	3925	0.06	3602	668	1244	1854	621	4005	0.11	3095	< 0.01	177	3450	0.01	1022	3815	0.03	1826	3645	0.01	325
E08	4710	4335	0.06	3587	651	1192	1892	589	4555	0.13	3290	< 0.01	156	3940	0.01	1053	4265	0.03	1913	4145	0.01	325
E09	5780	5495	0.11	5175	1108	1692	3127	845	5695	0.23	4530	0.01	399	4935	0.03	1489	5240	0.07	3021	5000	0.01	417
E10	3605	3450	0.03	2005	559	788	985	379	3515	0.05	2630	< 0.01	163	3015	0.01	629	3245	0.01	941	3325	< 0.01	238
E11	4637	4375	0.07	3561	1018	1330	1893	607	4525	0.13	3575	< 0.01	270	4010	0.02	1182	4100	0.03	1547	4105	0.01	425
E12	4180	4005	0.06	3321	546	1079	1856	497	4065	0.10	3220	< 0.01	203	3630	0.01	1033	3900	0.03	1911	3755	0.01	367
E13	3345	3230	0.02	2011	397	633	1170	292	3260	0.09	2695	< 0.01	79	2875	0.01	626	2930	0.01	831	2830	< 0.01	193
E14	4115	3940	0.03	2247	682	809	1222	431	3990	0.05	3135	< 0.01	150	3465	0.01	716	3690	0.01	1065	3655	< 0.01	210
E15	4189	4095	0.08	3645	1197	1398	1910	791	4155	0.12	3470	< 0.01	225	3275	0.02	1162	3975	0.05	2191	3985	0.01	519
E16	3755	3445	0.04	2661	672	1029	1460	539	3635	0.10	2485	< 0.01	106	3115	0.01	955	3365	0.02	1350	3205	< 0.01	212
E17	2740	2450	0.02	928	239	324	460	192	2595	0.08	1890	< 0.01	57	2180	< 0.01	352	2375	0.01	593	2635	< 0.01	186
E18	3825	3720	0.06	3256	905	1147	1678	577	3785	0.09	3165	< 0.01	162	3315	0.01	995	3525	0.03	1624	3505	0.01	391
E19	3222	2895	0.06	3734	761	1359	1985	610	3030	0.11	2340	< 0.01	118	2725	0.02	1122	2915	0.04	2116	3000	0.01	439
E20	2802	2640	0.03	1857	463	675	1030	285	2735	0.05	2395	< 0.01	146	2395	0.01	641	2620	0.02	1089	2480	< 0.01	172
E21	3728	3430	0.03	1954	553	720	1116	207	3500	0.06	2585	< 0.01	124	3070	0.01	651	3375	0.02	1077	3440	< 0.01	227
E22	2470	2345	0.02	1509	325	499	859	299	2390	0.04	2090	< 0.01	149	1995	0.01	509	2285	0.01	979	2260	< 0.01	204
E23	3686	3455	0.10	4463	832	1571	2623	560	3560	0.18	2860	< 0.01	255	3095	0.03	1383	3295	0.05	2074	3275	0.01	308
E24	4001	3810	0.12	5034	1127	1866	2666	822	3930	0.22	2955	< 0.01	293	3295	0.03	1427	3510	0.07	2741	3730	0.01	496
E25	1615	1610	< 0.01	520	168	188	253	96	**1615**	0.01	1380	< 0.01	59	1420	< 0.01	147	1495	< 0.01	198	1600	< 0.01	84
best											0			0			13			12		

Table 13
Dual ascent results for *F* dataset.

Ins	Opt	Dual ascent									Single cuts			Complete cuts			Connected cuts			MST cuts		
		Cost	Time	Cuts	SGL	CMP	CON	MST	Int	Time	Cost	Time	Cuts	Cost	Time	Cuts	Cost	Time	Cuts	Cost	Time	Cuts
F01	4040	3875	0.06	3198	1007	1021	1771	534	4015	0.12	3800	< 0.01	239	3450	0.01	719	3700	0.04	1816	3825	0.01	396
F02	3300	3235	0.03	2220	519	723	1109	477	3290	0.06	3120	< 0.01	152	2820	0.01	417	3160	0.02	1083	3290	< 0.01	301
F03	1665	**1665**	0.01	803	273	309	406	118	**1665**	0.02	1585	< 0.01	79	1555	< 0.01	262	1575	0.01	512	1655	< 0.01	88
F04	3476	3415	0.05	2810	874	850	1648	482	3475	0.09	3270	< 0.01	203	3045	0.01	657	3150	0.03	1516	3345	0.01	320
F05	3605	3525	0.04	2426	557	803	1285	410	**3605**	0.10	3280	< 0.01	190	3065	0.01	606	3385	0.03	1479	3230	< 0.01	197
F06	1875	1830	0.01	735	198	242	384	133	**1875**	0.03	1720	< 0.01	46	1650	< 0.01	322	1725	0.01	558	1830	< 0.01	99
F07	3335	3205	0.05	3048	630	1093	1556	547	3315	0.11	3095	< 0.01	177	2665	0.01	868	3000	0.03	1577	3135	0.01	362
F08	3690	3635	0.06	3203	691	949	1665	599	3685	0.11	3270	< 0.01	156	3190	0.01	868	3430	0.03	1577	3405	0.01	348
F09	4730	4555	0.12	4848	1100	1526	2713	780	**4730**	0.25	4480	0.01	399	4070	0.02	1080	4385	0.07	2784	4480	0.01	665
F10	2925	2860	0.03	1743	456	596	966	296	**2925**	0.05	2600	< 0.01	163	2465	0.01	532	2610	0.02	1174	2690	< 0.01	233
F11	3835	3780	0.07	3360	941	1076	1824	503	**3835**	0.13	3575	< 0.01	270	3290	0.01	857	3535	0.04	1954	3755	0.01	397
F12	3390	3375	0.06	3051	565	863	1754	512	3385	0.11	3220	< 0.01	203	2880	0.01	743	3150	0.03	1547	3345	0.01	507
F13	2855	2720	0.02	1363	314	379	770	197	2845	0.04	2695	< 0.01	79	2425	< 0.01	410	2600	0.01	677	2770	< 0.01	265
F14	3330	3160	0.03	1959	615	644	1005	355	3320	0.04	3125	< 0.01	150	2740	0.01	592	3045	0.01	924	3160	< 0.01	241
F15	3560	3495	0.08	3668	974	1106	2062	641	**3560**	0.15	3445	< 0.01	223	2895	0.02	780	3240	0.04	1834	3530	0.01	496
F16	2725	2655	0.03	1918	520	736	992	279	**2725**	0.07	2485	< 0.01	106	2425	0.01	565	2545	0.02	934	**2725**	< 0.01	151
F17	2055	2030	0.01	669	223	243	329	90	**2055**	0.02	1890	< 0.01	57	1825	< 0.01	263	1975	0.01	465	2005	< 0.01	145
F18	3063	3035	0.06	2849	851	878	1546	523	3060	0.11	2925	< 0.01	159	2660	0.01	712	2815	0.03	1367	3025	0.01	520
F19	2500	2455	0.05	2627	676	830	1362	423	2485	0.09	2310	< 0.01	118	2120	0.01	743	2325	0.03	1280	2470	0.01	370
F20	2445	2395	0.03	1593	485	535	829	261	**2445**	0.05	2385	< 0.01	146	2070	0.01	443	2270	0.02	965	2380	< 0.01	262
F21	2930	2865	0.03	1951	574	618	1092	259	**2930**	0.05	2585	< 0.01	124	2440	0.01	450	2540	0.02	870	2885	< 0.01	267
F22	2075	1990	0.02	1486	310	497	817	290	2060	0.04	1940	< 0.01	153	1705	< 0.01	366	1900	0.01	871	1985	< 0.01	215
F23	2994	2860	0.08	3437	693	1147	1905	552	2945	0.15	2860	< 0.01	255	2565	0.02	906	2795	0.05	1884	2890	0.01	342
F24	3210	3115	0.10	3876	818	1109	2342	642	3205	0.20	2955	< 0.01	297	2680	0.02	925	2930	0.08	2708	3110	0.01	439
F25	1390	1380	< 0.01	339	127	103	177	69	**1390**	0.01	1340	< 0.01	56	1200	< 0.01	117	1275	< 0.01	143	**1390**	< 0.01	84
best											3			0			2			21		

dual ascent, we take all the cuts from the resulting pool, add them into an one-index formulation and use CPLEX 12.4 to solve it to optimality. The results of these tests are shown in Tables 8–14.

As in the previous experiments, columns `Ins`, `Opt` and `LB` show the name and the optimal value or the best known lower bound for each instance, respectively. The next 9 columns show the results regarding the full execution of the dual ascent.

Table 14
Dual ascent results for *eglese* dataset.

Ins	Opt	Dual ascent									Single cuts			Complete cuts			Connected cuts			MST cuts		
		Cost	Time	Cuts	SGL	CMP	CON	MST	Int	Time	Cost	Time	Cuts	Cost	Time	Cuts	Cost	Time	Cuts	Cost	Time	Cuts
e1-A	3548	3468	0.08	4368	836	1637	2381	593	3527	0.12	2089	< 0.01	175	3005	0.02	1416	3386	0.05	2758	3442	0.01	412
e1-B	4498	4294	0.09	5093	857	1957	2919	792	4372	0.13	2097	< 0.01	166	3831	0.02	1707	4225	0.05	3079	4272	0.01	559
e1-C	5595	5345	0.08	4643	921	1917	2678	807	5459	0.11	4363	< 0.01	192	4912	0.03	2048	5277	0.05	3039	5089	0.01	444
e2-A	5018	4834	0.09	4996	1471	2087	2533	888	4898	0.12	2702	< 0.01	280	4201	0.02	1702	4561	0.05	2915	4748	0.01	523
e2-B	6305	6165	0.08	4716	1475	2060	2468	953	6192	0.12	2931	< 0.01	266	5457	0.03	1926	5686	0.06	2765	5797	0.01	462
e2-C	8335	7752	0.09	5370	1534	2269	3027	1061	7936	0.14	3252	< 0.01	258	7309	0.03	2220	7580	0.04	2430	7394	0.01	461
e3-A	5898	5715	0.09	5163	1879	2019	2856	908	5783	0.12	3150	< 0.01	292	5012	0.03	1976	5403	0.05	2462	5499	0.01	671
e3-B	7729	7412	0.08	4599	2085	2181	2590	1019	7478	0.12	3260	< 0.01	319	6739	0.03	2085	6974	0.05	3149	6914	0.01	585
e3-C	10,244	9769	0.08	4719	2019	2233	2803	962	9955	0.13	7131	< 0.01	239	9071	0.03	2318	9487	0.04	2738	9042	0.01	464
e4-A	6408	6237	0.08	4419	1659	1726	2593	862	6242	0.11	3322	< 0.01	245	5611	0.03	1953	5861	0.04	2456	5820	0.01	469
e4-B	8935	8681	0.09	5079	1643	2154	2886	1024	8763	0.13	3612	< 0.01	248	7878	0.03	2041	7852	0.04	2692	8009	0.01	522
e4-C	11,493	10,940	0.08	5139	1666	2293	3052	1041	11,243	0.13	8091	< 0.01	262	10476	0.03	2428	10,177	0.05	3282	10,220	0.01	463
s1-A	5018	4693	0.48	14,843	1784	5979	8413	2659	4841	0.90	2476	0.01	752	4189	0.28	7996	4740	0.23	7882	4305	0.03	1257
s1-B	6388	5850	0.63	15994	1748	6634	9831	3007	6109	1.10	2759	0.01	766	5565	0.29	8102	5918	0.25	9222	5342	0.03	905
s1-C	8518	7983	0.64	20,068	1876	7269	11,834	3660	8230	1.38	4864	0.01	708	7699	0.29	8266	8113	0.25	9428	7282	0.03	1025
s2-A	9825	9411	0.56	16,026	5956	7240	8914	3045	9605	0.97	4971	0.01	807	8404	0.29	8114	9077	0.27	9277	8606	0.03	1528
s2-B	13,017	12,431	0.60	17,613	6113	7997	11,002	3367	12,745	2.05	4844	0.01	864	11,699	0.29	8242	12,306	0.26	9886	10,631	0.03	1224
s2-C	16,425	15,715	0.61	18,153	5975	7897	11,637	3821	16,059	1.74	5543	0.01	630	15,110	0.28	8055	15,517	0.28	10,828	13,190	0.03	970
s3-A	10,146	9608	0.51	14,609	5117	6753	8274	2557	9801	0.81	4730	0.01	755	8628	0.25	7089	9363	0.27	9639	8370	0.03	1256
s3-B	13,648	13,190	0.58	16,767	5490	7609	10,479	3051	13391	1.62	5067	0.01	704	12270	0.26	7300	12922	0.26	9673	10,994	0.03	1287
s3-C	17,188	16,491	0.73	18,648	5531	7683	10,903	3508	16,766	1.75	5285	0.01	633	15,843	0.28	7865	16332	0.28	10923	14613	0.03	1279
s4-A	12,144	11,721	0.56	14,912	5251	7109	8580	3000	11,881	0.99	5209	0.01	781	10,759	0.26	7539	11,357	0.26	9092	10,549	0.03	1141
s4-B	16,103	15,557	0.63	16,854	5292	7628	10,643	3331	15,800	1.37	5246	0.01	751	14,729	0.29	8117	15,106	0.27	10,388	13,570	0.03	1195
s4-C	20,430	19,767	0.60	17,697	5450	7964	11,296	4153	20,064	1.53	5660	0.01	755	19,127	0.29	7843	19,598	0.28	10,895	17,023	0.03	1444
best											0			1			17			6		

Columns `Cost` and `Time` show the solution cost and time before calling the one-index formulation. Columns `Cuts`, `SGL`, `CMP`, `CON` and `MST` show the total number of distinct cuts found overall and the total number of cuts found by each strategy, respectively. Columns `Int` and `Time` show the solution cost and time after calling the one-index formulation. Next, we show the results for each of the four strategies. Columns `Cost`, `Time`, `Cuts` show the cost, the time and the total number of cuts found.

With the view of comparing the different strategies used, we underline the best cost found among them. Moreover, the total of best costs for each strategy is shown in the last row of each table, called `best`. In addition, when a value from any `Cost` column is optimal or equal to the best known, it is highlighted in boldface. Notice that the dual ascent heuristic is capable of finding good bounds quite fast, thus generating a large number of cuts.

In Table 15, we show the results of the improvement obtained using the cuts of the dual ascent heuristic in our exact separation. In addition to the lower running time, one can notice a decrease in the separation of the cuts, more prominent in the almost total absence of separation of odd-degree cutset cuts.

As pointed before, with the use of the dual ascent heuristic, we were capable of running our exact separation for the *egl-large* instance dataset, proposed by Brandão and Eglese in 2008 [10]. The results are shown in Table 16. As shown in previous tables, columns `Ins`, $|V|$, $|E_R|$, $|E|$ and $|I|$ show the name, number of vertices, required edges, total edges and number of vehicles of each instance, respectively. The next three columns, `Cost`, `Cuts` and `Time`, show the cost, the number of cuts and the total time in seconds of the dual ascent heuristic, *without* calling the one-index formulation. The last four columns, `Cost`, `Cap`, `Odd` and `Time`, show the cost, the number of capacity cuts, the number of odd-edge cutset cuts and the total time of our exact separation using the dual ascent heuristic as hot-start. Furthermore, in contrast to what was done for the other datasets, we only performed the separation on the linear relaxation of the one-index formulation, interrupting the execution when the linear optimum was achieved. The continuous values were rounded up to the next integer.

Table 15
Improvement of the exact separation using the dual ascent heuristic as hot-start.

Dataset	Cap (%)	Odd (%)	Time (%)
kshs	100.00	100.00	34.03
gdb	100.00	100.00	36.20
bccm	91.03	99.50	48.79
C	76.30	95.00	66.76
D	83.07	99.68	64.94
E	77.16	97.11	68.71
F	91.86	99.22	69.76
eglese	69.62	97.73	32.94

6.3. Iterated local search heuristic

The ILS-RVND algorithm was coded in C++ (g++ 4.4.3) and executed in an Intel Core i5 3.2 GHz with 4 GB of RAM running Ubuntu Linux 10.04 64-bits. Only a single thread was used in our experiments. The following parameters values were selected after some preliminary experiments: (i) *MaxIter*=10, if $|E| \geq 200$, *MaxIter*=50, otherwise; (ii) *MaxIterILS*=3000, if $|E| \geq 200$, *MaxIterILS*=1500, otherwise; (iii) number of successive perturbation moves was randomly selected from the set {1,2,3}.

Table 16
Dual ascent and exact separation results for *egl-large* dataset.

Ins	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert E_R\rvert$	$\lvert I\rvert$	Dual ascent			DA + Our			
					Cost	Cuts	Time	Cost	Cap	Odd	Time
g1-a	255	347	375	20	927,232	54,246	4.201	970,495	351	196	2091.639
g1-b	255	347	375	25	1,044,780	58,934	4.542	1,085,096	323	106	2149.614
g1-c	255	347	375	30	1,153,372	59,753	4.605	1,201,028	475	147	5394.857
g1-d	255	347	375	35	1,263,641	69,159	5.336	1,325,317	557	256	6509.326
g1-e	255	347	375	40	1,384,581	73,761	5.699	1,461,469	610	266	7456.939
g2-a	255	375	375	22	1,020,539	54,511	4.298	1,061,103	278	240	1965.443
g2-b	255	375	375	27	1,129,794	57,237	4.440	1,173,286	379	254	3181.108
g2-c	255	375	375	32	1,252,044	62,286	4.701	1,295,036	416	89	3868.572
g2-d	255	375	375	37	1,360,453	67,949	5.267	1,430,267	571	46	5748.443
g2-e	255	375	375	42	1,479,110	73,621	5.725	1,557,159	574	101	7919.063

Table 17
ILS-RVND results for the *egl-large* dataset.

Ins	TSA2		RTS*		ILS-RVND				
	Best Sol.	Scaled time(s)	Best Sol.	Scaled time(s)	Best Sol.	Avg. Sol.	Avg. Gap(%)	#NI	Time(s)
g1-a	1,049,708	377.23	**1,025,765**	1213.92	<u>**1,002,264**</u>	1,010,937.4	−1.45	10	1242.08
g1-b	1,140,692	414.41	**1,135,873**	1300.48	<u>**1,126,509**</u>	1137141.5	0.11	4	1111.99
g1-c	1,282,270	439.16	**1,271,894**	1299.32	<u>**1,260,193**</u>	1,266,576.8	−0.42	10	1044.69
g1-d	1,420,126	406.53	**1,402,433**	1522.59	<u>**1,397,656**</u>	1,406,929.0	0.32	3	1012.75
g1-e	1,583,133	321.19	**1,558,548**	1556.25	<u>**1,541,853**</u>	1554220.2	−0.28	8	1011.17
g2-a	1,129,229	695.51	**1,125,602**	1519.11	<u>**1,111,127**</u>	1,118,363.0	−0.64	9	1830.11
g2-b	1,255,907	536.25	**1,242,542**	1530.78	<u>**1,223,737**</u>	1,233,720.5	−0.71	9	1671.24
g2-c	1,418,145	405.67	**1,401,583**	1727.24	<u>**1,366,629**</u>	1,374,479.7	−1.93	10	1237.03
g2-d	1,516,103	862.6	**1,516,072**	1594.61	<u>**1,506,024**</u>	1,515,119.3	−0.06	5	1141.95
g2-e	1,701,681	420.43	**1,668,348**	1701.09	<u>**1,650,657**</u>	1,658,378.1	−0.60	9	1093.28
Mean		487.90		1496.54			−0.57	7.7	1239.63

Table 18
Mean of the average gaps (%) obtained for small/medium datasets.

Dataset	TSA2	VNS	MAENS	Ant-CARP_12	GRASP	ILS-RVND
gdb	0.07	–	0.01	0.10[a]	0.11	0.02 (0.01[a])
bccm	0.13	0.07	0.17	0.11[a]	0.16	0.16 (0.17[a])
C	0.13	–	0.97	0.51[a]	–	0.44 (0.37[a])
D	0.60	–	0.79	0.34[a]	–	0.47 (0.50[a])
E	0.36	–	1.41	0.80[a]	–	1.24 (1.19[a])
F	0.90	–	1.01	0.77[a]	–	0.48 (0.48[a])
eglese	0.72	0.54	0.56	0.56[a]	0.47	0.88 (0.83[a])

[a] Mean of the average gaps between the median solutions and the BKSs.

Table 19
Mean of the average scaled times (s) obtained for small/medium datasets.

Dataset	TSA2	VNS	MAENS	Ant-CARP_12	GRASP	ILS-RVND
gdb	1.1	–	3.9	1.0	4.8	13.2
bccm	8.8	49.4	42.6	7.9	57.7	75.6
C	37.6	–	116.6	56.6	–	72.2
D	16.8	–	154.6	72.1	–	85.0
E	40.2	–	113.3	56.3	–	69.7
F	18.5	0.0	117.5	72.5	0.0	86.0
eglese	127.5	566.1	351.1	251.5	748.7	209.5

We ran the ILS-RVND heuristic 10 times for each instance and a comparison is performed with the algorithms of Brandão and Eglese (TSA2) [10], Mei et al. (RTS*) [12], Polacek et al. (VNS) [32], Tang et al. (MAENS) [33], Santos et al. [34] (Ant-CARP_12) and Usberti et al. (GRASP) [35]. These algorithms were tested in a Pentium M 1.4 GHz, Xeon 2.0 GHz, Pentium IV 3.6 GHz, Xeon 2.0 GHz, Pentium III 1.0 GHz and Core 2 Quad 3.0 GHz, respectively. In order to perform a rough comparison among the running times of the different machines, we multiplied the original computing times by a factor that denotes the ratio between the CPU clock of the machine used in the corresponding work and the CPU clock of our i5 3.2 GHz. This type of approximate comparison was also performed by other authors [12,34,35]. Hence, the approximate runtime factors for the Pentium 1.4 GHz, Xeon 2.0 GHz, Pentium IV 3.6 GHz, Pentium III 1.0 GHz, Core 2 Quad 3.0 GHz are 1.4/3.2, 2.0/3.2, 3.6/3.2, 1.0/3.2 and 3.0/3.2, respectively.

Table 17 contains the results found by ILS-RVND and the deterministic algorithms of Brandão and Eglese [10] and Mei et al. [12]. In this table, `Ins` is the name of the test-problem, `Best Sol` and `Scaled Time(s)` indicate, respectively, the best solution and the associated scaled time in seconds of the corresponding work, `Avg. Sol` represents the average solution of the 10 runs, `Avg. Gap` corresponds to the gap between the average solution found by the ILS-RVND and the best known solution, `#NI` denotes the number of improved solutions found in the 10 runs, `Time(s)` indicates the average computational time in seconds. The best known solutions are highlighted in boldface and improved solutions are underlined.

By observing the results presented in Table 17 it can be noticed that the ILS-RVND algorithm improved the Best Known Solution (BKS) of all instances. The average gap between the average solutions obtained by ILS-RVND and the BKSs was -0.57%.

The average computing time of the full execution of ILS-RVND seems to be equivalent to the algorithm of Mei et al. [12] but slower than the one of Brandão [10]. However, if we stop the execution of ILS-RVND when the algorithm obtains or improves the solutions reported by both the competitors, the average running times decrease considerably. This happens especially in the instances where the gap was negative.

Table 18 presents the mean of the average gaps between the average solutions (or single-run in case of the deterministic algorithm of Brandão) and the BKSs for the small/medium scale instances. It is important to mention that Santos et al. [34] did not report the average costs, but the median ones. For the sake of comparison, we also report the mean of the average gaps between the median solutions and the BKSs. Nevertheless, in practice, both measurements produced similar values.

From Table 18, it can be observed that ILS-RVND performance in terms of solution quality was competitive with the best known heuristic approaches available in the literature. In some datasets ILS-RVND even appear to be one of the most efficient strategies as in the case of *gdb* and *F*.

Table 19 reports the mean of the average scaled times, in seconds, obtained by ILS-RVND as well as those found by the competitors. Keeping in mind that this is only a approximate comparison, it can be seen that ILS-RVND seems slower in some datasets but, faster in others.

Finally, we also ran ILS-RVND in the *kshs* dataset and it was observed that the average gaps between the average solution and the BKSs were 0.00% for all instances, whereas the average computational time was 5.2 s.

Although ILS-RVND was originally designed to solve vehicle routing problems, the algorithm clearly outperformed, in terms of solution quality, those that dealt with large scale CARP instances. Surprisingly, ILS-RVND was capable of producing high quality solutions, even when applied to transformed instances. We believe that the employment of multiple neighborhood structures helped the algorithm to successfully explore the search space despite dealing with instances with fixed edges. It is in this context that the neighborhood structures that move or exchange arcs, i.e., Shift(2,0), Swap(2,1), Swap(2,2), Or-opt2, play a crucial role. These operators allow for generating neighbor solutions by modifying the position of the customers associated with fixed edges, but without eliminating such edges, thus avoiding the need of special procedures to prevent undesirable edge eliminations.

7. Conclusions

This work dealt the exact and heuristic approaches for the CARP with emphasis on large scale instances. We presented a new exact separation for the capacity cuts and a dual ascent heuristic that, together with a known exact separation for the odd-degree cutset cuts, were capable of producing the first lower bounds for the *egl-large* instance dataset. These two developed procedures can be very useful in any cutting plane based algorithm such as Branch-and-Cut and Branch-Cut-and-Price. Moreover, we transformed these instances to CVRP instances, using the procedure described in [25], and applied an ILS based heuristic that was capable of improving all known upper bounds. Finally, we have also reported the results obtained by the developed solution methods for well-known small/medium scale instances.

As for future work, one can extend the proposed exact separation and the dual ascent heuristic to other routing problems such as the Capacitated Vehicle Routing Problem (CVRP) or to virtually any other solution approach that relies on capacity cuts.

Acknowledgements

The contribution by Rafael Martinelli and Marcus Poggi have been partially supported by the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), grants 140849/2008-4 and 309337/2009-7.

References

[1] Wøhlk S. Contributions to arc routing. PhD thesis. Faculty of Social Sciences, University of Southern Denmark; 2005.
[2] Golden BL, Wong RT. Capacitated arc routing problems. Networks 1981 ;11:305–15.
[3] Belenguer JM, Benavent E. A cutting plane algorithm for the capacitated arc routing problem. Computers & Operations Research 2003;30:705–28.
[4] Ahr D. Contributions to multiple postmen problems. PhD thesis. Department of Computer Science, Heidelberg University; 2004.
[5] Li Lyo. Vehicle routeing for winter gritting. PhD thesis. Department of Management Science, Lancaster University; 1992.
[6] Li LYO, Eglese RW. An interactive algorithm for vehicle routeing for winter-gritting. Journal of the Operational Research Society 1996;47:217–28.
[7] Bartolini E, Cordeau JF, Laporte G. Improved lower bounds and exact algorithm for the capacitated arc routing problem. Technical Report CIRRELT-2011-33, Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation; 2011.
[8] Bode C, Irnich S. Cut-first branch-and-price-second for the capacitated arc routing problem. Technical Report LM-2011-01, Chair of Logistics Management, Gutenberg School of Management and Economics, Johannes Gutenberg University, Mainz, Germany; 2011.
[9] Martinelli R, Pecin D, Poggi M, Longo H. A branch-cut-and-price algorithm for the capacitated arc routing problem. In: Pardalos P, Rebennack S, editors. Experimental algorithms. Lecture notes in computer science, vol. 6630. Berlin/Heidelberg: Springer; 2011. p. 315–26.
[10] Brandão J, Eglese R. A deterministic tabu search algorithm for the capacitated arc routing problem. Computers & Operations Research 2008;35:1112–26.
[11] Golden BL, DeArmon JS, Baker EK. Computational experiments with algorithms for a class of routing problems. Computers & Operations Research 1983;10(1):47–59.
[12] Mei Y, Tang K, Yao X. A global repair operator for capacitated arc routing problem. IEEE Transactions on Systems, Man and Cybernetics 2009;39(3):723–34.
[13] Letchford A, Oukil A. Exploiting sparsity in pricing routines for the capacitated arc routing problem. Computers & Operations Research 2009;36:2320–7.
[14] Padberg MW, Rao MR. Odd minimum cut-sets and b-matchings. Mathematics of Operations Research 1982;7:67–80.
[15] Gomory RE, Hu TC. Multi-terminal network flows. Journal of the Society for Industrial and Applied Mathematics 1961;9(4):551–70.
[16] Edmonds J, Karp RM. Theoretical improvements in algorithmic efficiency for network flow problems. Journal of the ACM 1972;19:248–64.
[17] Fukasawa R, Longo H, Lysgaard J, Poggi de Arag ao M, Reis M, Uchoa E, et al. Robust branch-and-cut-and-price for the capacitated vehicle routing problem. Mathematical Programming 2006;106(3):491–511.
[18] Fischetti M, Lodi A. Optimizing over the first chvátal closure. Mathematical Programming 2007;110:3–20.
[19] Wong R. A dual ascent approach for steiner tree problems on a directed graph. Mathematical Programming 1984;28:271–87.
[20] Kruskal Jr. JB. On the shortest spanning subtree of a graph and the traveling salesman problem. Proceedings of the American Mathematical Society 1956;7(1):48–50.
[21] Lourenço HR, Martin OC, Stützle T. Handbook of metaheuristics, iterated local search. Kluwer Academic Publishers; 2003 Chap. p. 321–53.
[22] Penna PHV, Subramanian A, Ochi LS. An iterated local search heuristic for the heterogenous fleet vehicle routing problem. Journal of Heuristics; 2011; http://dx.doi.org/10.1007/s10732-011-9186-y. In press.
[23] Pearn WL, Assad A, Golden BL. Transforming arc routing into node routing problems. Computers & Operations Research 1987;14(4):285–8.
[24] Longo H, Poggi de Aragão M, Uchoa E. Solving capacitated arc routing problems using a transformation to the CVRP. Computers & Operations Research 2006;33:1823–7.
[25] Baldacci R, Maniezzo V. Exact methods based on node-routing formulations for undirected arc-routing problems. Networks 2006;47(1):52–60.
[26] Mladenović N, Hansen P. Variable neighborhood search. Computers & Operations Research 1997;24(11):1097–100.
[27] Subramanian A, Drummond L, Bentes C, Ochi L, Farias R. A parallel heuristic for the vehicle routing problem with simultaneous pickup and delivery. Computers & Operations Research 2010;37(11):1899–911.
[28] Kiuchi M, Shinano Y, Hirabayashi R, Saruwatari Y. An exact algorithm for the capacitated arc routing problem using parallel branch and bound method. In: Abstracts of the 1995 spring national conference of the operational research society of Japan; 1995. p. 28–9 [In Japanese].
[29] DeArmon JS. A comparison of heuristics for the capacitated Chinese postman problem. Master's thesis. University of Maryland, College Park, MD; 1981.
[30] Benavent E, Campos V, Corberan A, Mota E. The capacitated arc routing problem: lower bounds. Networks 1992;22:669–90.
[31] Beullens P, Muyldermans L, Cattrysse D, Oudheusden DV. A guided local search heuristic for the capacitated arc routing problem. European Journal of Operational Research 2003;147(3):629–43.
[32] Polacek M, Doerner K, Hartl R, Maniezzo V. A variable neighborhood search for the capacitated arc routing problem with intermediate facilities. Journal of Heuristics 2008;14:405–23.
[33] Tang K, Mei Y, Yao X. Memetic algorithm with extended neighborhood search for capacitated arc routing problems. Transactions on Evolutionary Computation 2009;13(5):1151–66.
[34] Santos L, Coutinho-Rodrigues J, Current J. An improved ant colony optimization based algorithm for the capacitated arc routing problem. Transportation Research Part B 2010;44(2):246–66.
[35] Usberti FL, França PM, França ALM. Grasp with evolutionary path-relinking for the capacitated arc routing problem. Computers and Operations Research; 2011 http://dx.doi.org/10.1016/j.cor.2011.10.014.