PUC

ISSN 0103-9741

Monografias em Ciência da Computação

nº 14/11

Improved Bounds for Large Scale Capacitated Arc Routing Problem

Rafael Martinelli

Marcus Poggi

Anand Subramanian

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22451-900

RIO DE JANEIRO - BRASIL

Monografias em Ciência da Computação, No. 14/11 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena September, 2011

Improved Bounds for Large Scale Capacitated Arc Routing Problem *

Rafael Martinelli Marcus Poggi
Anand Subramanian[1]

[1]Departamento de Engenharia de Produção, Universidade Federal da Paraíba (UFPB)

rmartinelli@inf.puc-rio.br , poggi@inf.puc-rio.br , anand@ct.ufpb.br

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 two hundred vertices and three hundred 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 apply a modified Iterated Local Search procedure to Capacitated Vehicle Routing Problem (CVRP) instances obtained through a transformation from the CARP original instances. Computational experiments were carried out on the set of large instances from Brandão and Eglese and also on the regular size set. The experiments on the latter allows evaluating the quality of the proposed lower bounds, while the ones on the former presents improved lower and upper bounds to all the set of larger instances.

Keywords: Arc Routing, Integer Programming, Dual Ascent, Capacity Cuts, Metaheuristics

Resumo. O *Capacitated Arc Routing Problem* (CARP) é ums dos problemas combinatórios mais difíceis de ser resolvido ou de encontrar soluções com alta qualidade. Isto se torna mais destacado ao se tratar de instâncias grandes. Este trabalho apura métodos que melhoram os limites inferiores e superiores de instâncias em grafos com mais de duzentos vértices e trezentas arestas, dimensões as quais, nos dias de hoje, podem ser consideradas de larga escala. Para os limites inferiores, é proposto explorar a velocidade de uma heurística *dual ascent* a fim de gerar cortes de capacidade. Estes cortes são aprimorados utilizando-se uma nova separação exata ligada à resolução de um programa linear executada após a heurística dual. Para os limites superiores, uma busca local iterada modificada é aplicada em instâncias do *Capacitated Vehicle Routing Problem* (CVRP) obtidas via uma transformação das instâncias originais do CARP. Experimentos computacionais foram feitos no conjunto de instâncias de Brandão e Eglese e também no conjunto clássico. Os experimentos neste último permite avaliar a qualidade dos limites inferiores propostos, enquanto os experimentos no primeiro conjunto apresenta melhorias tanto nos limites inferiores quanto nos superiores para todo o conjunto de instâncias grandes.

Palavras-chave: Roteamento em Arcos, Programação Inteira, Dual Ascent, Cortes de Capacidade, Metaheurísticas

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

In charge of publications:

Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC-Rio Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22451-900 Rio de Janeiro RJ Brasil
Tel. +55 21 3527-1516 Fax: +55 21 3527-1530
E-mail: bib-di@inf.puc-rio.br
Web site: http://bib-di.inf.puc-rio.br/techreports/

Contents

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 $c : E \to \mathbb{Z}^+$ and a demand $d : E \to \mathbb{Z}_0^+$. 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 the known instance datasets of the CARP. Before this work, the best known CARP lower bounds were found mainly by heuristic algorithms.

Since the work from 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 restrictions, 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 now, the larger instance solved to optimality is the *egl-s3-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 by Bartolini et al. in 2011 [7] using a cut-and-column technique combined with an exact algorithm based on 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, 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 objective of this paper is to provide a methodology capable of obtaining good lower and upper bounds for larger sets of 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 new 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.

In Section 2, we present the mathematical formulation needed for the dual ascent heuristic. In Section 3, we explain the known exact separation algorithms and we also introduce a new exact separation for the capacity cuts. In Section 4, we describe our dual ascent heuristic and how it generates cuts to hot-start the cutting plane algorithm. In Section 5, we explain the known transformation to the CVRP and the ILS heuristic. Finally, conclusions are given in Section 6.

2 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 is devised using only variables which represent the deadheading of an edge. An edge is deadheaded when a vehicle traverses this edge without servicing it. Besides that, all the vehicles are aggregated. This means that the 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 the set of all vertices except 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 sets with vertices that have 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 service the demands in $\delta_R(S) \cup E_R(S)$, $k(S) = \lceil \sum_{e \in \delta_R(S) \cup E_R(S)} d_e / Q \rceil$. 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) cuts are the same, we can represent them in the formulation using just one constraint. This can be done by introducing $\alpha(S)$, which is defined as follows:

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

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

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

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

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

The objective function (4) minimizes the cost of the deadheaded edges. Constraints (5) are the (1) and (2) cuts together. 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.

3 Exact Cut Separation

3.1 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 $\mathcal{O}(|V|.|E|^2)$, resulting in a total complexity of $\mathcal{O}(|V|^2.|E|^2)$.

3.2 Capacity Cuts Separation

3.2.1 Ahr's Exact 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 quantity of vehicles. This approach was inspired on the exact separation of the capacity cuts for the CVRP proposed by Fukasawa et al. [17]. The separation for the CARP was used to identify violated cuts on a complete formulation. 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 quantity of vehicles $k = 0...\lceil \sum_{e \in E_R} d_e / Q \rceil - 1$:

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

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

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

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

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

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

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

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

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

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

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

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

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

The objective function (7) uses the 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), (9) and (10) bind the variables s_i and h_e. Analogously, constraints (11), (12) and (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) assure that the total demand of the cut found is at least $kQ + 1$. Constraint (17) forbids the inclusion of the depot in a cut. Note that because of 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 quantity of vehicles. Depending on the instance, this number may be up to 42. Besides that, in his work, Ahr could not manage to run this separation for all CARP instances due to memory restrictions.

3.2.2 New Exact 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 to separate exactly a capacity cut considering any possible quantity 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. Besides these variables, 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 $[0, 1-\delta]$, for a small $\delta > 0$.

In addition, we use the constraints (8), (9), (10), (11), (12), (13), (14), (15) and (17) from Ahr's formulation. These constraints are required to depict a capacity cut. Following, we write our formulation omitting these constraints:

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

$$s.t. \quad \kappa = \sum_{e \in E} \frac{d_e(h_e + f_e)}{Q} + \gamma \tag{21}$$

$$s_0 = 0 \tag{22}$$

$$h_e, f_e \in \{0,1\} \qquad \forall e \in E \tag{23}$$

$$s_i \in [0,1] \qquad \forall i \in V \backslash \{0\} \tag{24}$$

$$\kappa \in \mathbb{Z}_0^+ \tag{25}$$

$$\gamma \in [0, 1-\delta] \tag{26}$$

The objective function (20) maximizes the violation of the capacity cut, while constraint (21) limits the difference between κ and the fractional value using the slack variable γ. This formulation can perform better in practice than Ahr's formulation.

3.3 Computational Experiments

The exact separation algorithms were implemented in C++, using Windows Vista 32-bits, Visual C++ 2010 Express Edition and IBM Cplex 12.2. Tests were conducted on an Intel Core 2 Duo 2.8 GHz, with 4 GB of RAM and using only one core (IBM Cplex 12.2 uses both cores when running the branch-and-cut for the mixed-integer program). For the sake of comparison, we applied the algorithms to the classical instance dataset *eglese* [5,6]. This dataset was constructed using as underlying graph regions of the road network of the county of Lancashire (UK), where cost and demands are proportional to the length of the edges and most of the instances have non-required edges.

For both algorithms, we first applied the separation on the linear relaxation of the one-index formulation. Once the linear optimum was found, the z_e variables were then shifted to integer and the separation continued until the integer optimum was obtained. For our exact separation, we used $\delta = 0.0001$. Results are shown in Table 1.

Columns `Name`, $|V|$, $|E_R|$, $|E|$ and $|I|$ show the name, number of vertices, required edges, total edges and number of vehicles of each instance, respectively. Column `LB` is the known lower bound. Column `Cost` shows the cost of the separation of (1) and (2) cuts, which is the same for all algorithms. 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 for each algorithm. As one can notice, our algorithm performs better on all instances, except for `s1-b`.

Table 1: Exact cut separation results for eglese dataset

							Ahr's Exact Sep			Our Exact Sep		
Name	$\|V\|$	$\|E\|$	$\|E_R\|$	$\|I\|$	LB	Cost	Cap	Odd	Time	Cap	Odd	Time
e1-a	77	98	51	5	3548	3527	228	2	35.974	267	2	25.873
e1-b	77	98	51	7	4498	4468	312	9	48.871	310	38	29.735
e1-c	77	98	51	10	5595	5513	305	0	48.974	349	0	38.906
e2-a	77	98	72	7	5018	4995	196	2	30.499	165	2	14.014
e2-b	77	98	72	10	6305	6273	217	18	37.753	233	59	25.008
e2-c	77	98	72	14	8335	8165	282	26	52.565	237	5	31.638
e3-a	77	98	87	8	5898	5898	164	117	42.982	117	107	11.715
e3-b	77	98	87	12	7729	7649	210	60	49.191	153	54	18.682
e3-c	77	98	87	17	10244	10138	205	8	51.440	184	19	20.672
e4-a	77	98	98	9	6408	6378	116	47	22.242	93	50	6.516
e4-b	77	98	98	14	8935	8838	153	22	33.287	117	10	11.914
e4-c	77	98	98	19	11493	11383	215	9	51.063	159	8	19.065
s1-a	140	190	75	7	5018	5010	654	0	314.841	878	0	296.828
s1-b	140	190	75	10	6388	6368	810	0	437.358	1087	0	479.417
s1-c	140	190	75	14	8518	8404	1040	0	768.406	1009	0	431.669
s2-a	140	190	147	14	9825	9737	344	266	364.681	362	229	279.811
s2-b	140	190	147	20	13017	12901	478	31	569.686	462	109	415.037
s2-c	140	190	147	27	16425	16274	561	21	1414.899	493	98	1216.596
s3-a	140	190	159	15	10146	10083	322	133	350.966	296	194	282.400
s3-b	140	190	159	22	13648	13568	497	21	560.915	391	21	315.105
s3-c	140	190	159	29	17188	17019	485	46	2390.941	447	56	1144.586
s4-a	140	190	190	19	12144	12026	295	26	284.699	227	26	116.047
s4-b	140	190	190	27	16103	16001	471	69	729.154	353	57	448.918
s4-c	140	190	190	35	20430	20256	515	22	1080.952	427	122	545.269
mean					9703	9620	379	40	407.181	368	53	259.393

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 we will discuss next.

4 Dual Ascent Heuristic

Even with the improvement on the exact separation of the capacity cuts, the separation 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:

$$\text{Max} \quad \sum_{S \subseteq V \setminus \{0\}} \alpha(S)\pi_S \tag{27}$$

$$s.t. \quad \sum_{S \subseteq V \setminus \{0\}: e \in \delta(S)} \pi_S \leq c_e \quad \forall e \in E \tag{28}$$

$$\pi_S \in \mathbb{R}_0^+ \qquad \forall S \subseteq V \setminus \{0\} \tag{29}$$

In this formulation, the variables π_S are associated with constraints (5) and constraints (28) 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. After testing different kinds of choices, a good cut for us is one with a large $\alpha(S)$ or, in the case of a tie, one with a large contribution for the objective function. The contribution of a cut can be calculated as presented in (30).

$$\sigma(S) = \alpha(S).min\{c_e : e \in \delta(S)\} \tag{30}$$

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 (28) 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 the 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 just 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.

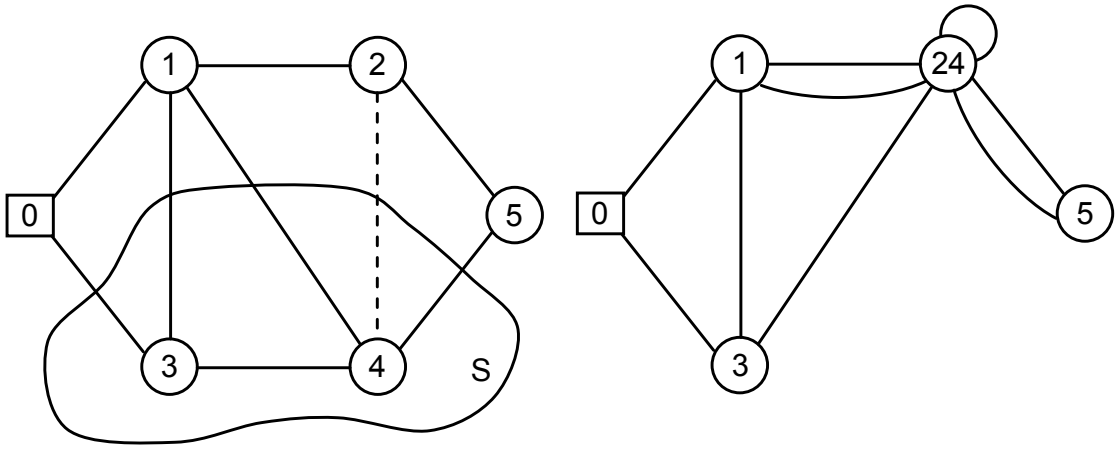

Figure 1: Example of vertex contraction: vertices 2 and 4 are contracted, becoming one vertex.

4.2.1 Simple Cuts

In the *simple cuts* strategy, we create a set of cuts $S = \{v\}$, $\forall v \in V \backslash \{0\}$, which contain only one vertex. Such vertex cannot be the depot. Notice 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. Examples of this strategy are shown in Fig. 2. This strategy takes time $\mathcal{O}(|V|)$ and generates at most $|V| - 1$ cuts.

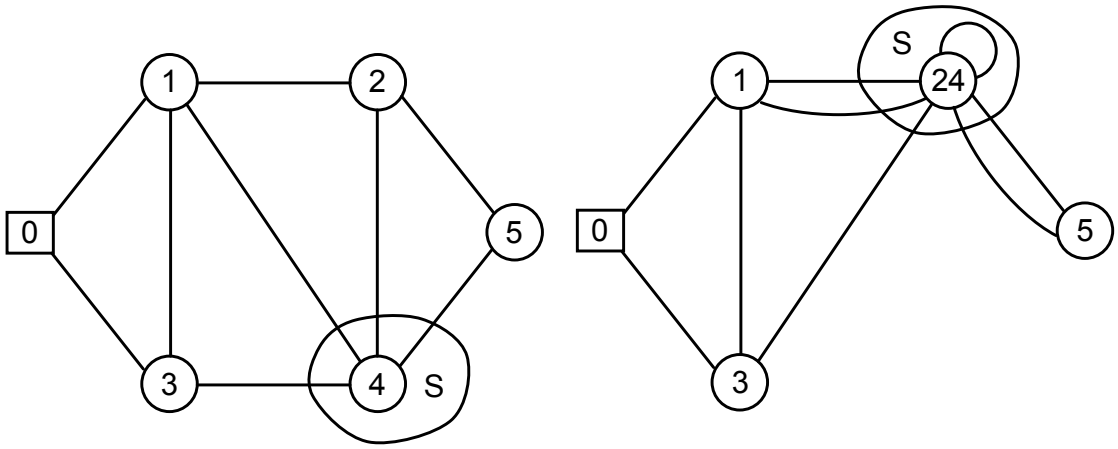

Figure 2: Examples of the Simple Cuts strategy.

4.2.2 Complete Cuts

In the *complete cuts* strategy, we create a set of cuts $S = V \backslash \{0, v\}$, $\forall v \in V$, 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. Examples of this strategy are shown in Fig. 3. This strategy takes time $\mathcal{O}(|V|^2)$ and generates at most $|V|$ cuts.

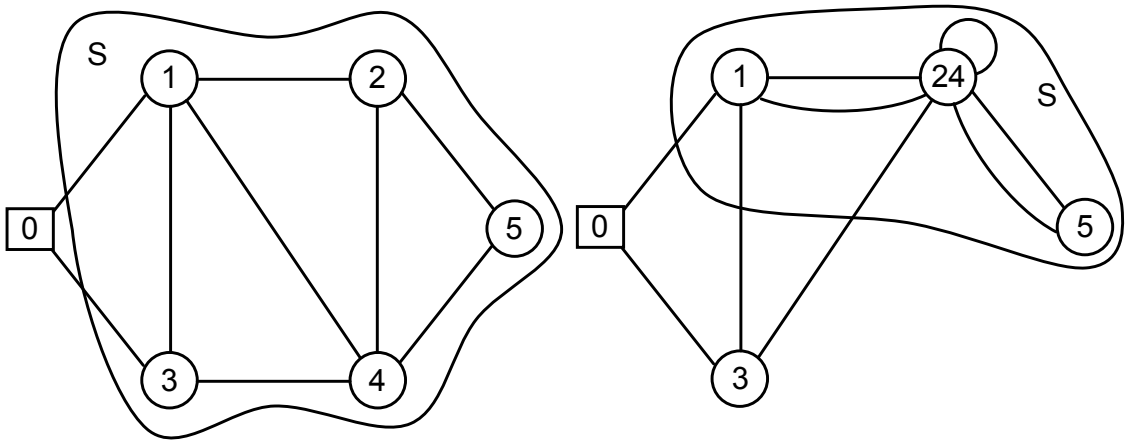

Figure 3: Examples of the Complete Cuts strategy.

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 is then generated as a cut (see Fig. 4). 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.

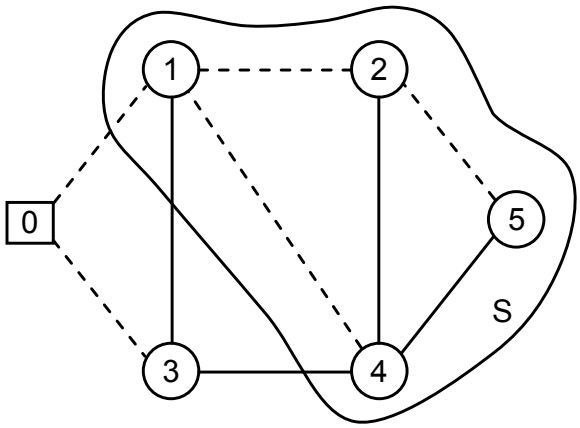

Figure 4: Example of a MST cut defined by edge $(0, 1)$. The minimum spanning tree is shown by dashed edges.

4.3 Computational Experiments

The dual ascent heuristic was implemented using the same configuration of the exact separation algorithms. As previously stated, the *eglese* instance dataset was used in order to compare the running times of the exact separation algorithms with and without the dual ascent heuristic as hot-start. In Table 2, the first 6 columns show the same information from Table 1. The next 3 columns, `Cost`, `Cuts` and `Time` show the cost, the number of cuts and the total time of the dual ascent heuristic, respectively. The last four columns, `Cost`, `Cap`, `Odd` and `Time` show the cost, total number of capacity cuts, the total number of odd-degree cutset cuts and the total time in seconds of the dual ascent together with our exact separation, respectively.

Table 2: Dual Ascent results for eglese dataset

						Dual Ascent			DA + Our			
Name	$\|V\|$	$\|E\|$	$\|E_R\|$	$\|I\|$	LB	Cost	Cuts	Time	Cost	Cap	Odd	Time
e1-a	77	98	51	5	3548	3468	4368	0.082	3527	0	0	0.392
e1-b	77	98	51	7	4498	4294	5093	0.085	4468	46	0	12.277
e1-c	77	98	51	10	5595	5345	4643	0.075	5513	29	0	15.198
e2-a	77	98	72	7	5018	4834	4996	0.089	4995	15	0	2.279
e2-b	77	98	72	10	6305	6165	4716	0.082	6273	36	0	7.614
e2-c	77	98	72	14	8335	7752	5370	0.085	8165	45	0	13.004
e3-a	77	98	87	8	5898	5715	5163	0.090	5898	50	0	8.979
e3-b	77	98	87	12	7729	7412	4599	0.084	7649	59	0	13.206
e3-c	77	98	87	17	10244	9769	4719	0.082	10138	50	1	16.257
e4-a	77	98	98	9	6408	6237	4419	0.081	6378	13	0	1.703
e4-b	77	98	98	14	8935	8681	5079	0.085	8838	22	0	5.914
e4-c	77	98	98	19	11493	10940	5139	0.082	11383	33	0	11.748
s1-a	140	190	75	7	5018	4693	14843	0.482	5010	72	0	55.607
s1-b	140	190	75	10	6388	5850	15994	0.539	6368	180	0	308.441
s1-c	140	190	75	14	8518	7983	20068	0.634	8404	116	0	221.561
s2-a	140	190	147	14	9825	9411	16026	0.561	9737	57	0	124.812
s2-b	140	190	147	20	13017	12431	17613	0.602	12901	67	0	163.043
s2-c	140	190	147	27	16425	15715	18153	0.604	16274	124	0	774.481
s3-a	140	190	159	15	10146	9608	14609	0.515	10083	101	0	186.893
s3-b	140	190	159	22	13648	13190	16767	0.572	13568	79	0	169.424
s3-c	140	190	159	29	17188	16491	18648	0.601	17019	103	0	931.588
s4-a	140	190	190	19	12144	11721	14912	0.530	12026	40	0	45.324
s4-b	140	190	190	27	16103	15557	16854	0.570	16001	95	6	388.003
s4-c	140	190	190	35	20430	19767	17697	0.594	20256	94	0	477.917
mean					9702	9293	10854	0.325	9620	64	0	164.819

These results show the improvement obtained using the cuts of the dual ascent heuristic in the 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 the exact separation for the *egl-large* instance dataset, proposed by Brandão and Eglese in 2008 [10]. The results are shown in Table 3. This table uses the same columns from Table 2, except for column `LB` because there are no known lower bounds available

for this instance dataset. Furthermore, in contrast to what was done in the *eglese* dataset, 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 3: Dual Ascent results for egl-large dataset

					Dual Ascent			DA + Our			
Name	$\|V\|$	$\|E\|$	$\|E_R\|$	$\|I\|$	Cost	Cuts	Time	Cost	Cap	Odd	Time
g1-a	255	375	347	20	927232	54246	4.219	970495	329	319	3205.015
g1-b	255	375	347	25	1044780	58936	4.514	1085097	356	160	3502.042
g1-c	255	375	347	30	1153372	59753	4.548	1201030	427	468	10752.462
g1-d	255	375	347	35	1263641	69159	5.329	1325317	546	249	9458.474
g1-e	255	375	347	40	1384581	73761	5.676	1461469	591	262	16463.889
g2-a	255	375	375	22	1020539	54511	4.287	1061103	276	195	2896.446
g2-b	255	375	375	27	1129794	59239	4.356	1173286	355	206	3798.149
g2-c	255	375	375	32	1252044	62286	4.697	1295036	284	105	4587.657
g2-d	255	375	375	37	1360453	67949	5.193	1430267	528	96	7296.234
g2-e	255	375	375	42	1479110	73621	5.694	1557159	562	73	12406.700
mean					1201555	63347	4.851	1256026	426	214	7436.707

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 modifying 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 $g(k) = \left(c^u_{ik} + c^u_{kj} - c^u_{ij}\right) - \gamma\left(c^u_{0k} + c^u_{k0}\right)$, 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 RVND procedure is composed by the following six inter-route neighborhood structures. **Shift(1,0)**, a customer k is transferred from a route r_1 to a route r_2. **Swap(1,1)**, permutation between a customer k from a route r_1 and a customer l, from 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(2,1)**, permutation of two adjacent customers, k and l, from a route r_1 by a customer k' from a route r_2. As in Shift(2,0), we consider 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 five classical Traveling Salesman Problem neighborhood structures. **Exchange**, permutation between two customers. **2-opt**, two nonadjacent 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. **Or-opt3**, three 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 feasible moves, that is, those that do not violate the capacity of the vehicle.

Two simple perturbation mechanisms were adopted. **Multiple-Swap(1,1)**, where multiple Swap(1,1) moves are performed in sequence randomly and **Multiple-Shift(1,1)**, where multiple Shift(1,1) moves are performed in sequence randomly. The Shift(1,1) consists in transferring a customer k from a route r_1 to a route r_2, whereas a customer l from r_2 is transferred to r_1. Only feasible perturbation moves are admitted.

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

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

Step 1. Apply local search using the RVND procedure.

Step 2. Let $iterILS$ be the current number of perturbations without improvements. If $iterILS \leq MaxIterILS$ 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.

5.2 Computational Experiments

The ILS-RVND algorithm was coded in C++ (g++ 4.4.3) and executed in an Intel Core i7 2.93 GHz with 8 GB of RAM running Ubuntu Linux 10.04 64-bits (kernel version 2.6.32). Only a single thread was used in our experiments. The following parameters values were selected after some preliminary experiments: (i) $MaxIter = 10$; (ii) $MaxIterILS = 1000$; (iii) number of successive Multi-Swap(1,1) moves was set to $0.5v$, where v is the number of vehicles; (iv) number of successive Multi-Shift(1,1) was randomly selected from the interval $\{0.5v, 0.6v, \ldots, 1.4v, 1.5v\}$.

We ran the ILS-RVND heuristic 10 times for each instance and a comparison is performed with the algorithms of Brandão and Eglese [10] and Mei et al. [12]. These two

algorithms were tested in an Intel Pentium M 1.4 GHz and in an Intel Xeon E5335 2.0 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 i7 2.93. Hence, the runtime factors for the Pentium 1.4 GHz and Xeon 2.0 GHz are 1.40/2.93 and 2.00/2.93, respectively.

Table 4 contains the results found by ILS-RVND and those of Brandão and Eglese [10] and Mei et al. [12]. In this table, `Name` is the name of the test-problem, $|V|$, $|E_R|$, $|E|$ indicate, respectively, the number of vertices, number of required edges and total number of edges, `Best` and `Time` indicate, respectively, the best solution and the average (or single run) scaled runtime time in seconds associated to the corresponding work, `Worst` is the worst solution found by ILS-RVND, `Avg.` represents the average solution of the 10 runs, `Gap` corresponds to the gap between the average solution found by the ILS-RVND and the best known solution.

The best solutions are highlighted in boldface and the solutions improved by the ILS-RVND algorithm are underlined. The gap was calculated using Eq. 31. Negative values indicate that there has been an improvement.

$$gap = \frac{\text{ILS-RVND\_solution} - \text{literature\_solution}}{\text{literature\_solution}} \times 100 \tag{31}$$

By observing the results presented in Table 4 it can be noticed that the ILS-RVND algorithm was found capable to improve the Best Known Solution (BKS) of all instances. The average gap between the average solution obtained by ILS-RVND and the BKSs was -0.22%.

The average computing time of the full execution of ILS-RVND seems higher then those spent by the other approaches. However, if we stop the execution of ILS-RVND when the algorithm obtains or improves the solutions reported by Brandão and Eglese [10] and Mei et al. [12], the average running times decrease considerably. This happens especially in the instances where the gap was negative.

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 allows 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.

6 Conclusions

In this work, we presented a new exact separation for the capacity cuts and a dual ascent heuristic which, together with a known exact separation for the odd-degree cutset cuts, were able to give the first lower bounds for the *egl-large* instance dataset. Furthermore, we transformed these instances to CVRP instances and applied an Iterated Local Search based heuristic in order to improve the known upper bounds. The results are shown

Table 4: ILS-RVND algorithm results for egl-large dataset

					Brandão and Eglese		Mei et al.		ILS-RVND				
Name	$\|V\|$	$\|E\|$	$\|E_R\|$	$\|I\|$	Best	Time	Best	Time	Best	Worst	Avg.	Gap	Time
g1-a	255	375	347	20	1049708	377.23	**1025765**	1213.92	**1004864**	1022527	1014930.9	-1.06	2274.7
g1-b	255	375	347	25	1140692	414.41	**1135873**	1300.48	**1129937**	1150556	1143221.7	0.65	1911.9
g1-c	255	375	347	30	1282270	439.16	**1271894**	1299.32	**1262888**	1277172	1270100.3	-0.14	2022.4
g1-d	255	375	347	35	1420126	406.53	**1402433**	1522.59	**1398958**	1417309	1409811.9	0.53	1774.5
g1-e	255	375	347	40	1583133	321.19	**1558548**	1556.25	**1543804**	1567030	1556138.5	-0.15	1971.3
g2-a	255	375	375	22	1129229	695.51	**1125602**	1519.11	**1115339**	1133787	1126561.0	0.09	2985.2
g2-b	255	375	375	27	1255907	536.25	**1242542**	1530.78	**1226645**	1244228	1237741.8	-0.39	2488.1
g2-c	255	375	375	32	1418145	405.67	**1401583**	1727.24	**1371004**	1381547	1376931.6	-1.76	2485.3
g2-d	255	375	375	37	1516103	862.60	**1516072**	1594.61	**1509990**	1528062	1520794.3	0.31	2377.9
g2-e	255	375	375	42	1701681	420.43	**1668348**	1701.09	**1659217**	1673199	1664230.2	-0.25	2224.4
mean						487.90		1496.54				-0.22	2251.6

in Table 5. Columns LB and UB present the lower bounds found using the dual ascent heuristic and our exact separation and the upper bounds found using the ILS-RVND. The column Gap shows the gap between the lower and upper bounds.

Table 5: Improved bounds for egl-large dataset

Name	$\|V\|$	$\|E\|$	$\|E_R\|$	$\|I\|$	LB	UB	Gap
g1-a	255	375	347	20	970495	1004864	3.420%
g1-b	255	375	347	25	1085097	1129937	3.968%
g1-c	255	375	347	30	1201030	1262888	4.898%
g1-d	255	375	347	35	1325317	1398958	5.264%
g1-e	255	375	347	40	1461469	1543804	5.333%
g2-a	255	375	375	22	1061103	1115339	4.863%
g2-b	255	375	375	27	1173286	1226645	4.350%
g2-c	255	375	375	32	1295036	1371004	5.541%
g2-d	255	375	375	37	1430267	1509990	5.280%
g2-e	255	375	375	42	1557159	1659217	6.151%
mean					1256026	1322265	4.907%

It is important to notice that the dual ascent together with the exact separation can be useful in any algorithm which needs to separate cuts for the CARP, like Branch-and-Cut and Branch-Cut-and-Price.

References

[1] WØHLK, S.. **Contributions to Arc Routing**. PhD thesis, Faculty of Social Sciences, University of Southern Denmark, 2005.

[2] GOLDEN, B. L.; WONG, R. T.. **Capacitated Arc Routing Problems**. Networks, 11:305–315, 1981.

[3] BELENGUER, J. M.; BENAVENT, E.. **A Cutting Plane Algorithm for the Capacitated Arc Routing Problem**. Computers & Operations Research, 30:705–28, 2003.

[4] AHR, D.. **Contributions to Multiple Postmen Problems**. PhD thesis, Department of Computer Science, Heidelberg University, 2004.

[5] LI, L. Y. O.. **Vehicle Routeing for Winter Gritting**. PhD thesis, Department of Management Science, Lancaster University, 1992.

[6] LI, L. Y. O.; EGLESE, R. W.. **An Interactive Algorithm for Vehicle Routeing for Winter-Gritting**. Journal of the Operational Research Society, 47:217–228, 1996.

[7] BARTOLINI, E.; CORDEAU, J.-F.; 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, volumen 6630 de **Lecture Notes in Computer Science**, p. 315–326. Springer Berlin / Heidelberg, 2011.

[10] BRANDÃO, J.; EGLESE, R.. **A Deterministic Tabu Search Algorithm for the Capacitated Arc Routing Problem**. Computers & Operations Research, 35:1112–1126, 2008.

[11] GOLDEN, B. L.; DEARMON, J. S.; BAKER, E. K.. **Computational Experiments with Algorithms for a Class of Routing Problems**. Computers & Operations Research, 10(1):47–59, 1983.

[12] MEI, Y.; TANG, K.; YAO, X.. **A Global Repair Operator for Capacitated Arc Routing Problem**. IEEE Transactions on Systems, Man and Cybernetics, 39(3):723–734, 2009.

[13] LETCHFORD, A.; OUKIL, A.. **Exploiting Sparsity in Pricing Routines for the Capacitated Arc Routing Problem**. Computers & Operations Research, 36:2320–2327, 2009.

[14] PADBERG, M. W.; RAO, M. R.. **Odd minimum cut-sets and b-matchings**. Mathematics of Operations Research, 7:67–80, 1982.

[15] GOMORY, R. E.; HU, T. C.. **Multi-terminal network flows**. Journal of the Society for Industrial and Applied Mathematics, 9(4):pp. 551–570, 1961.

[16] EDMONDS, J.; KARP, R. M.. **Theoretical improvements in algorithmic efficiency for network flow problems**. J. ACM, 19:248–264, April 1972.

[17] FUKASAWA, R.; LONGO, H.; LYSGAARD, J.; POGGI DE ARAGÃO, M.; REIS, M.; UCHOA, E.; WERNECK, R. F.. **Robust Branch-and-Cut-and-Price for the Capacitated Vehicle Routing Problem**. Mathematical Programming, 106(3):491–511, 2006.

[18] FISCHETTI, M.; LODI, A.. **Optimizing over the first chvátal closure**. Mathematical Programming, 110:3–20, 2007.

[19] WONG, R.. **A Dual Ascent Approach for Steiner Tree Problems on a Directed Graph**. Mathematical Programming, 28:271–287, 1984.

[20] KRUSKAL, JR., J. B.. **On the shortest spanning subtree of a graph and the traveling salesman problem**. Proceedings of the American Mathematical Society, 7(1):48–50, 1956.

[21] LOURENÇO, H. R.; MARTIN, O. C.; STÜTZLE, T.. **Handbook of Metaheuristics**, chapter Iterated Local Search, p. 321–353. Kluwer Academic Publishers., 2003.

[22] PENNA, P. H. V.; SUBRAMANIAN, A.; OCHI, L. S.. **An iterated local search heuristic for the heterogenous fleet vehicle routing problem**. Journal of Heuristics, 2011. To appear.

[23] PEARN, W.-L.; ASSAD, A.; GOLDEN, B. L.. **Transforming arc routing into node routing problems**. Computers & Operations Research, 14(4):285–288, 1987.

[24] LONGO, H.; POGGI DE ARAGÃO, M.; UCHOA, E.. **Solving Capacitated Arc Routing Problems Using a Transformation to the CVRP**. Computers & Operations Research, 33:1823–1827, 2006.

[25] BALDACCI, R.; MANIEZZO, V.. **Exact methods based on node-routing formulations for undirected arc-routing problems**. Networks, 47(1):52–60, 2006.

[26] MLADENOVIĆ, N.; HANSEN, P.. **Variable neighborhood search**. Computers & Operations Research, 24(11):1097–1100, 1997.

[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, 37(11):1899 – 1911, 2010.