

Exploiting Sparsity in Pricing Routines for the Capacitated Arc Routing Problem

Adam N. Letchford* and Amar Oukil

*Department of Management Science, Lancaster University,
Lancaster LA1 4YW, United Kingdom*

October 2007. Revised May 2008, August 2008.

Abstract: The *Capacitated Arc Routing Problem* (CARP) is a well-known and fundamental vehicle routing problem. A promising exact solution approach to the CARP is to model it as a set covering problem and solve it via branch-cut-and-price. The bottleneck in this approach is the pricing (column generation) routine. In this paper, we note that most CARP instances arising in practical applications are defined on *sparse graphs*. We show how to exploit this sparsity to yield faster pricing routines. Extensive computational results are given.

Keywords: arc routing, column generation, branch-cut-and-price.

*Corresponding author. Fax: +44 (0)1524 844885.

E-mail address: a.n.letchford@lancaster.ac.uk (A.N. Letchford).

1 Introduction

In most vehicle routing problems, the customers are located at the vertices of a network. In *Arc Routing Problems* (ARPs), however, the demands are located along the edges. Although this might seem like a minor change, ARPs have a rather different structure to standard vehicle routing problems and specialised methods have been devised for them — see for example the edited volume by Dror [8] and the references therein.

This paper is concerned with one of the most well-known ARPs, the *Capacitated Arc Routing Problem* (CARP), which is defined as follows. We are given an undirected graph $G = (V, E)$, a set $E_R \subseteq E$ of *required* edges, and a specified depot vertex. Each edge has a *traversing cost* $c_e > 0$, and each required edge has a *demand* $q_e > 0$. An unlimited fleet of identical vehicles, each of capacity $Q > 0$, is available. The task is to find a minimum cost set of vehicle routes, each starting and ending at the depot, such that each required edge is serviced exactly once.

The CARP was introduced by Golden & Wong [15], who showed that even finding a 3/2-approximate solution is \mathcal{NP} -hard. Comprehensive surveys of heuristics, lower bounds and exact algorithms for the CARP, up to the year 2000, can be found in Dror [8]. For surveys of more recent work, see Ahr [1], Wøhlk [29] and Section 2 of this paper.

At present, the most promising approach to solving the CARP to proven optimality is that of Longo *et al.* [24]. Their algorithm consists of two steps. First, shortest-path computations are used to transform the CARP into its node routing counterpart, the *Capacitated Vehicle Routing Problem* (CVRP). Then, the CVRP instance is solved with the algorithm of Fukasawa *et al.* [12]. This algorithm is based on the formulation of the CVRP as a set covering problem, which is then solved by branch-cut-and-price.

In our view, it seems odd that transforming the CARP into the CVRP should be necessary to obtain good results. Indeed, most CARPs arising in practice (and most standard test instances) are defined on *sparse* graphs, such as road networks. The transformation used by Longo *et al.* leads to a CVRP instance defined on a *complete* graph. This seems to be an unnecessary ‘squaring’ of the amount of data. (We elaborate on this point in Subsection 2.1.)

Our ultimate goal is to develop a branch-cut-and-price algorithm for the CARP that fully exploits graph sparsity. In this paper, however, we concentrate on a key component of such an algorithm: the *pricing* (column generation) routine. We will show that, when G is sufficiently sparse, pricing can be performed more quickly. Moreover, unlike in the case of the CVRP, we can price over so-called *elementary routes*, which leads to a significant improvement in the lower bound.

The remainder of the paper is structured as follows. In Section 2, we review the existing exact approaches to the CARP and point out their

strengths and weaknesses. In Sections 3 and 4, we describe our new pricing routines, for non-elementary and elementary routes, respectively. In Section 5, we give extensive computational results. Some concluding remarks are made in Section 6.

Throughout the paper we use the following (fairly standard) notation and terminology. For any $S \subset V$, we denote by $E(S)$ (respectively, $\delta(S)$) the set of edges with both end-vertices (exactly one end-vertex) in S . We also define $E_R(S) = E_R \cap E(S)$ and $\delta_R(S) = E_R \cap \delta(S)$. We let V' denote the set of non-depot vertices and V_R denote the set of vertices that are incident on at least one required edge. Finally, *deadheading* an edge $e \in E$ means traversing e without servicing it. (Note that a non-required edge can only be traversed if it is deadheaded.)

2 Existing Integer Programming Approaches

In this section we briefly review the existing integer programming approaches to the CARP.

2.1 The node routing approach

In the *node routing* approach, the given CARP instance is first transformed into an instance of the standard CVRP. The transformation involves the computation of all-pairs shortest paths in G . The resulting CVRP instance can then be solved by any CVRP algorithm. This approach was first suggested by Pearn *et al.* [27], and explored further by Letchford [21], Dror & Langevin [9], Longo *et al.* [24] and Baldacci & Maniezzo [2]. It benefits from the fact that some excellent exact algorithms exist for the CVRP, such as the branch-and-cut algorithm of Lysgaard *et al.* [25] and the branch-cut-and-price algorithm of Fukasawa *et al.* [12].

Although the node routing approach has given rather good results so far, it does have some drawbacks. For example:

1. The transformation leads to a CVRP instance defined on a complete graph with $2|E_R| + 1$ vertices (or even $3|E_R| + 1$ in the case of Pearn *et al.* [27]). If the original graph G is sparse, yet $|E_R|$ is large, this leads to a huge increase in memory requirements.
2. The graph G may have some kind of special structure that could be exploited algorithmically. (E.g., if G is a tree, then one can adapt the approach of Labbé *et al.* [20] to the CARP.) Any structure of this kind is lost in the transformation.
3. The approach can suffer seriously from *symmetry*. Suppose, for example, that the graph G is a star, all edges are required, and only one vehicle is needed. When any of the approaches that we describe in the

following three subsections are used, there is only one optimal solution: deadhead each edge exactly once. The CVRP instance resulting from the transformation, on the other hand, has $|E|!$ alternative optimal solutions.

In our opinion, approaches that can avoid this transformation and work directly on the original graph G merit serious consideration.

2.2 The two-index approach

Suppose that an upper bound, say m , is known on the number of vehicles to be used in the optimal solution. Under these circumstances, it is possible to formulate the CARP using $\mathcal{O}(m|E|)$ variables, which looks desirable when m is small and G is sparse. To our knowledge, this approach was first suggested by Welz [28]. However, Belenguer & Benavent [3] were the first to explore the approach in depth.

In the approach of Belenguer & Benavent [3], there is a binary variable x_{ek} for each $e \in E_R$ and for $k = 1, \dots, m$, taking the value 1 if and only if the edge e is serviced by that vehicle. There is also a general integer variable y_{ek} for each $e \in E$ and for $k = 1, \dots, m$, representing the number of times the edge e is deadheaded by the given vehicle.

Valid inequalities and separation algorithms for the two-index formulation have been devised by Belenguer & Benavent [3], Letchford [21] and Ahr [1]. Computational results presented in [1, 3] indicate that the two-index approach works very well when m is small (less than five). However, when m is larger, it becomes unattractive. This is due in part to the large number of variables, but also to the presence of symmetry: there always exist at least $m!$ optimal solutions. Ghiani *et al.* [13] recently proposed some ‘symmetry-breaking’ constraints, in an attempt to alleviate this latter problem.

2.3 The one-index approach

In what we call the *one-index* approach, there is only one variable for each edge. This approach was originally proposed for a sparse variant of the CVRP by Fleischmann [10]. It was adapted to the CARP independently by Letchford [21] and Belenguer & Benavent [4].

In the case of the CARP, the variables are defined as follows. For each $e \in E$, let z_e be a general integer variable representing the total number of times e is deadheaded, summed over all vehicles. These variables can be regarded as *aggregated* versions of the y_{ek} variables in the two-index approach, in the sense that $z_e = \sum_{k=1}^m y_{ek}$.

Although there is no known way to formulate the CARP as an integer program using only these aggregated variables, it is easy to derive valid inequalities that must be satisfied by feasible solutions. Letchford [21] and

Belenguer & Benavent [4] independently proposed the following *capacity* inequalities:

$$\sum_{e \in \delta(S)} z_e \geq 2k(S) - |\delta_R(S)| \quad (S \subset V'),$$

where $k(S)$ is any lower bound on the number of vehicles that must enter S . They also proposed the following *R-odd cut* inequalities:

$$\sum_{e \in \delta(S)} z_e \geq 1 \quad (S \subset V' : |\delta_R(S)| \text{ odd}).$$

Belenguer & Benavent [4] also presented a third class of inequalities, called *disjoint path* inequalities. Since the description of these inequalities is rather complicated, we do not go into details.

Effective exact and heuristic separation algorithms for the capacity, *R-odd cut* and *disjoint path* inequalities can be found in Belenguer & Benavent [4] and Ahr [1]. Their computational results show that the one-index approach gives excellent lower bounds extremely quickly. Unfortunately, one cannot build a full exact algorithm for the CARP using the z variables alone, because it is strongly \mathcal{NP} -hard to decide whether a vector $z \in \mathbb{Z}_+^{|E|}$ represents a feasible solution to the CARP.

2.4 A direct set covering approach

Gómez-Cabrero *et al.* [16] proposed formulating the CARP as a set covering problem directly, without any transformation to the CVRP. We refer the reader to Dror & Langevin [9] and Bramel & Simchi-Levi [6] for introductions to the set covering approach to vehicle routing and associated concepts such as column generation and branch-and-price.

Let Ω denote the set of all possible feasible routes for a single vehicle. We define a binary variable λ_r for each $r \in \Omega$, taking the value 1 if and only if route r is used. We also denote by c_r the cost of route r . The set covering formulation takes the following standard form:

$$\begin{aligned} \min \quad & \sum_{r \in \Omega} c_r \lambda_r \\ \text{s.t.} \quad & \sum_{r \in \Omega} a_{re} \lambda_r \geq 1 \quad (e \in E_R) \\ & \lambda_r \in \{0, 1\} \quad (r \in \Omega), \end{aligned} \tag{1}$$

where a_{re} denotes the number of times edge e is serviced (not merely traversed) by route r .

Since this formulation has an exponential number of variables, one must solve its LP relaxation via column generation. Unfortunately, the pricing (column generation) subproblem is itself easily shown to be strongly \mathcal{NP} -hard. Gómez-Cabrero *et al.* [16] therefore enlarge the set of columns by permitting *non-elementary* routes, i.e., routes in which customers may be

serviced more than once. When the problem is relaxed in this way, the pricing subproblem can be solved in pseudo-polynomial time by dynamic programming.

Gómez-Cabrero *et al.* [16] strengthen the LP relaxation by adding the capacity and R -odd cut inequalities mentioned above. To link the z variables with the λ variables, they use the identities:

$$z_e = \sum_{r \in \Omega} a'_{re} \lambda_r \quad (e \in E),$$

where a'_{re} denotes the number of times route r deadheads edge e . (A similar approach was used for the CVRP by Fukasawa *et al.* [12].)

Since the resulting extended formulation has an exponential number of both variables and constraints, Gómez-Cabrero *et al.* [16] use both row and column generation (cut-and-price) to solve its LP relaxation. The resulting lower bounds are much better than those obtained by pricing alone, and slightly better than those obtained by cutting alone.

The pricing subroutine of Gómez-Cabrero *et al.* [16] exploits the sparsity of G only to some extent. In the next section, we present alternative pricing subroutines that fully exploit sparsity.

3 Pricing with Non-Elementary Routes

In this section we give algorithms for pricing, when non-elementary routes are permitted, that exploit graph sparsity. In the first subsection, we consider exact pricing, and in the second, we turn to heuristic pricing.

3.1 Exact pricing

If we transform the CARP into the standard CVRP, as done in Longo *et al.* [24], then pricing with non-elementary routes can easily be performed in $\mathcal{O}(Q|E_R|^2)$ time by dynamic programming (see, e.g., Bramel & Simchi-Levi [6]). The pricing algorithm described by Gómez-Cabrero *et al.* [16] is a little more sophisticated, and can be shown to run in $\mathcal{O}(Q|V_R|^2)$ time. However, both approaches require all-pairs shortest paths to be computed in G in a pre-processing stage.

We now show that pricing with non-elementary routes can in fact be performed in $\mathcal{O}(Q(|E| + |V| \log |V|))$ time without any all-pairs shortest path computations. Like the existing algorithms, the new algorithm is based on dynamic programming. However, we use Dijkstra's single-source shortest path algorithm as a subroutine to speed up certain computations.

To explain our new algorithm, we will need some notation. For each required edge $e \in E_R$, let π_e be the dual price of the associated constraint (1) in the optimal solution to the current restricted master LP. For $i \in V$ and

$q = 0, \dots, Q$, let $f(i, q)$ denote the cumulative reduced cost of the best path starting at the depot and ending at vertex i , having delivered a cumulative load of q . (If no such path exists, $f(i, q) = \infty$.) Now, for $i \in V_R$ and $q = 0, \dots, Q$, let $g(i, q)$ denote the cumulative reduced cost of the best path starting at the depot and ending at vertex i , having delivered a cumulative load of q , subject to the restriction that it has just serviced an edge when it arrives at i . (Again, if no such path exists, $g(i, q) = \infty$.) Note that, by definition, $g(i, q) \geq f(i, q)$ for all $i \in V_R$ and for $q = 0, \dots, Q$.

The algorithm is as follows:

Initialisation: set all $f(i, q)$ and $g(i, q)$ values to ∞ .

For $q = 0, \dots, Q - 1$ do:

(*) Call Dijkstra's algorithm to compute $f(i, q)$ for all $i \in V$.

For each $\{i, j\} \in E_R$ such that $q + q_{ij} \leq Q$ do:

If $f(i, q) + c_{ij} - \pi_{ij} < f(j, q + q_{ij})$

set $g(j, q + q_{ij}) := f(i, q) + c_{ij} - \pi_{ij}$.

If $f(j, q) + c_{ij} - \pi_{ij} < f(i, q + q_{ij})$

set $g(i, q + q_{ij}) := f(j, q) + c_{ij} - \pi_{ij}$.

Call Dijkstra's algorithm to compute $f(i, Q)$ for all $i \in V$.

For $q = 1, \dots, Q$ do:

If $f(0, q) < 0$, output the corresponding column.

The steps of this algorithm are self-explanatory, with the exception of the step marked with an asterisk. When $q = 0$, it suffices to call Dijkstra's algorithm on G with the depot as source (since the vehicle will always take a shortest path from the depot to the node at which servicing begins). When $q \geq 1$, however, things are more complicated. We have already computed $g(i, q)$ for all $i \in V_R$, and we wish to use that information to compute $f(i, q)$ for all $i \in V$. Of course, we have:

$$f(i, q) := \min_{j \in V_R} \{g(j, q) + sp(j, i)\},$$

where $sp(j, i)$ denotes the cost of the shortest path from j to i in G (and by convention $sp(i, i) = 0$). Computing all-pairs shortest paths is, however, precisely what we want to avoid. Instead, we use Dijkstra's algorithm in the following way. We construct a copy of G , with edge-costs c_e as usual, and add a dummy vertex, say v^* . For each $i \in V_R$, we then add a directed arc from v^* to i with cost equal to $g(i, q)$. By definition, $f(i, q)$ is then equal to the cost of the shortest path from v^* to i in this auxiliary graph. Thus, it suffices to call Dijkstra's algorithm once with v^* as source. (The presence of negative $g(i, q)$ values does not cause any problem, since one can simply add a large positive constant to the cost of the directed arcs, and then subtract it from the cost of the shortest paths afterwards.)

We therefore have the following result:

Theorem 1 *For the CARP, pricing with non-elementary routes permitted can be performed in $\mathcal{O}(Q(|E| + |V| \log |V|))$ time.*

Proof. We call Dijkstra’s algorithm once on G , and Q times on a graph with $|V| + 1$ vertices, $|E|$ edges and $|V|$ arcs. If we use a Fibonacci heap implementation of Dijkstra’s algorithm (Fredman and Tarjan [11]), this takes $\mathcal{O}(Q(|E| + |V| \log |V|))$ time. The remainder of the algorithm, scanning the required edges, takes only $\mathcal{O}(Q|E_R|)$ time. \square

We close this subsection with a remark. It usually happens that there are several required edges with negative reduced cost, i.e., edges for which $\pi_e > c_e$. When this occurs, it is possible for our pricing algorithm to generate a non-elementary route in which a small number of edges are serviced many times. This is analogous to the well-known phenomenon of *k-cycles* in the VRP literature; see for example Irnich & Villeneuve [18]. Just as in the case of the VRP, one can modify our pricing algorithm to prevent *k-cycles*, but this would be at the expense of increasing the running time.

3.2 Heuristic pricing

Although the pricing algorithm described above exploits graph sparsity, it can still be rather slow for large CARP instances. This led us to devise fast *heuristics* for pricing, which can be called before invoking the exact pricing routines.

Intuitively, one would expect the columns that are basic at the optimal solution of the master LP to correspond to routes that include very little deadheading. Our first pricing heuristic tries to exploit this observation, by pricing over the (not necessarily elementary) routes that have the following ‘three-phase’ structure:

1. The vehicle departs from the depot and deadheads to the first edge to be serviced.
2. The vehicle services one or more required edges without any deadheading taking place.
3. The vehicle departs from the last edge serviced and deadheads back to the depot.

Phase 1 or 3, or both, can of course be ‘empty’, if the vehicle starts or ends by servicing a required edge that is incident on the depot.

It turns out that pricing over the routes with this structure can easily be performed in $\mathcal{O}(Q|E_R|)$ time, assuming that we have already computed, for each $i \in V_R$, the cost c_i^* of the shortest path from the depot to i . The

algorithm is as follows:

Set $f(i, q) = \infty$ for all $i \in V_R$ and for $q = 1, \dots, Q$. Set $f(i, 0) = c_i^*$ for all $i \in V_R$.

For $q = 0, \dots, Q - 1$ do:

For each $\{i, j\} \in E_R$ such that $q + q_{ij} \leq Q$ do:

If $f(i, q) + c_{ij} - \pi_{ij} < f(j, q + q_{ij})$ set $f(j, q + q_{ij}) := f(i, q) + c_{ij} - \pi_{ij}$.

If $f(j, q) + c_{ij} - \pi_{ij} < f(i, q + q_{ij})$ set $f(i, q + q_{ij}) := f(j, q) + c_{ij} - \pi_{ij}$.

For $q = 1, \dots, Q$ do:

For each $i \in V_R$ do:

If $f(i, q) + c_i^* < 0$, output the corresponding column.

This heuristic is very fast in practice and extremely easy to implement, but it suffers from an important drawback. If the subgraph of G induced by the required edges has many connected components, then many ‘useful’ routes may be lost. (The heuristic permits the vehicle to pass through at most one such component while servicing.)

Our second pricing heuristic attempts to get around the above-mentioned limitation, by permitting a limited amount of deadheading to occur in phase 2. Specifically, we ‘pretend’ that one unit of demand is consumed each time an edge is deadheaded. (Putting this another way, we insist that the total demand serviced by the vehicle, plus the number of edges deadheaded in phase 2, must not exceed Q .) One can price over the resulting set of routes in $\mathcal{O}(Q|E|)$ time, simply by inserting the following steps in the above algorithm immediately after the start of the ‘for q ’ loop:

For each $\{i, j\} \in E$ do:

If $f(i, q) + c_{ij} < f(j, q + 1)$ set $f(j, q + 1) := f(i, q) + c_{ij}$.

If $f(j, q) + c_{ij} < f(i, q + 1)$ set $f(i, q + 1) := f(j, q) + c_{ij}$.

Although slightly slower and more complicated than the first heuristic, this second heuristic is still very fast in practice and much easier to implement than the exact pricing algorithm described in Section 3.

4 Pricing with Elementary Routes

Although pricing with elementary routes is strongly \mathcal{NP} -hard, it turns out that one can often solve it exactly in a reasonable amount of time when G is sparse. The key is to formulate the pricing problem as a mixed-integer program (MIP) with only $\mathcal{O}(|E|)$ variables and constraints, as we explain in the following subsections.

4.1 A directed MIP formulation

We first present a MIP formulation that uses directed flow variables. For each $\{i, j\} \in E_R$, define two binary variables x_{ij} and x_{ji} . The variable x_{ij} takes the value 1 if and only if edge $\{i, j\}$ is serviced from i to j , and similarly for x_{ji} . Then, for each $\{i, j\} \in E$, define two binary variables y_{ij} and y_{ji} , representing the number of times $\{i, j\}$ is *traversed* (whether servicing or not) in either direction. (It is easy to show that there always exists at least one optimal solution to the pricing subproblem such that no edge is traversed more than once in any given direction.) Finally define for each $\{i, j\} \in E$ two continuous variables f_{ij} and f_{ji} . The variable f_{ij} takes the value zero if $y_{ij} = 0$, but if $y_{ij} = 1$ it represents the remaining load on the vehicle when the vehicle arrives at j from i . The variable f_{ji} is defined similarly.

Since we are working with directed variables, it is helpful to define $\delta^+(i)$ and $\delta^-(i)$, the set of directed arcs leaving and entering vertex i , respectively. We define $\delta_R^+(i)$ and $\delta_R^-(i)$ analogously. Letting 0 denote the depot vertex, the pricing subproblem then amounts to minimising

$$\sum_{\{i,j\} \in E} c_{ij}(y_{ij} + y_{ji}) - \sum_{\{i,j\} \in E_R} \pi_{ij}(x_{ij} + x_{ji})$$

subject to the following constraints:

$$x_{ij} + x_{ji} \leq 1 \quad (\{i, j\} \in E_R) \quad (2)$$

$$y_{ij} \geq x_{ij}, y_{ji} \geq x_{ji} \quad (\{i, j\} \in E_R) \quad (3)$$

$$y(\delta^-(i)) = y(\delta^+(i)) \quad (i \in V') \quad (4)$$

$$f(\delta^+(i)) = f(\delta^-(i)) - \sum_{\{i,j\} \in \delta_R^+(i)} q_{ij}x_{ij} \quad (i \in V') \quad (5)$$

$$f(\delta^+(0)) - f(\delta^-(0)) + \sum_{(0,j) \in \delta_R^+(0)} q_{0j}x_{0j} \leq Q \quad (6)$$

$$f_{ij} \leq Qy_{ij}, f_{ji} \leq Qy_{ji} \quad (\{i, j\} \in E \setminus E_R) \quad (7)$$

$$f_{ij} \leq Qy_{ij} - q_{ij}x_{ij}, f_{ji} \leq Qy_{ji} - q_{ji}x_{ji} \quad (\{i, j\} \in E_R) \quad (8)$$

$$f_{ij}, f_{ji} \geq 0 \quad (\{i, j\} \in E)$$

$$x_{ij}, x_{ji} \in \{0, 1\} \quad (\{i, j\} \in E_R)$$

$$y_{ij}, y_{ji} \in \{0, 1\} \quad (\{i, j\} \in E).$$

Constraints (2) ensure that required edges are serviced at most once and constraints (3) ensure that edges are traversed when they are serviced. Constraints (4) ensure that the vehicle leaves each vertex as many times as it enters. Constraints (5) ensure that servicing reduces vehicle load. The constraint (6) ensures that the total demand on the route is no more than Q . Constraints (7) and (8) restrict vehicle load *en route*. The remaining constraints are trivial non-negativity and binary integrality conditions.

This formulation, which is easily seen to be valid, has only $\mathcal{O}(|E|)$ variables, constraints and non-zero constraint coefficients. Moreover, its LP relaxation is reasonably strong, as indicated by the following proposition:

Proposition 1 *The feasible region of the LP relaxation of the above MILP formulation satisfies the following inequalities:*

$$Qy(\delta^-(S)) \geq \sum_{\{i,j\} \in E_R(S) \cup \delta_R(S)} q_{ij}(x_{ij} + x_{ji}) \quad (\forall S \subset V'). \quad (9)$$

Proof. We use the projection technique of Gouveia [17]. Summing the constraints (5) over all $i \in S$, and re-arranging gives:

$$f(\delta^-(S)) = f(\delta^+(S)) + \sum_{(i,j) \subset S} q_{ij}x_{ij} + \sum_{i \in S, j \in V \setminus S} q_{ij}x_{ij}.$$

Since the f variables are non-negative we have:

$$f(\delta^-(S)) \geq \sum_{(i,j) \subset S} q_{ij}x_{ij} + \sum_{i \in S, j \in V \setminus S} q_{ij}x_{ij}.$$

The result then follows from the bounds (7) and (8). \square

Note that the inequalities (9) are exponential in number. That is, a linear number of constraints involving the f variables implies an exponential number of constraints in the subspace defined by the x and y variables.

We remark that the constraints (5) and (6) also imply the ‘obvious’ valid inequality $\sum_{\{i,j\} \in E_R} q_{ij}(x_{ij} + x_{ji}) \leq Q$.

Although the formulation has these nice properties, there are two simple ways to improve it. First, one can add the constraint $y(\delta^+(0)) = 1$, since any feasible route in which $y(\delta^+(0)) > 1$ can be decomposed into two or more simpler routes. Second, one can fix the variables x_{ij} and x_{ji} to zero whenever $\pi_{ij} = 0$. Note that, to use this second improvement, one must modify the formulation dynamically during the course of the column generation algorithm.

4.2 Undirected and mixed MIP formulations

The above directed MIP formulation suffers from a high degree of symmetry (many feasible solutions of the same cost). One can eliminate this symmetry, and reduce the number of variables, by using undirected x and y variables. However, to ensure that vertex degrees are even, one then needs to use an additional general integer variable for each (non-depot) vertex. This leads

to the following formulation, which we call the *undirected* formulation:

$$\begin{array}{ll}
\text{Min} & \sum_{e \in E} c_e y_e - \sum_{e \in E_R} \pi_e x_e \\
\text{S.t.} & y_e \geq x_e \quad (e \in E_R) \\
& y(\delta(i)) = 2z_i \quad (i \in V') \\
& f(\delta^+(i)) = f(\delta^-(i)) - \frac{1}{2} \sum_{e \in \delta_R(i)} q_e x_e \quad (i \in V') \\
& f(\delta^+(0)) - f(\delta^-(0)) + \frac{1}{2} \sum_{e \in \delta_R(0)} q_e x_e \leq Q \\
& f_{ij}, f_{ji} \leq \frac{Q}{2} y_{ij} \quad (\{i, j\} \in E \setminus E_R) \\
& f_{ij}, f_{ji} \leq \frac{1}{2} (Q y_{ij} - q_{ij} x_{ij}) \quad (\{i, j\} \in E_R) \\
& f_{ij}, f_{ji} \geq 0 \quad (\{i, j\} \in E) \\
& x_e \in \{0, 1\} \quad (e \in E_R) \\
& y_e \in \{0, 1, 2\} \quad (e \in E) \\
& z_i \in \mathbb{Z}_+ \quad (i \in V').
\end{array}$$

We remark that, although this gives a valid formulation of the pricing problem, the f variables no longer have any natural physical interpretation.

By analogy with the directed formulation, one can strengthen the undirected formulation by adding the constraint $y(\delta(0)) = 2$ and by fixing x_e to zero whenever $\pi_e = 0$.

It can be shown that the optimal solutions to the LP relaxations of the directed and undirected formulations have the same cost. The directed formulation does, however, have one advantage over the undirected one: if during the branch-and-bound process we impose the constraint $y_{ij} = 1$, the constraints (4) immediately ensure that $y(\delta^-(S)) \geq 1$ for all S with $i \in S$, $j \notin S$. In the undirected case, on the other hand, imposing $y_e = 1$ has no analogous effect. The result is that the branch-and-bound tree tends to have more nodes when the undirected formulation is used, rather than fewer as one might expect.

The above considerations led us to devise a third formulation, which we call *mixed*, in which the y variables are directed but the x variables are undirected. The derivation of the mixed formulation is straightforward, and we omit details for brevity. Nevertheless, our computational experience (Section 5) indicates that the mixed formulation performs slightly better than the directed and undirected formulations.

4.3 Heuristic pricing

For large instances, solving the pricing sub-problem exactly by mixed-integer programming can be very time-consuming. So, we were led to consider fast heuristics for pricing with elementary routes. We experimented with various constructive heuristics, but did not find any method that consistently led to better overall performance. Instead, we found it more fruitful to adjust the parameters of the MIP solver itself, as follows. In a first

step, we solve the MIP *approximately* by branch-and-bound, aborting whenever the gap between upper and lower bound drops below 0.01. (We do this in ILOG CPLEX by setting the parameters `CPX_PARAM_EPAGAP` and `CPX_PARAM_MIPEMPHASIS` to 0.01 and 1, respectively. Apart from that, all other settings were left at their default values.) If a column is found that has a reduced cost less than 10^{-3} , we add it to the restricted master LP. Otherwise, we proceed to solve the MIP to proven optimality with a much higher degree of precision. (We do this in CPLEX by setting `CPX_PARAM_EPAGAP` to 10^{-6} and `CPX_PARAM_MIPEMPHASIS` to 0. Again, no other parameters were changed.) If a column is found whose reduced cost is less than 10^{-6} , we add it to the master.

5 Computational Experiments

Four sets of CARP instances are commonly used in the literature. The set created by Golden *et al.* [14] contains 23 instances, defined on artificial graphs. The set of Benavent *et al.* [5] contains 34 instances, based on the road network in Valencia. The third set, due to Kiuchi *et al.* [19], contains 6 instances, defined on artificial graphs. In each of these three sets, the demands are randomly generated and all edges are required. The fourth set, due to Li [22] and Li & Eglese [23], contains 24 instances based on the road network in Cumbria. The complete data for these instances are available on the web (<http://www.uv.es/~belengue/carp.html>).

Tables 1 to 4 present some detailed results for these instances. The first four columns give the instance name, the number of vertices and edges, and the minimum number of vehicles needed. The columns marked LB_{NE} and LB_E give the lower bounds obtained with our column generation algorithm, with non-elementary and elementary routes, respectively. The column marked LB_A gives, for comparison, the lower bound obtained by Ahr's cutting plane algorithm [1], based on the exact separation of capacity and R -odd cut inequalities. (Ahr's algorithm ran into time and memory problems for the 12 larger `egl` instances. For those instances, the number reported within parentheses is the best lower bound obtained by his algorithm at the point where the run was aborted.) The last column displays the optimal solution value (in the case of the `gdb`, `val` and `kshs` instances), or the best known upper bound at the time of writing (in the case of the `egl` instances). The optimal solution values are taken from Belenguer & Benavent [4], Longo *et al.* [24], Baldacci & Maniezzo [2] and Ghiani *et al.* [13]. The upper bounds are taken from Brandão & Eglese [7]. Bounds that have been proven optimal are in bold font.

We point out that not all of the `gdb` and `kshs` instances are sparse. Indeed, the instances `gdb16` and `kshs3` are defined on complete graphs.

Two main conclusions can be drawn from these tables. First, forbidding

non-elementary routes leads to significantly improved bounds, especially in the case of the `egl` instances. Second, the bounds obtained with column generation alone are weaker than those obtained with cutting planes alone. However, LB_E is stronger than LB_A in a few cases (specifically, in two `val` instances, one `kshs` instance and eight `egl` instances).

To explore further the quality of the various bounds, we constructed Table 5. This table shows, for the three sets of instances that have been solved to optimality and for several lower bounds, the average ratio between the bound and the optimum. The meaning of the headings LB_{NE} , LB_E and LB_A is as before; LB_{BB} corresponds to the bound obtained by Belenguer & Benavent’s cutting plane algorithm [4], when heuristics are used for separation of capacity and R -odd cut inequalities; and LB_G to the bound obtained with the cut-and-price algorithm of Gómez-Cabrero *et al.* [16], which used non-elementary routes in the pricing routine, and heuristic separation of capacity and R -odd cut inequalities. Based on these figures, we conjecture that a cut-and-price algorithm using exact pricing with elementary routes, or exact separation of capacity and R -odd cut inequalities, or both, would give extremely strong lower bounds.

Finally, Table 6 reports, for each set of instances and for six different pricing strategies, the average time (in seconds) taken to solve the master LP to optimality. As before, the columns headed NE correspond to the case in which non-elementary routes are permitted, whereas the columns headed E correspond to the case in which only elementary routes are permitted. For the column NE-C, we used the traditional pricing routine, based on a transformation to a *complete* graph. For the column NE-S, we used our *sparse* pricing routine, and for the column NE-SH we also used our pricing *heuristics*. The columns E-D, E-U and E-M correspond to the *directed*, *undirected* and *mixed* MIP formulations, respectively.

The cpu times corresponding to the case in which non-elementary routes are permitted show clearly that the sparse approach performs better than the complete approach for all sets of instances, except the `egl` instances. A closer look at the results for the `egl` instances revealed that the complete approach works better than the sparse approach when there is a significant number of non-required edges. Therefore, the sparse approach appears to be more suited for instances where most or all of the edges are required.

As for the case where only elementary routes are generated, the undirected MIP formulation is the slowest of the three formulations. Although it has fewer variables and constraints than the other MIP formulations, and the LP relaxation is typically solved more quickly, the number of branch-and-bound nodes tends to be higher. The directed MIP formulation performs well for small instances, but rather poorly for larger ones, probably due to the relatively large number of variables and constraints. Finally, the mixed MIP formulation is solved considerably more quickly than the others.

Even using the mixed MIP formulation, running times are excessive for

the `val` and `egl` instances. In an attempt to understand this, we produced Figures 1 to 3. The first figure shows how the time taken to solve the pricing subproblem typically evolves: the subproblems faced in the later iterations tend to be the hardest ones. This may be because the later routes tend to accommodate more customers, which makes the ‘knapsack’ aspect of the pricing subproblem harder. The second and third figures show that the column generation process also exhibits a strong ‘tailing-off’ phenomenon, i.e., small changes in the objective function value and in the reduced costs in the later iterations. This behaviour is typical of column generation algorithms.

6 Concluding Remarks

In this paper, we have argued that exact algorithms for the CARP should be capable of exploiting the sparsity of typical real-life CARP instances. We have also argued that algorithms based on the set covering formulation are ideal for doing this. To back up this claim, we have shown how to exploit sparsity in a core component of such algorithms, namely the pricing subproblem.

There are several potential topics for future research. First, one could experiment with *k-cycle elimination* in the exact routine for non-elementary routes, as mentioned in Subsection 3.1. Second, in the case of elementary routes, one might consider solving the pricing MIPs themselves by branch-and-cut, instead of just feeding them to a standard MIP solver. Third, one could attempt to improve the convergence of the column generation process by using *stabilization* techniques (see, e.g., du Merle *et al.* [26]). Fourth, one could develop a full *branch-cut-and-price* algorithm for sparse CARP instances, using one or more of our pricing algorithms as subroutines. Some thought would be needed to devise branching rules that effectively exploited graph sparsity.

Finally, we remark that our algorithm for pricing with non-elementary routes can be adapted with little modification to other VRPs or ARPs defined on sparse graphs. For example, one can easily obtain an $\mathcal{O}(QT|E|)$ pricing algorithm for the *Capacitated Arc Routing Problem with Time Windows* (CARPTW), where T is the number of time periods. However, the approach we have given for pricing with elementary routes is not so easily adapted. For example, we do not know how to formulate the pricing problem for the CARPTW as a mixed-integer program with only $\mathcal{O}(|E|)$ variables.

References

- [1] D. Ahr (2004) Contributions to Multiple Postmen Problems. *PhD dissertation*, Department of Computer Science, Heidelberg University.

- [2] R. Baldacci & V. Maniezzo (2006) Exact methods based on node-routing formulations for undirected arc-routing problems. *Networks*, 47, 52–60.
- [3] J.M. Belenguer & E. Benavent (1998) The capacitated arc routing problem: valid inequalities and facets. *Comput. Optim. Appl.*, 10, 165–187.
- [4] J.M. Belenguer & E. Benavent (2003) A cutting plane algorithm for the capacitated arc routing problem. *Comput. Oper. Res.*, 30, 705–728.
- [5] E. Benavent, V. Campos, A. Corberán & E. Mota (1992) The capacitated arc routing problem: lower bounds. *Networks*, 22, 669–690.
- [6] J. Bramel & D. Simchi-Levi (2002) Set-covering-based algorithms for the capacitated VRP. In P. Toth & D. Vigo (eds.) *The Vehicle Routing Problem*. SIAM Publications.
- [7] J. Brandão & R.W. Eglese (2008) A deterministic tabu search algorithm for the capacitated arc routing problem. *Comput. Oper. Res.*, 35, 1112–1126.
- [8] M. Dror (ed.) (2000) *Arc Routing: Theory, Solutions and Applications*. Kluwer Academic Publishers.
- [9] M. Dror & A. Langevin (2000) Transformations and exact solutions for arc and node routing by column generation. In M. Dror (ed.), *op. cit.*
- [10] B. Fleischmann (1982) *Linear programming approaches to traveling salesman and vehicle scheduling problems*. Presented at the XIth International Symposium on Mathematical Programming, Bonn.
- [11] M.L. Fredman & R.E. Tarjan (1987) Fibonacci heaps and their uses in improved network optimization algorithms. *J. of the ACM*, 34, 596–615.
- [12] R. Fukasawa, H. Longo, J. Lysgaard, M.P. de Aragao, M. Reis, E. Uchoa & R.F. Werneck (2006) Robust branch-and-cut-and-price for the capacitated vehicle routing problem. *Math. Program.*, 106, 491–511.
- [13] G. Ghiani, D. Laganá, G. Laporte & R. Musmanno (2007) A branch-and-cut algorithm for the capacitated arc routing problem. *Working paper*.
- [14] B.L. Golden, J.S. De Armon & E.K. Baker (1983) Computational experiments with algorithms for a class of routing problems. *Comput. Oper. Res.*, 10, 47–59.
- [15] B.L. Golden & R.T. Wong (1981) Capacitated arc routing problems. *Networks*, 11, 305–315.

- [16] D. Gómez-Cabrero, J.M. Belenguer & E. Benavent (2005) Cutting plane and column generation for the capacitated arc routing problem. Presented at ORP3, Valencia.
- [17] L. Gouveia (1995) A result on projection for the vehicle routing problem. *Eur. J. Opl Res.*, 85, 610-624.
- [18] S. Irnich & D. Villeneuve (2006) The shortest-path problem with resource constraints and k -cycle elimination for $k \geq 3$. *INFORMS J. Comput.*, 18, 391-406.
- [19] M. Kiuchi, Y. Shinano, R. Hirabayashi & Y. Saruwatari (1995) An exact algorithm for the capacitated arc routing problem using parallel branch and bound method. Presented at the *1995 Spring National Conference of the Operational Research Society of Japan*.
- [20] M. Labbé, G. Laporte & H. Mercure (1991) Capacitated vehicle routing on trees. *Oper. Res.*, 39, 616-622.
- [21] A.N. Letchford (1997) Polyhedral Results for Some Constrained Arc Routing Problems. *PhD Dissertation*, Department of Management Science, Lancaster University.
- [22] L.Y.O. Li (1992) Vehicle Routeing for Winter Gritting. *Ph.D Dissertation*, Department of Management Science, Lancaster University.
- [23] L.Y.O. Li & R.W. Eglese (1996) An interactive algorithm for vehicle routeing for winter-gritting. *J. Oper. Res. Soc.*, 47, 217-228.
- [24] H. Longo, M. Poggi de Aragão & E. Uchoa (2006) Solving capacitated arc routing problems using a transformation to the CVRP. *Comput. Oper. Res.*, 33, 1823-1837.
- [25] J. Lysgaard, A.N. Letchford & R.W. Eglese (2004) A new branch-and-cut algorithm for the capacitated vehicle routing problem. *Math. Program.*, 100, 423-445.
- [26] O. du Merle, D. Villeneuve, J. Desrosiers & P. Hansen (1999) Stabilized column generation. *Discr. Math.*, 194, 229-237.
- [27] W.-L. Pearn, A. Assad & B.L. Golden (1987) Transforming arc routing into node routing problems. *Comput. Oper. Res.*, 14, 285-288.
- [28] S.A. Welz (1994) Optimal Solutions for the Capacitated Arc Routing Problem Using Integer Programming. *PhD dissertation*, Department of QT and OM, University of Cincinnati.
- [29] S. Wøhlk (2005) Contributions to Arc Routing. *PhD dissertation*, Faculty of Social Sciences, University of Southern Denmark.

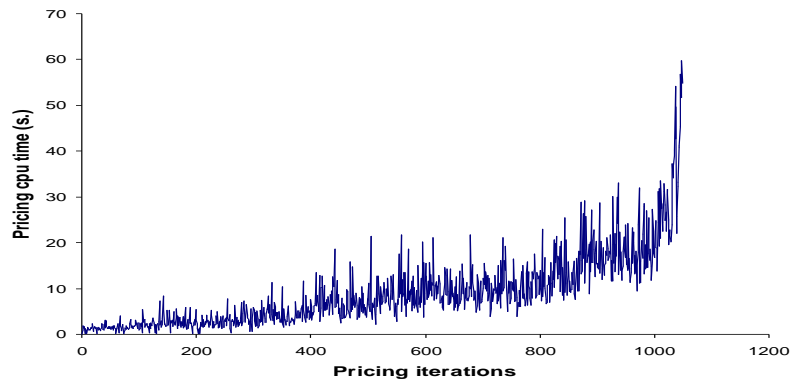


Figure 1: Pricing cpu time when only the elementary routes are generated

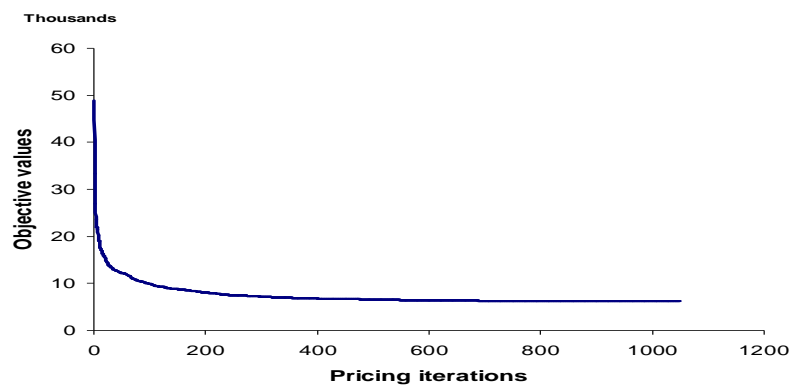


Figure 2: Tail-Off effect curve when only the elementary routes are generated

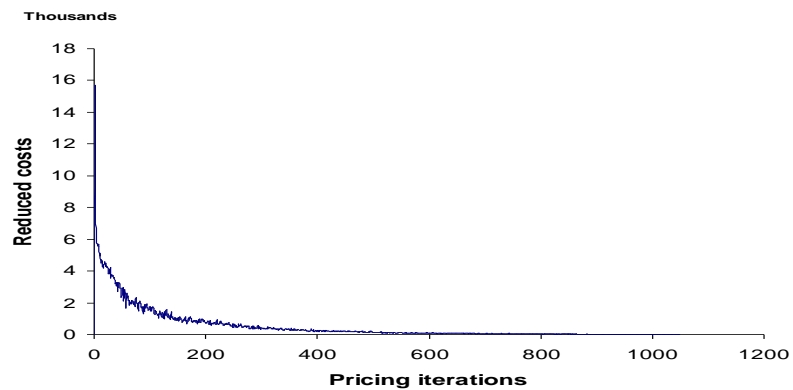


Figure 3: Reduced costs when only the elementary routes are generated

Inst.	$ V $	$ E $	m	LB_{NE}	LB_E	LB_A	Opt
gdb1	12	22	5	282	285	316	316
gdb2	12	26	6	313	314	339	339
gdb3	12	22	5	248	250	275	275
gdb4	11	19	4	266	272	287	287
gdb5	13	26	6	358	359	377	377
gdb6	12	22	5	282	284	298	298
gdb7	12	22	5	288	293	325	325
gdb8	27	46	10	319	330	344	348
gdb9	27	51	10	291	294	303	303
gdb10	12	25	4	254	254	275	275
gdb11	22	45	5	364	364	395	395
gdb12	13	23	7	422	444	450	458
gdb13	10	28	6	525	525	536	536
gdb14	7	21	5	98	98	100	100
gdb15	7	21	4	56	56	58	58
gdb16	8	28	5	122	122	127	127
gdb17	8	28	5	85	85	91	91
gdb18	9	36	5	159	159	164	164
gdb19	8	11	3	47	55	55	55
gdb20	11	22	4	107	114	121	121
gdb21	11	33	6	151	151	156	156
gdb22	11	44	8	196	196	200	200
gdb23	11	55	10	233	233	233	233

Table 1: Results for Golden *et al.* instances

Inst.	$ V $	$ E $	m	LB_{NE}	LB_E	LB_A	Opt
kshs1	8	15	4	13363	13553	14661	14661
kshs2	10	15	4	8195	8723	9863	9863
kshs3	6	15	4	8401	8654	9320	9320
kshs4	8	15	4	11442	11498	11098	11498
kshs5	8	15	3	10215	10370	10957	10957
kshs6	9	15	3	9080	9232	10197	10197

Table 2: Results for Kiuchi *et al.* instances

Inst.	$ V $	$ E $	m	LB_{NE}	LB_E	LB_A	Opt
val1a	24	39	2	220	220	247	247
val1b	24	39	3	223	225	247	247
val1c	24	39	8	294	313	309	319
val2a	24	34	2	270	277	298	298
val2b	24	34	3	300	304	328	330
val2c	24	34	8	515	528	526	528
val3a	24	35	2	92	93	105	105
val3b	24	35	3	99	101	111	111
val3c	24	35	7	153	155	159	162
val4a	41	69	3	478	478	522	522
val4b	41	69	4	490	492	534	534
val4c	41	69	5	511	515	550	550
val4d	41	69	9	615	621	642	652
val5a	34	65	3	524	524	566	566
val5b	34	65	4	545	548	586	589
val5c	34	65	5	574	578	610	617
val5d	34	65	9	682	689	714	720
val6a	31	50	3	300	305	330	330
val6b	31	50	4	309	315	336	340
val6c	31	50	10	397	405	414	424
val7a	40	66	3	352	358	382	382
val7b	40	66	4	354	361	386	386
val7c	40	66	9	401	407	430	437
val8a	30	63	3	489	489	522	522
val8b	30	63	4	501	502	531	531
val8c	30	63	9	633	638	645	657
val9a	50	92	3	407	407	450	450
val9b	50	92	4	412	412	453	453
val9c	50	92	5	419	419	459	459
val9d	50	92	10	481	484	505	518
val10a	50	97	3	590	590	637	637
val10b	50	97	4	597	597	645	645
val10c	50	97	5	608	609	655	655
val10d	50	97	10	691	695	731	735

Table 3: Results for Benavent *et al.* instances

Inst.	$ V $	$ E $	$ E_R $	m	LB_{NE}	LB_E	LB_A	UB
egl-e1-A	77	98	51	5	2983	3425	3516	3548
egl-e1-B	77	98	51	7	3791	4291	4436	4498
egl-e1-C	77	98	51	10	4931	5472	5481	5595
egl-e2-A	77	98	72	7	4221	4832	4963	5018
egl-e2-B	77	98	72	10	5463	6105	6271	6317
egl-e2-C	77	98	72	14	7679	8187	8155	8335
egl-e3-A	77	98	87	8	5076	5706	5866	5898
egl-e3-B	77	98	87	12	6882	7541	7649	7777
egl-e3-C	77	98	87	17	9434	10086	10119	10305
egl-e4-A	77	98	98	9	5634	6233	6378	6456
egl-e4-B	77	98	98	14	8048	8678	8838	9000
egl-e4-C	77	98	98	19	10770	11416	11376	11601
egl-s1-A	140	190	75	7	4170	4985	(4975)	5018
egl-s1-B	140	190	75	10	5542	6284	(6180)	6388
egl-s1-C	140	190	75	14	7716	8423	(8286)	8518
egl-s2-A	140	190	147	14	8867	9667	(9718)	9956
egl-s2-B	140	190	147	20	12146	12801	(12835)	13165
egl-s2-C	140	190	147	27	15618	16262	(16216)	16524
egl-s3-A	140	190	159	15	9190	9925	(9991)	10260
egl-s3-B	140	190	159	22	12752	13388	(13520)	13807
egl-s3-C	140	190	159	29	16390	17014	(16958)	17234
egl-s4-A	140	190	190	19	11314	11905	(12007)	12341
egl-s4-B	140	190	190	27	15266	15891	(15897)	16462
egl-s4-C	140	190	190	35	19651	20197	(20176)	20591

Table 4: Results for Li and Eglese instances

Set	LB_{NE}	LB_E	LB_{BB}	LB_A	LB_G
gdb	.9368	.9524	.9987	.9987	.9990
val	.9230	.9327	.9940	.9941	.9964
kshs	.9103	.9315	.9942	.9942	—

Table 5: Average ratios computed for thee test sets and five lower bounds

Set	NE-C	NE-S	NE-SH	E-D	E-U	E-M
gdb	2.26	0.45	0.17	28.61	54.23	20.10
val	64.33	31.99	10.81	4364.19	6696.91	3531.59
ksh	0.85	0.22	0.09	3.75	3.86	3.75
egl	410.20	500.14	385.40	36890.00	57932.94	30542.09

Table 6: Time to solve the master LP under six pricing strategies