Math. Program., Ser. A 100: 423–445 (2004)

Digital Object Identifier (DOI) 10.1007/s10107-003-0481-8

Jens Lysgaard · Adam N. Letchford · Richard W. Eglese

A new branch-and-cut algorithm for the capacitated vehicle routing problem

Received: March 18, 2003 / Accepted: September 24, 2003
Published online: November 21, 2003 – © Springer-Verlag 2003

Abstract. We present a new branch-and-cut algorithm for the *capacitated vehicle routing problem* (CVRP). The algorithm uses a variety of cutting planes, including *capacity*, *framed capacity*, *generalized capacity*, *strengthened comb*, *multistar*, *partial multistar*, *extended hypotour* inequalities, and classical *Gomory mixed-integer cuts*.

For each of these classes of inequalities we describe our separation algorithms in detail. Also we describe the other important ingredients of our branch-and-cut algorithm, such as the branching rules, the node selection strategy, and the cut pool management. Computational results, for a large number of instances, show that the new algorithm is competitive. In particular, we solve three instances (`B-n50-k8`, `B-n66-k9` and `B-n78-k10`) of Augerat to optimality for the first time.

Key words. vehicle routing – branch-and-cut – separation

1. Introduction

The *Capacitated Vehicle Routing Problem* (CVRP), which can be traced back to [18], is defined as follows. A complete undirected graph $G = (V, E)$ is given, with $V = \{0, \ldots, n\}$. Vertex 0 represents a depot, the other vertices represent customers. The cost of travel between vertices i and j is denoted by c_{ij}, and we assume that costs are symmetric, i.e., that $c_{ij} = c_{ji}$. A fleet of identical vehicles, each of capacity $Q > 0$, is available. (A limit on the number of vehicles, or a fixed charge associated with hiring a vehicle, may or may not be given.) Each customer i has an integer demand q_i, with $0 < q_i \leq Q$. Each customer must be served by a single vehicle and no vehicle can serve a set of customers whose demand exceeds its capacity. The task is to find a set of vehicle routes of minimum cost, where each vehicle used leaves from and returns to the depot.

The CVRP is strongly $\mathcal{NP}$-hard, since it includes the *Traveling Salesman Problem* as a special case. Nevertheless, small to medium-sized instances can be solved to proven optimality. A variety of optimization approaches have been applied to the CVRP, see [10, 27, 28, 42] for detailed surveys. Some of these are based on the use of graph theoretical relaxations such as b-matchings [32] or K-trees [20]; others are based on dynamic programming [26]; still others are based on set partitioning and column generation [2].

J. Lysgaard: Department of Management Science and Logistics, Aarhus School of Business, Denmark
e-mail: `lys@asb.dk`

A.N. Letchford: Department of Management Science, Lancaster University, Lancaster LA1 4YW, England
e-mail: `A.N.Letchford@lancaster.ac.uk`

R.W. Eglese: Department of Management Science, Lancaster University, Lancaster LA1 4YW, England
e-mail: `R.Eglese@lancaster.ac.uk`

However, at present, the most promising solution technique appears to be *branch-and-cut*, in which valid linear inequalities are used as cutting planes to strengthen a linear programming relaxation at each node of a branch-and-bound tree [5, 7, 12, 33, 37, 40]. These branch-and-cut algorithms are all based on the following so-called *two-index formulation* of the CVRP.

Let x_{ij} represent the number of times a vehicle travels between vertices i and j. (Because the problem is undirected, x_{ij} and x_{ji} represent the same variable.) Let $V_c = V \setminus \{0\}$ denote the set of customers. Given a set of customers $S \subseteq V_c$, let $q(S)$ denote $\sum_{i \in S} q_i$, $\delta(S)$ denote the set of edges in G with exactly one end-vertex in S, $E(S)$ denote the set of edges in G with both end-vertices in S, and $r(S)$ denote the minimum number of vehicles required to serve the customers in S. That is, $r(S)$ is the optimal solution to the *Bin Packing Problem* (BPP) with bin capacity Q and item sizes given by the demands of the customers in S. Finally, given an arbitrary $F \subseteq E$, $x(F)$ will denote $\sum_{e \in F} x_e$. The integer programming formulation is then:

$$
\begin{aligned}
&\textit{Minimize} && \textstyle\sum_{e \in E} c_e x_e && && \\
&\textit{Subject to:} && x(\delta(\{i\})) = 2 && (i = 1, \ldots, n) && (1)\\
& && x(\delta(S)) \geq 2r(S) && (S \subseteq V_c, |S| \geq 2) && (2)\\
& && x_{ij} \in \{0, 1\} && (1 \leq i < j \leq n) && (3)\\
& && x_{ij} \in \{0, 1, 2\} && (i = 0, j = 1, \ldots, n). && (4)
\end{aligned}
$$

The *degree equations* (1) ensure that customers are visited exactly once. The *capacity inequalities* (2) impose the vehicle capacity restrictions and also ensure that the routes are connected. Calculating $r(S)$ exactly is as hard as the BPP and is therefore strongly $\mathcal{NP}$-hard. However, the formulation remains valid if one replaces $r(S)$ on the right-hand side with the obvious lower bound $k(S) = \lceil q(S)/Q \rceil$, which yields the so-called *rounded capacity inequalities* [33]. Finally, constraints (3) and (4) are the integrality conditions. Note that the x_{ij} are permitted to take the value 2 when $i = 0$, to allow routes in which a vehicle serves a single customer.

Some other variants of the CVRP can be easily incorporated into this framework. If there is an upper bound K on the number of vehicles to be used, then the inequality $x(\delta(\{0\})) \leq 2K$ can be added. This can be made into an equation if *exactly* K vehicles must be used. Alternatively, if hiring a vehicle incurs a fixed cost C, then $C/2$ can be added to the objective function coefficient of each edge incident on the depot. Finally, if routes containing only one customer are forbidden, then all variables can be made binary. Branch-and-cut algorithms can be used in all of these cases.

The valid inequalities used in branch-and-cut algorithms are drawn from studies of the polytope associated with the above formulation, viz., the convex hull in $\mathbb{R}^{|E|}$ of incidence vectors x satisfying (1) – (4) (see [34]). Valid inequalities for this polytope (or closely related polyhedra) appear in [1, 7, 17, 22, 29, 30]. Also, in [3–5, 13], special attention is given to the *unit demand* case, i.e., the special case where $q_i = 1$ for $i = 1, \ldots, n$.

The purpose of the present paper is to describe a new branch-and-cut algorithm for the CVRP. In Section 2 we present exact and heuristic algorithms for the separation of various classes of inequalities. In Section 3, we describe some of the other components

Table 1. The CVRP instances.

Name	Q	loading	UB
E-n22-k4	6000	.94	375*
E-n23-k3	4500	.75	569*
E-n30-k3	4500	.94	534*
E-n33-k4	8000	.92	835*
E-n51-k5	160	.97	521*
E-n76-k7	220	.89	682*
E-n76-k8	180	.95	735*
E-n76-k10	140	.97	830
E-n76-k14	100	.97	1021
E-n101-k8	200	.91	815*
E-n101-k14	112	.93	1071
M-n101-k10	200	.91	820*
F-n45-k4	2010	.90	724*
F-n72-k4	30000	.96	237*
F-n135-k7	2210	.95	1162*

of our branch-and-cut algorithm, including branching and node selection rules, and management of the cut pool. Computational results are given in Section 4 and concluding comments are made in Section 5.

To avoid the risk of over-tuning of parameters in our branch-and-cut algorithm, we used a small set of CVRP instances when coding and tuning the algorithm and a much larger set when running the final algorithm. The smaller set of instances is shown in Table 1. They are all Euclidean, with edge costs rounded to integers according to the TSPLIB standard [41]. The name of the instance indicates the source of the instance, the number of vertices, and the number of vehicles, which for all of these instances is fixed at the minimum possible. For example, `E-n22-k4` is taken from [15], and has 22 vertices and 4 vehicles. In Table 1 we also show the vehicle capacity Q, the 'loading' (which is the total demand divided by KQ) and the best known upper bound at the time of writing, taken from [12, 33, 38, 40, 43]. (The upper bound of 1071 for `E-n101-k14` was taken from www.tem.nctu.edu.tw/network and checked for feasibility.) The upper bound is marked with a '*' if it has been shown to be optimal.

2. Separation algorithms

The initial linear programming (LP) relaxation, consisting of the degree equations (1) and the bounds on the variables implied by (3), (4), is easily solved. This generally gives an extremely weak lower bound, which can then be strengthened by adding cutting planes to the LP. To generate these cutting planes, *separation algorithms* are needed, see [23, 34]. An *exact* separation algorithm for a given class of inequalities is a routine which takes as input an LP solution vector x^* and outputs one or more violated inequalities in that class (if any exist). A *heuristic* separation algorithm is similar, except that it may fail to detect violated inequalities in the class.

In the following subsections we describe our exact and heuristic separation algorithms for various classes of valid inequalities for the CVRP.

We will use the following notation throughout. For any LP solution vector x^*, we denote the corresponding support graph by $G^* = (V, E^*)$, where $E^* = \{e \in E :$

$x_e^* > 0\}$. Similarly, we let $G_c^* = (V_c, E_c^*)$ denote the graph obtained by removing the depot from G^*. For two disjoint vertex sets S and T, we denote by $E(S : T)$ the set of edges with one end-vertex in S and the other in T. Finally, by *checking* an inequality we mean that we generate it if it is violated.

2.1. Capacity inequalities

The separation of the capacity inequalities (2) is known to be strongly $\mathcal{NP}$-hard, see [7, 33]. Therefore we resort to heuristics. For the sake of computational tractability, we replace $r(S)$ by $k(S)$ in (2), i.e., we separate rounded capacity inequalities (RCIs). Altogether we use four heuristics for separating RCIs.

The first heuristic is a simple *connected components* heuristic that works as follows. First we compute the connected components $S_1, \ldots, S_p$ of G_c^*. Then, for every $i = 1, \ldots, p$ we check the RCI for S_i as well as for $V_c \setminus S_i$. Finally we check the RCI for the union of those components which are not connected to the depot in G^*. We note that this heuristic with certainty finds a violated RCI (if one exists) if x^* is integer. Similar heuristics based on finding connected components are used in other branch-and-cut algorithms for the CVRP [7, 40].

If this heuristic fails, we proceed by using our last three heuristics, all of which take a *shrunk* support graph as input. The idea of shrinking is well-known [36]. In our context we iteratively choose a customer set S and shrink it to a single *supervertex* s having demand $q(S)$; an edge $\{s, j\}$ in the shrunk graph is given the weight $x^*(E(S : \{j\}))$. Under certain conditions the shrinking is *safe*, which means that whenever there is a violated capacity inequality in G^*, there exists a set of supervertices in the shrunk graph whose union defines a capacity inequality with at least the same violation. It is well-known that it is safe to shrink edges e satisfying $x_e^* \geq 1$ ([7, 8, 40]). However, we have generalized the condition for safe shrinking to cases where $|S| > 2$, as described in proposition 1.

Proposition 1. *For separation of capacity inequalities, it is safe to shrink a customer set S if $x^*(\delta(S)) \leq 2$ and $x^*(\delta(R)) \geq 2 \forall R \subset S$.*

Proof. Let T be a customer set which *crosses* S, i.e., such that $T \cap S$, $T \setminus S$ and $S \setminus T$ are all non-empty. We show that the capacity inequality on $S \cup T$ is violated by at least as much as the capacity inequality on T. From the definition of the capacity inequalities this holds if and only if $x^*(\delta(T)) - x^*(\delta(S \cup T)) + 2r(S \cup T) - 2r(T) \geq 0$. It is trivially true that $2r(S \cup T) - 2r(T) \geq 0$, so it suffices to show that $x^*(\delta(T)) - x^*(\delta(S \cup T)) \geq 0$. It follows from the submodularity of the cut function ([34], p. 660) that $x^*(\delta(T)) - x^*(\delta(S \cup T)) \geq x^*(\delta(S \cap T)) - x^*(\delta(S))$. Since $x^*(\delta(S \cap T)) \geq 2$ and $x^*(\delta(S)) \leq 2$, we have that $x^*(\delta(T)) - x^*(\delta(S \cup T)) \geq 0$. □

We note that the condition for safe shrinking in proposition 1 also applies to RCIs.

In our code we consider sets of the following types as candidates for shrinking (which is done iteratively as long as possible):

- Sets of cardinality 2 or 3;
- Sets S with $k(S) = 1$ for which we have generated a capacity inequality;

- Sets S for which $x(\delta(S)) = 2$ has been imposed during branching (see Subsection 3.2).

We let $GS = (VS, ES)$ denote the shrunk support graph obtained by this shrinking procedure, and similarly we let $GS_c = (VS_c, ES_c)$ be the graph obtained by removing the depot from GS. We will refer to each member of VS_c as a *supervertex*, even if it represents only one customer.

With our second heuristic, we separate the so-called *fractional capacity inequalities*, which are obtained by replacing the right hand side of (2) with the lower bound $2q(S)/Q$. As noted by several authors (see [33]), these weaker inequalities can be separated in polynomial time by solving a max-flow problem in a suitably-defined weighted graph. If a violated fractional capacity inequality is found, then the stronger RCI is of course violated. We have modified the basic version of this separation routine so that we are able to find not only one cut, but several different cuts identified by solving a max-flow problem. The key idea is that, by changing the weights of certain arcs in the above-mentioned max-flow problem to infinity, we can easily force any vertex to lie either inside S or outside it. More precisely, let T_{in} and T_{out} be disjoint subsets of VS_c. Then by solving a max-flow problem we can find, among all sets $S \subset VS_c$ such that $T_{in} \subseteq S$ and $T_{out} \cap S = \emptyset$, the one which minimizes the slack of the fractional capacity inequality. By running the max-flow algorithm several times, each time with different sets T_{in} and T_{out}, we can generate several different sets S.

The way that we choose T_{in} and T_{out} can be briefly outlined as follows. For $s = 1, \ldots, |VS_c|$ we define a max-flow problem by letting $T_{in} = \{s\}$ and by heuristically determining T_{out} such that all previously generated cuts are effectively forbidden (we call s the *seed* of the resulting cut). The procedure for determining T_{out} simply goes back through the previously generated cuts, adding the seed of a cut to T_{out} if that cut has not yet been prohibited. We have found that it is worthwhile to run more than one *round* of $|VS_c|$ max-flow problems. Specifically, we run three rounds in total, where we in subsequent rounds also prohibit cuts from earlier rounds.

Our third heuristic is a greedy construction heuristic, where we again consider each supervertex of GS_c as a seed. More formally, for $s = 1, \ldots, |VS_c|$ we set $S = \{s\}$ and then iteratively expand S by one supervertex and check the RCI for the resulting set. The supervertex j that we add to S is the one that minimizes the slack of the RCI for $S \cup \{j\}$ subject to the restriction that we have not generated the set $S \cup \{j\}$ before. When we cannot expand the current set without generating a previously generated set, we proceed to the next seed.

Our fourth and last heuristic runs through all the RCIs that we have generated until the latest LP reoptimization. For every such inequality, defined on customer set S, we first replace S by the smallest set of supervertices in GS_c which contains S. (In view of proposition 1, this cannot lead to an increase in the slack of the RCI on S.) We then consider the supervertices of this set in non-decreasing order of demand, and remove from the set any supervertex whose removal decreases the slack of the RCI. From there we iteratively add a supervertex to, or drop a supervertex from, or replace one supervertex by another in the current set as long as the slack of the RCI decreases. Here we choose that supervertex or pair of supervertices which decreases the slack of the RCI by the largest amount. When this is no longer possible we check the RCI and proceed to the next among the previously generated RCIs.

Table 2. Lower bounds using rounded capacity inequalities.

Name	Aug.	Our	Time
E-n22-k4	375	375*	2
E-n23-k3	569	569*	2
E-n30-k3	508.5	508.5	2
E-n33-k4	833.5	833.5	4
E-n51-k5	514.524	514.524	4
E-n76-k7	661.299	661.358	7
E-n76-k8	711.17	711.201	10
E-n76-k10	789.416	789.441	19
E-n76-k14	—	947.983	27
E-n101-k8	796.314	796.414	12
E-n101-k14	—	1008.146	65
M-n101-k10	819.5	819.5	30
F-n45-k4	724	724	4
F-n72-k4	232.5	232.5	3
F-n135-k7	1158.25	1158.25	59

We invoke these four heuristics as follows. If the first heuristic generates one or more inequalities, we do not invoke any of the last three heuristics. If the first heuristic fails, we call the last three heuristics in the order mentioned. We restrict the second heuristic to generate no more than $\min\{n/2, 50\}$ inequalities; moreover, we generate no more than $\min\{n, 100\}$ inequalities in total.

Table 2 displays our results obtained by RCIs. The first column labelled 'Aug.' gives the best bound obtained using capacity separation heuristics only given in Augerat et al. [7, 8]. (The figure is missing for two instances because they do not appear in those papers.) Then we have 'Our', which is the bound obtained by our capacity separation heuristics.

The last column shows our computing time in seconds. Throughout we will report the computing time rounded to the nearest whole number of seconds. The hardware used is a PC with a 700 MHz Intel Celeron processor and 256 MB of RAM running under Microsoft Windows 98. We used the ILOG CPLEX 6.0 callable library and the Watcom C/C++ compiler v. 11.0.

Our bounds are equal to or slightly better than those obtained in [7, 8]. Moreover, by comparing with the bounds obtained by exact enumeration in [11] it can be concluded that our bounds equal or are very close to the best possible bounds that can be obtained by RCIs.

As shown in Table 2, two of the instances, namely `E-n22-k4` and `E-n23-k3` can be solved using only RCIs.

2.2. *Framed capacity inequalities*

In [6], the following inequalities were presented, which have come to be known as *framed capacity inequalities* [33]. For some $S \subseteq V_c$, let $\Omega = \{S_1, \ldots, S_p\}$ be a partition of S. Now let $r(S, \Omega)$ equal the minimum number of vehicles needed to service S *given that* the capacity inequality for each S_i holds with equality. A good lower bound on $r(S, \Omega)$ can be found by solving a BPP. Then the framed capacity inequality (FCI) is:

Table 3. Lower bounds using framed capacity inequalities.

Name	Aug.	Our	Time
E-n30-k3	532.5	532.5	7
F-n72-k4	235	236.25	16

$$x(\delta(S)) + \sum_{i=1}^{p} x(\delta(S_i)) \geq 2r(S, \Omega) + 2\sum_{i=1}^{p} r(S_i). \tag{5}$$

Intuitively, this inequality works as follows: if all of the capacity inequalities for the sets S_i are tight, then the summation on the left hand side equals the summation on the right hand side, and therefore $x(\delta(S))$ must be at least $2r(S, \Omega)$, as required. Note that, if any S_i is a singleton, then the i'th terms in the summations can be omitted due to the degree equations. In the special case where $S = V_c$, the inequality is referred to as a *generalized capacity inequality* in [7], which is slightly different from the *weak generalized capacity inequality* defined in [33], where it is assumed that $q(S_i) \leq Q$ holds for every $i = 1, \ldots, p$. Strictly speaking, the same condition applies to the FCI as defined in [33]. However, in our implementation we allow that the demand of a supervertex exceeds Q, as we have found computationally that this is important for the strength of the FCIs. For the sake of computational tractability, we replace $r(S_i)$ by $k(S_i)$ for $i = 1, \ldots, p$ in our implementation.

It is easy to show that, if any FCI is violated, then there exists a most violated FCI such that S equals or is a subset of one of the connected components in G_c^*. Thus each component can be treated separately. For the sake of computational tractability, we consider only whole components as candidates for S in (5).

We have implemented a depth-first tree search procedure for separating FCIs. From node p with partition Ω^p in the search tree we generate s descendant nodes, one for each of the s edges in the shrunk graph at node p. The descendant nodes represent partitions $\Omega_1^p, \ldots, \Omega_s^p$, where the k'th partition is obtained by shrinking the edge of k'th largest weight in Ω^p and by prohibiting shrinking of any of the $k-1$ edges with larger weights (ties are broken arbitrarily) in the subtree rooted at descendant node k. We also perform induced shrinking of any newly formed edges with weights of one or more.

For each partition we solve the corresponding BPP using the procedures in [31] and check the FCI. Backtracking is performed if the FCI is violated, or has slack at least 2, or if a prohibited shrinking is induced. We place a limit on the number of nodes that we consider in the search tree (see Section 3).

The separation heuristics for FCIs in [7, 33] can be viewed as restricted versions of our procedure. In fact, they correspond to examining only one path in our search tree.

With this separation procedure, together with the separation of RCIs, we obtain the results displayed in Table 3. We find that the FCIs give an improvement over the bounds obtained by RCIs on only two instances, which was also the conclusion reached in [7]. However, for these two instances the improvement is substantial. Indeed, the bound obtained for `F-n72-k4` is sufficient to prove that the optimal solution is indeed optimal.

2.3. Strengthened comb inequalities

The *comb* inequalities [24, 25] are a class of inequalities for the TSP which have proven to be highly useful as cutting planes. Several authors have attempted to adapt them to the CVRP [3, 7, 12, 29]. Here we follow the approach of [3].

We define a *comb* to be a vertex set $H \subset V_c$, called the *handle*, and $t \geq 2$ other vertex sets $T_1, \ldots, T_t$, called *teeth*, such that:

- $H \cap T_j \neq \emptyset$ and $T_j \setminus H \neq \emptyset$ for $j = 1, \ldots, t$;
- for each pair $\{i, j\} \subset \{1, \ldots, t\}$: $T_i \cap T_j \subset H$ or $T_i \cap T_j \cap H = \emptyset$.

This is more general than in the case of the TSP, because we allow teeth to intersect and we do not require that the number of teeth be odd.

Now, for any set $S \subset V$, let $\tilde{r}(S)$ equal $r(S)$ if $0 \notin S$, and $r(V \setminus S)$ otherwise. We define the quantity:

$$S(H, T_1, \ldots, T_t) := \sum_{j=1}^{t} \left(\tilde{r}(T_j \cap H) + \tilde{r}(T_j \setminus H) + \tilde{r}(T_j) \right).$$

If $S(H, T_1, \ldots, T_t)$ is odd, then the following *strengthened comb inequality* is valid for the CVRP:

$$x(\delta(H)) + \sum_{j=1}^{t} x(\delta(T_j)) \geq S(H, T_1, \ldots, T_t) + 1. \tag{6}$$

(In fact, in [3] validity of this inequality is shown only for the special case in which teeth do not intersect, but the proof of validity carries over easily to this more general case.)

The strengthened comb inequalities (6) can be shown to generalize and dominate the comb inequalities given in [3, 7, 12, 29], with the exception of some special inequalities mentioned in [3, 12], concerned with so-called *large teeth*. They reduce to ordinary comb inequalities [24, 25] when $S(H, T_1, \ldots, T_t) = 3t$ (the smallest possible value) and teeth are not permitted to intersect. If, in addition, $|T_j| = 2$ for all j, we obtain the *2-matching* inequalities of Edmonds [19].

Our separation heuristic for the strengthened comb inequalities is as follows. For the sake of computational tractability, we replace $r(S)$ by $k(S)$ for any set S in our implementation.

- First, we iteratively shrink sets $S \subset V_c$ such that
  - $|S| \in \{2, 3\}$,
  - $x^*(\delta(S)) = 2$,

  and such that at least one of the following conditions holds:
  - $x^*(E(S : \{i\})) = 1$ for some $i \in V_c \setminus S$,
  - $x^*(E(S : \{0\})) = 1$ and $2k(V_c \setminus S) = 2k(V_c) = x^*(\delta(\{0\}))$.

  Using similar arguments to those in [36], it can be shown that a violated strengthened comb inequality exists in the shrunk graph if and only if one existed in the original graph.

Table 4. Lower bounds using comb inequalities.

Name	Aug.	Our	Time
E-n51-k5	517.111	517.176	10
E-n76-k7	662.661	663.329	20
E-n76-k8	712.415	713.342	21
E-n76-k10	790.792	791.224	34
E-n76-k14	—	948.018	30
E-n101-k8	798.358	799.088	38
E-n101-k14	—	1010.043	88
F-n135-k7	1158.48	1158.825	124

- We identify a number of *candidate handles* using ideas similar to those in [36]. For a given $0 < \epsilon \leq \frac{1}{2}$ we delete all edges from the shrunk graph apart from those with $\epsilon \leq x_e^* \leq 1 - \epsilon$. Each connected component in the resulting graph, and each biconnected component (block), is a candidate handle. In fact, we implicitly consider *all* values of ϵ by introducing edges e into the shrunk graph in non-decreasing order of $|x_e^* - \frac{1}{2}|$ and checking at each stage if a new component or block has been created.
- For each candidate handle, we find (in the shrunk graph) the 2-matching inequality with minimum slack. The teeth of the 2-matching inequality may intersect, but only in the depot. This yields an ordinary comb inequality in the original graph.
- For each comb inequality found (violated or not), we try to modify the teeth in order to find a violated strengthened comb inequality. This is a simple heuristic which enlarges one tooth at a time by iteratively adding a vertex in a greedy way.
- If no violated inequality has yet been found, we run the polynomial-time exact separation algorithm for 2-matching inequalities, due to Padberg & Rao [35], on the shrunk graph. This is not too time-consuming, as the shrunk graph is quite small. This may produce candidate handles which have not yet been considered. For each such handle, we apply the above-mentioned procedure in an attempt to find a violated strengthened comb inequality.

With this separation procedure, together with the separation of RCIs, we obtain the results displayed in Table 4. We find that the strengthened comb inequalities give an improvement over the bounds obtained by RCIs on eight instances. Moreover it is clear that our separation heuristic outperforms that of [7].

2.4. Multistar and partial multistar inequalities

Multistar inequalities were originally defined by Araque, Hall & Magnanti [4] for the VRP with *unit demands*. These inequalities take the form

$$\alpha x(E(N)) + \beta x(E(N : S)) \leq \gamma, \tag{7}$$

where $N \subset V_c$ is the so-called *nucleus*, $S \subseteq V_c \setminus N$ is the set of so-called *satellites*, and α, β and γ are constants which depend on $|N|$ and $|S|$. The same authors also introduced the *partial* multistar inequalities, which take the form:

$$\alpha x(E(N)) + \beta x(E(C : S)) \leq \gamma, \tag{8}$$

Table 5. Lower bounds using multistar inequalities.

Name	LB	Time
E-n33-k4	833.875	6
E-n51-k5	514.556	4
E-n76-k7	663.173	24
E-n76-k8	714.33	43
E-n76-k10	796.501	103
E-n76-k14	966.281	202
E-n101-k8	798.687	53
E-n101-k14	1023.787	363
M-n101-k10	820*	33
F-n135-k7	1159.586	152

where $C \subset N$ is the set of so-called *connector* vertices, and, again, α, β and γ are constants which depend on $|N|$, $|S|$ and $|C|$.

In [30], we showed how to generalize these inequalities to the case of the general CVRP, yielding the so-called *homogeneous* multistar and partial multistar inequalities. We also gave reasonably effective separation heuristics for them. For the sake of space we do not repeat the detailed description of the separation heuristics here. It suffices to say that greedy methods are used to select the nucleus N, and then a greedy heuristic is used to select the satellites S (and the connectors C in the case of partial multistars).

A related, but not identical, set of valid inequalities was proposed by Gouveia [22] for the CVRP with general demands:

$$Qx(E(N)) + \sum_{j \in V_c \setminus N} q_j x(E(N : \{j\})) \leq Q|N| - q(N). \tag{9}$$

These have come to be known as *generalized large multistar* inequalities [1, 30]. It is well-known [12, 22, 30] that, although the GLM inequalities dominate the fractional capacity inequalities (see Subsection 2.1), the separation problem for them can be solved in polynomial time. For related results see [1, 12, 30].

Using our separation procedure for multistar (and partial multistar) inequalities, together with the separation of RCIs, we obtain the results displayed in Table 5. We find that the multistar inequalities give an improvement over the bounds obtained by RCIs on ten instances. Moreover, this improvement is significant for the instances with a large number of vehicles.

2.5. Hypotour inequalities

Let $F \subset E$ be such that any feasible CVRP solution uses at least one edge from F. Then the *hypotour inequality* is:

$$x(F) \geq 1, \tag{10}$$

which was introduced in the context of the CVRP in [6]. Several variations of (10) have been proposed, including the *extended hypotour inequality*, which was also introduced in [6]. Other variations are presented in [7].

We have considered only one of these variations, which we call a *2-edges extended hypotour inequality* (2EH). For a given $W \subset V_c$ and two distinct edges $e_1, e_2 \in \delta(W)$, the 2EH is:

$$x(\delta(W)) + 2x(F) \geq 2x_{e_1} + 2x_{e_2}, \tag{11}$$

where $F \subset E$. Let $e_i = \{u_i, v_i\}$ for $i = 1, 2$ with $v_1, v_2 \notin W$. In the following we call v_1, v_2 the *terminals* of the 2EH.

The 2EH expresses that in any feasible CVRP solution with $x(\delta(W)) = 2$ and $x_{e_1} = x_{e_2} = 1$, at least one edge from F must be used. That is, any feasible route visiting all customers in $W \cup \{v_1, v_2\}$ consecutively, beginning and ending at the given terminals, must use at least one edge from F. The identification of F is essentially based on the fact that this hamiltonian path through $W \cup \{v_1, v_2\}$ must be extended by two paths to the depot, such that these paths are vertex disjoint except their intersection at the depot, and such that the total demand on the created cycle does not exceed the vehicle capacity.

Our separation procedure begins with generating a list of candidate sets each of which can be used as $S = W \cup \{v_1, v_2\}$. For this purpose we use a greedy search in the shrunk graph that we use for capacity separation, where we try to keep the boundary of the set as small as possible. Among all these candidate sets we remove some sets based on a (heuristic) dominance criterion, which says that S_i dominates S_j if $x^*(\delta(S_i)) \leq x^*(\delta(S_j))$ and $S_j \subset S_i$. For each remaining candidate set S, we consider as terminals all those vertex pairs $v_1, v_2 \in S$ such that $2x^*_{e_1} + 2x^*_{e_2} - x^*(\delta(W)) > 0$ (i.e., such that the inequality is violated if $x^*(F) = 0$). For every such combination of W and pair of terminals, we use the procedure described in the following.

Our separation procedure works on the subgraph of G^* induced by the vertices in $V \setminus W$. We denote this graph by D and its edge set by E_D. Each edge $\{i, j\} \in E_D$ is given the weight $d_{ij} = (w_i + w_j)/2$, where $w_i = 0$ for $i \in \{0, v_1, v_2\}$, and $w_i = q_i$ otherwise.

Let an $\{i, j\}$-*path* denote any elementary path from i to j in D. We call two paths *customer disjoint* if they have no customer in common. A key element of the separation is to find two customer disjoint paths, a $\{0, v_1\}$-*path* and a $\{0, v_2\}$-*path*, of minimum total weight in D. If their total weight exceeds $Q - q(S)$, we have a violated 2EH.

The problem of finding such a pair of paths of minimum total weight can be defined as a (linear) Assignment Problem (AP) using the edge weights in D. All non-existing edges in D are given some large weight, except that we set $d_{ii} = 0$ for $i \in V$. Now, by deleting rows v_1, v_2 and column 0 and by adding a copy of row 0 we obtain an AP to which any feasible (integer) solution contains a $\{0, v_1\}$-*path* and a $\{0, v_2\}$-*path*, intersecting only in 0. If $v(AP) > Q - q(S)$, where $v(AP)$ is the objective value of the optimal assignment, the 2EH inequality with $F = E \setminus E_D$ is valid. (We will later describe how we attempt to reduce F.)

For solving the AP, there immediately are $|V \setminus W| - 2$ allocations of zero value available, so only two iterations of the Hungarian algorithm ([14]) remain.

If a violated inequality is found, we proceed to the next combination of S and pair of terminals. Otherwise, we attempt to find a violated 2EH by deleting one or more edges from E_D, as we now describe.

Let L_{ij} be the length of the shortest $\{i, j\}$-*path* for any two distinct vertices i, j. Note that $L_{v_10} + L_{v_20} \leq v(AP)$. If we by deleting a certain set of edges from E_D obtain that $L_{v_10} + L_{v_20} > Q - q(S)$, the inequality with $F = E \setminus E_D$ is valid, and the inequality is violated if $x^*(F)$ is kept at a sufficiently small value.

We consider in turn each of the two terminals. Let v_i be the current terminal and v_j the other terminal, and let $L_{max} = Q - q(S) - L_{v_j0}$. We attempt to delete a set of edges with small x^*-value from E_D such that $L_{v_i0} > L_{max}$. This is done as follows.

Let $P = \{v_i, 0\} \cup \{p \in V_c \setminus S \colon L_{v_ip} + L_{p0} \leq L_{max}\}$. Any $\{v_i, 0\}$-*path* whose length does not exceed L_{max} can visit only vertices in P. Further, let D_P be the subgraph of D induced by P. We compute the blocks in D_P and check if the shortest $\{v_i, 0\}$-*path* visits more than one block. Suppose that the path is $\ldots, v_a, v_b, v_c, \ldots$, such that the two edges $\{v_a, v_b\}$ and $\{v_b, v_c\}$ are in different blocks, say block A and B, respectively. If all edges adjacent to v_b in block A are deleted, the result is $L_{v_i0} > L_{max}$, as intended. The edges adjacent to v_b in block B are treated similarly. If the edges that are deleted have sufficiently small x^*-value, the resulting 2EH is violated. As such, for each articulation point in D_P we may generate one or two inequalities. We treat the current terminal and the depot in a way similar to the articulation points, except that at most one inequality is generated for each of these two endpoints of the path.

If the above procedure does not generate any violated inequalities, we delete from E_D the edge of smallest x^*-value among those edges used in the AP solution and repeat the procedure. This is done as long as the total x^*-value of the deleted edges is sufficiently small to allow for a violated inequality.

Finally we describe how we attempt to reduce F once a violated inequality is found. At this stage we have a $D = (V, E_D)$ such that the inequality with $F = E \setminus E_D$ is valid. Let an *in-path*(i, j) denote any elementary path from i to j which uses only edges in E_D, and let an *out-path*(i, j) denote any elementary path from i to j which uses at least one edge that is not in E_D. Any pair of paths from the terminals to the depot whose total length does not exceed $Q - q(S)$ must satisfy the following disjunction: i) An *out-path*$(v_1, 0)$ whose length is at most $Q - q(S)$ is used, or ii) An *in-path*$(v_1, 0)$ (the length of which is no smaller than L_{v_10}) is used and an *out-path*$(v_2, 0)$ whose length is at most $Q - q(S) - L_{v_10}$ is used. Let $F_1, F_2 \in E \setminus E_D$ be such that every *out-path* in i) uses at least one edge from F_1, and every *out-path* in ii) uses at least one edge from F_2. Then $F = F_1 \cup F_2$ gives a valid inequality. The sets F_1, F_2 are found as follows.

Let the two terminals be numbered such that $L_{v_10} \geq L_{v_20}$, and let $q_0 = 0$. Regarding the *out-paths* from v_1 to 0, let $R_1 = \{r \in V_c \setminus S \colon L_{v_1r} + \frac{q_r}{2} \leq Q - q(S)\}$. Any path from v_1 to 0 in E_D cannot visit any customer outside R_1 without accumulating excessive demand. In any feasible solution, any *out-path*$(v_1, 0)$ whose length does not exceed $Q - q(S)$ must use an edge not in E_D when leaving some customer in $v_1 \cup R_1$. That is, we set $F_1^1 = \{\{v_1, j\} \notin E_D \colon j \in V \setminus S, q_j \leq Q - q(S)\}$, $F_1^2 = \{\{i, j\} \notin E_D \colon i \in R_1, j \in V \setminus S, L_{v_1i} + \frac{q_i}{2} + q_j \leq Q - q(S)\}$, and $F_1 = F_1^1 \cup F_1^2$.

Based on similar arguments for v_2, we set $R_2 = \{r \in V_c \setminus S \colon L_{v_2r} + \frac{q_r}{2} \leq Q - q(S) - L_{v_10}\}$, $F_2^1 = \{\{v_2, j\} \notin E_D \colon j \in V \setminus S, q_j \leq Q - q(S) - L_{v_10}\}$, $F_2^2 = \{\{i, j\} \notin E_D \colon i \in R_2, j \in V \setminus S, L_{v_2i} + \frac{q_i}{2} + q_j \leq Q - q(S) - L_{v_10}\}$, and $F_2 = F_2^1 \cup F_2^2$.

To our knowledge, this is the first published separation heuristic for hypotour inequalities of polynomial time complexity. Indeed, the individual calls of the Hungarian algorithm, the blocking procedure, and the procedure for finding shortest paths in D,

Table 6. Lower bounds using hypotour inequalities.

Name	Aug.	Bla. & Hoch.	Our	Time
E-n30-k3	509.833	509.5	508.5	2
E-n33-k4	834.154	>834	834.353	12
E-n51-k5	515.96	517.8	516.794	8
E-n76-k7	663.261	663.705	663.423	53
E-n76-k8	712.578	713.153	712.908	34
E-n76-k10	793.089	794.154	794.249	81
E-n76-k14	—	—	956.019	80
E-n101-k8	798.097	800.178	798.949	91
E-n101-k14	—	—	1014.803	258
M-n101-k10	820*	—	820*	31
F-n72-k4	233.6	232.558	232.5	3
F-n135-k7	1158.83	1159.25	1159.328	181

are all bounded by $O(n^2)$ time. Since we have only a polynomial number of each, the overall polynomial time follows.

In terms of computational complexity, the advantage of our procedure is that we avoid a complete path enumeration scheme, which may lead to excessive computing times unless accompanied by certain stopping criteria, as in [7, 12].

Using our separation procedure for 2EHs, together with the separation of RCIs, we obtain the results displayed in Table 6, which also displays the results obtained in [7, 12].

Our results are obtained after 25 calls to hypotour separation, after which the bounds increase only very little. None of the three sets of results in Table 6 dominates the two others. We conclude that our procedure gives competitive results in a reasonable time.

2.6. Gomory cuts

Due to the heuristic nature of many of our separation routines, it may happen that no more violated cuts can be found, yet one or more such cuts actually exist. At that stage we can of course resort to branching, but we have found it more useful to use a limited number of *Gomory mixed-integer cuts* [21, 34] to cut off the current vector x^*. Although the addition of these cuts to the LP does not normally lead to a significant increase in the lower bound, we have found that it is an effective method for 'perturbing' x^*, thus giving the separation heuristics a second chance to succeed. However, the Gomory cuts tend to be quite dense, so we only perform this 'trick' at the root node of the branch-and-cut tree, and even then only once.

As in [9], we generate an entire *round* of Gomory cuts rather than only one at a time. That is, for each j such that x_j^* is fractional, we find the associated row of the simplex tableau, which will be of the form:

$$x_j + \sum_{i \in NB} \alpha_i x_i = x_j^*, \tag{12}$$

where NB represents the set of non-basic variables. Then we generate the Gomory mixed-integer cut:

$$\sum_{i \in NB} \min \left\{ \frac{f(\alpha_i)}{f(x_j^*)}, \frac{1 - f(\alpha_i)}{1 - f(x_j^*)} \right\} x_i \geq 1, \tag{13}$$

where for any real r, $f(r) := r - \lfloor r \rfloor$. All of the resulting cuts are added to the problem (after duplicates have been eliminated).

3. The branch-and-cut algorithm

3.1. Separation strategy

In order to design an efficient branch-and-cut algorithm, it is important not only to have efficient separation algorithms, but also to decide which separation algorithms will be called and when. We spent some time experimenting with various strategies, using only the instances in Table 1. In the end it proved best to treat the root node differently to the other nodes of the branching tree.

Our strategy for the root node is as follows. First we call capacity separation. If at least one RCI is violated by more than 0.2, we reoptimize the LP. Otherwise we consider separating framed capacity inequalities (FCIs). We call FCI separation only if there are at least $k(V_c) - 1$ *incompatible* supervertices in the shrunk graph GS_c, where two supervertices are incompatible if their total demand exceeds Q. We allow at most 20000 nodes to be explored in the search tree for FCI separation. If at least one violated FCI is found, we reoptimize the LP. Otherwise we proceed to the remaining classes of inequalities. Among the three classes of multistar, comb, and hypotour inequalities we have found it effective to dynamically change the order of separation. Specifically, we use a cyclic ordering of the three classes, so that each individual class is the first to be called every third time. If one of these classes does not find any inequality whose violation is greater than a certain limit (0.05 for multistars, 0.1 for combs, and 0.1 for hypotours), we proceed to the next class. Finally, as mentioned in Subsection 2.6, when tailing off occurs for all of these classes for the first time, we generate a round of Gomory cuts. When tailing off occurs for the second time, we finish separation at the root node.

Our strategy for the non-root nodes is simpler. We spend less time on separation at these nodes, because we have found it to be more effective to branch earlier than to spend much time on separation. First we call the connected components heuristic for RCIs to ensure that the support graph is connected. Then we call all other separation routines once (here we allow only 100 nodes in the search tree for FCI separation). All violated inequalities found, regardless of their type, are added to the LP. The LP is then reoptimized. Subsequently we separate RCIs only until no violated RCIs are identified.

3.2. Branching

If the separation algorithms tail off, and the solution is not integer, we branch. The standard way to branch is to branch on edge variables, i.e., to select a fractional binary

variable x_e^* and impose the disjunction $(x_e = 0) \vee (x_e = 1)$. Another commonly found technique [33] is to branch on *cutsets*: a set S is found for which $2 < x^*(\delta(S)) < 4$, and the disjunction $(x(\delta(S)) = 2) \vee (x(\delta(S)) \geq 4)$ is imposed. Note that this reduces to standard edge branching when $|S| = 2$.

We choose several candidate sets $S \subset V_c$ for which $x^*(\delta(S))$ is close to 3 by a simple greedy heuristic and sort them in non-decreasing order of $|x^*(\delta(S)) - 3|/q(S)$, giving an ordered list of sets $S_1, S_2, \ldots$. We proceed through the sets one by one. For a given S_i we compute the two lower bounds LB_i^1, LB_i^2 which would be obtained at the two descendant nodes if we used S_i for branching. If one of the descendant nodes can be fathomed, S_i is chosen immediately. Otherwise we use the following rule.

Let $LB_i^- = \min\{LB_i^1, LB_i^2\}$ and $LB_i^+ = \max\{LB_i^1, LB_i^2\}$. We prefer S_i to S_j if $\lceil LB_i^- \rceil > \lceil LB_j^- \rceil$. In the case of ties, we prefer S_i to S_j if $\lceil LB_i^+ \rceil > \lceil LB_j^+ \rceil$. If ties are still present, we use the same conditions, but without the rounding up.

We stop working through the list when the last two sets have not yielded any improvement. The best set found so far is then used for branching. That is, after having examined some $p \geq 3$ candidate sets $S_1, \ldots, S_p$ in this order, we choose set S_{p-2} for branching.

3.3. Node selection

Our branch-and-bound algorithm uses the *best bound first* node selection rule, so that the node with the smallest lower bound is always processed next. This is known to always lead to the smallest possible branch-and-bound tree, but it has the disadvantage that the entire tree (i.e., all currently non-fathomed nodes) must be stored in memory.

In fact we use a slightly modified version of the best bound first rule, taking into account the fact that, for our test instances, all objective coefficients are integer. Let the smallest lower bound equal LB^*. If there are several subproblems whose lower bounds are not greater than $\lceil LB^* \rceil$, we choose among them the subproblem at the deepest level in the search tree.

3.4. Cut pool management

As usual in branch-and-cut algorithms [33, 37], we maintain a cut pool which contains (some or all of) the cuts generated so far in the algorithm. One purpose of the cut pool during branching is to keep the cuts which allow us to reconstruct the LP at any active subproblem. Another purpose is to be able to separate faster (and possibly more effectively, if the separation procedures are heuristic) using the cut pool instead of generating new violated inequalities. However, the effort spent on managing a very large cut pool may outweigh the advantages, i.e., when the cut pool has grown to a certain size, it may be advantageous to permanently delete some cuts from the pool. After some experimenting, we have found the following to give a reasonable trade-off.

At the root node, we keep all generated cuts, except those in the LP, in a cut pool. Whenever the LP has been reoptimized, we first check all cuts in the cut pool, and reoptimize the LP, if any violated cuts have been found. Only when no cuts in the cut pool are violated do we call our separation routines for generating new violated inequalities.

We attempt to keep the LP small by storing cuts of positive slack in the cut pool. When separation finishes at the root node, we permanently delete all cuts of positive slack.

At the non-root nodes, we use a somewhat different strategy. For a given LP, we call a cut *binding* if its slack variable is non-basic in the optimal solution. For every 50 nodes that have been processed in the search tree, we permanently delete from the cut pool every cut which is not binding in at least one of the active nodes in the tree.

3.5. Basis reconstruction

As mentioned in [39], it is important to be able to reconstruct the LP solution at a given subproblem in an efficient way. For this purpose we store at each subproblem information on the set of binding cuts and on the structure of the optimal basis. This information is obtained when the branching set at the parent node is chosen. Despite the additional memory requirement involved, we have found this to give a considerable speed up of the entire algorithm.

4. Computational experiments

In this section we show the performance of the entire algorithm. We give results not only for the instances already mentioned, but also for two further 'E' instances, an additional 'M' instance, and all of the 'A', 'B', and 'P' instances (except one 'P' instance) produced by Augerat [6]. All of these instances are available on www.branchandcut.org. Moreover, they are all (with the exception of `E-n13-k4` and `E-n31-k7`) Euclidean, with integer edge costs following the TSPLIB standard [41]. For the instances in the class 'A', both customer locations and demands are random. The instances in class 'B', however, are *clustered* instances. The instances in class 'P' are modified versions of instances from the literature.

Table 7 shows the results for the instances of Christofides & Eilon [15]. The first two columns give the instance name and the upper bound, as described in Section 1. The next two columns give the lower bound obtained at the root node, using all separation

Table 7. Results for the E instances.

		Root node		Branch & cut	
Instance	UB	LB	Time	Time (LB)	Tree size
E-n13-k4	247*	247*	4	—	—
E-n22-k4	375*	375*	2	—	—
E-n23-k3	569*	569*	2	—	—
E-n30-k3	534*	534*	14	—	—
E-n31-k7	379*	377.028	11	28	10
E-n33-k4	835*	834.707	12	16	3
E-n51-k5	521*	519	24	59	17
E-n76-k7	682*	666.408	72	118683	8631
E-n76-k8	735*	717.852	136	(729)	2321
E-n76-k10	830	799.878	158	(816)	2209
E-n76-k14	1021	969.609	181	(986)	2081
E-n101-k8	815*	802.646	222	(811)	1621
E-n101-k14	1071	1026.94	555	(1040)	917

Table 8. Results for the F and M instances.

		Root node		Branch & cut	
Instance	UB	LB	Time	Time	Tree size
F-n45-k4	724*	724*	6	—	—
F-n72-k4	237*	237	38	40	3
F-n135-k7	1162*	1160	155	3092	265
M-n101-k10	820*	820*	33	—	—
M-n121-k7	1034	1017.422	979	(1025)	631

routines, and the time spent to obtain it (in seconds). The next column gives the total time taken to solve the problem to optimality. When the code failed to find the optimum in a reasonable amount of time, we give in parentheses the best lower bound obtained after eight hours. The final column gives the total number of nodes generated in the branch-and-cut tree, either to prove optimality or after eight hours.

The algorithm solved the first four instances at the root node, and the next three instances were solved after a few branches. After running the code for about 33 hours we obtained the optimal solution to `E-n76-k7`. The remaining five instances appear to be extremely difficult, at least with the currently known classes of cutting planes. This is in keeping with the experience of other authors [7, 12, 33, 40].

Table 8 gives results for the instances of Fisher [20] and two of the Christofides, Mingozzi & Toth [16] instances. Here, two were solved at the root node and two others were solved spending a relatively short amount of time branching. The instance `M-n121-k7` remained unsolved after eight hours.

Next, Table 9 shows the results for the 'A' instances of Augerat. The algorithm solved 17 out of the 27 instances in under an hour, and a further three were solved in a reasonable amount of time. The remaining seven, however, remained unsolved.

In Table 10 we present the results for the 'B' instances of Augerat. The algorithm solved 5 of the 23 instances at the root node, and a further 13 were solved with less than an hour of branching. We were able to solve four of the remaining five in a reasonable amount of time. As shown on the table, three of these (`B-n50-k8`, `B-n66-k9` and `B-n78-k10`) were solved to optimality for the first time in this paper. The optimal solutions are included in the Appendix for other researchers to verify.

Finally, we present in Table 11 the results for the 'P' instances (except `P-n55-k8`, in which the number of vehicles is not fixed at the minimum possible) of Augerat. The algorithm solved 3 of the 23 instances at the root node, and a further 13 were solved within a reasonable time.

All of the above results were obtained without giving any explicit upper bound to the code as input. Further, we did not use any particular heuristic method in the branch-and-cut process. Our only device for generating upper bounds was the LP itself, i.e., only when an LP solution was feasible and integer did we obtain an upper bound. Of course, any heuristic method (simulated annealing, tabu search, etc.) could be used to provide a good upper bound, which could then be used to reduce the solution time and number of tree nodes. In order to see the potential benefit of such a heuristic, we took some of the instances with known optimal solutions, and ran the code again with the optimum as input. We took only those which originally had more than 100 nodes in the branching tree, making 26 instances in total.

Table 9. Results for the A instances.

		Root node		Branch & cut	
Instance	UB	LB	Time	Time (LB)	Tree size
A-n32-k5	784*	782.028	12	15	3
A-n33-k5	661*	658.444	12	23	7
A-n33-k6	742*	733.476	11	38	15
A-n34-k5	778*	768.03	11	27	8
A-n36-k5	799*	790.218	10	51	24
A-n37-k5	669*	665.497	11	25	8
A-n37-k6	949*	925.165	12	531	304
A-n38-k5	730*	717.2	7	116	60
A-n39-k5	822*	810.134	39	138	53
A-n39-k6	831*	817.253	13	109	57
A-n44-k6	937*	921.818	31	620	211
A-n45-k6	944*	930.002	19	157	62
A-n45-k7	1146*	1115.478	66	19414	4170
A-n46-k7	914*	912.063	40	50	3
A-n48-k7	1073*	1055.145	35	372	113
A-n53-k7	1010*	998.7	24	363	107
A-n54-k7	1167*	1135.312	30	7246	1643
A-n55-k9	1073*	1058.282	17	468	152
A-n60-k9	1354	1319.634	63	(1342)	4627
A-n61-k9	1034*	1010.211	45	68636	8608
A-n62-k8	1290	1251.68	231	(1282)	3197
A-n63-k9	1616	1580.667	117	(1608)	3411
A-n63-k10	1315	1266.619	64	(1304)	3911
A-n64-k9	1402	1351.619	132	(1381)	3511
A-n65-k9	1174*	1155.175	45	1324	253
A-n69-k9	1159	1114.373	79	(1147)	3189
A-n80-k10	1763	1709.645	201	(1734)	1773

Table 10. Results for the B instances.

		Root node		Branch & cut	
Instance	UB	LB	Time	Time (LB)	Tree size
B-n31-k5	672*	672*	8	—	—
B-n34-k5	788*	784.25	12	36	19
B-n35-k5	955*	955*	5	—	—
B-n38-k6	805*	801	10	37	21
B-n39-k5	549*	549*	9	—	—
B-n41-k6	829*	827	9	42	24
B-n43-k6	742*	735.417	14	125	63
B-n44-k7	909*	909*	8	—	—
B-n45-k5	751*	748.68	14	46	21
B-n45-k6	678*	673.801	18	299	159
B-n50-k7	741*	741*	11	—	—
B-n50-k8	1313	1281.139	26	31026 (1312*)	5694
B-n51-k7	1032*	1025.571	7	209	122
B-n52-k7	747*	746	8	25	15
B-n56-k7	707*	705.018	20	46	14
B-n57-k7	1153*	1150.092	33	441	168
B-n57-k9	1598*	1589.23	49	1366	264
B-n63-k10	1496*	1481	31	6513	1752
B-n64-k9	861*	860.5	17	42	13
B-n66-k9	1318	1298.509	80	24424 (1316*)	4614
B-n67-k10	1032*	1024.805	28	3309	935
B-n68-k9	1275	1258.054	65	(1267)	5245
B-n78-k10	1221	1205.55	114	87408 (1221*)	8641

Table 11. Results for the P instances.

		Root node		Branch & cut	
Instance	UB	LB	Time	Time (LB)	Tree size
P-n16-k8	450*	449.587	8	10	3
P-n19-k2	212*	212*	5	—	—
P-n20-k2	216*	215.08	11	15	3
P-n21-k2	211*	211*	3	—	—
P-n22-k2	216*	216*	16	—	—
P-n22-k8	603*	594.041	13	44	16
P-n23-k8	529*	519.095	10	69	26
P-n40-k5	458*	457.272	15	19	3
P-n45-k5	510*	505.094	12	76	35
P-n50-k7	554*	542.406	25	805	263
P-n50-k8	649	602.138	28	(625)	5241
P-n50-k10	696*	668.486	29	73016	12551
P-n51-k10	741*	716.107	46	82469	11731
P-n55-k7	568*	550.053	26	11178	2507
P-n55-k10	699	662.119	53	(684)	5081
P-n55-k15	—	906.7	36	(934)	4441
P-n60-k10	756	718.395	44	(739)	4257
P-n60-k15	1033	929.802	84	(949)	4008
P-n65-k10	792	767.094	54	(786)	3585
P-n70-k10	834	795.615	90	(812)	2607
P-n76-k4	593*	589.097	55	535	141
P-n76-k5	627*	617.52	92	10970	1530
P-n101-k4	681*	678.6	127	281	29

The results are shown in Table 12. The total solution time was decreased by around 33%, and the total number of nodes was decreased by around 38%. However, note the anomalous behaviour on `B-n50-k8` and `B-n57-k9`, where both figures have *increased*. This is typical in branch-and-cut, where a small change can have counter-intuitive effects. In this case, the cause is the branching rule in Subsection 3.2: if one of the quantities LB_i^1, LB_i^2 exceeds the optimum minus one, then the algorithm will be immediately guided down the opposite branch, leading to different fractional solutions than would have been encountered otherwise.

Finally we address the issue of relating our results to those of other groups. Both analytical and computational results have been obtained by various groups working on branch-and-cut for the CVRP. However, as remarked in [40], it is unclear to which extent meaningful comparisons can be made between the computational results of various groups, not only due to differences in hardware and implementational details, but also because of considerable limitations with respect to the set of instances for which computational results are reported. (We believe that our inclusion of many more instances than those normally used is a measure to remedy the latter difficulty, looking ahead to yet unknown results of future groups.) Still another difficulty is that to the extent that results are reported only for instances which also have been used for tuning of parameters, it is doubtful whether the results reported are representative of those that would be encountered on other yet unknown instances. The following comparisons are therefore made with reservations.

A comparison with Blasum & Hochstättler ([12]) must be rather rudimentary, as they display results for only ten instances. We note that they managed to solve `E-n76-k7` in

Table 12. Results with the optimum as input.

		Without Opt		With Opt	
Instance	Opt	Time	Tree size	Time	Tree size
E-n76-k7	682*	118683	8631	81526	5824
F-n135-k7	1162*	3092	265	428	23
A-n37-k6	949*	531	304	385	139
A-n44-k6	937*	620	211	426	98
A-n45-k7	1146*	19414	4170	12518	2434
A-n48-k7	1073*	372	113	325	70
A-n53-k7	1010*	363	107	276	55
A-n54-k7	1167*	7246	1643	5503	997
A-n55-k9	1073*	468	152	475	129
A-n61-k9	1034*	68636	8608	46938	5245
A-n65-k9	1174*	1324	253	783	112
B-n45-k6	678*	299	159	130	42
B-n50-k8	1312*	31026	5694	34164	6035
B-n51-k7	1032*	209	122	164	66
B-n57-k7	1153*	441	168	182	47
B-n57-k9	1598*	1366	264	1572	269
B-n63-k10	1496*	6513	1752	3744	896
B-n66-k9	1316*	24424	4614	11460	1864
B-n67-k10	1032*	3309	935	2013	566
B-n78-k10	1221*	87408	8641	53272	4746
P-n50-k7	554*	805	263	514	125
P-n50-k10	696*	73016	12551	51493	7836
P-n51-k10	741*	82469	11731	45942	6239
P-n55-k7	568*	11178	2507	8785	1683
P-n76-k4	593*	535	141	280	50
P-n76-k5	627*	10970	1530	8340	1013
	Total:	554717	75529	371638	46603

less than 8 hours on a slower (400 MHz) computer, and that they also solved the instance `E-n76-k8`. Based on the few instances that are solved by both their code and our code, none of the two codes dominates the other. A more rigorous conclusion seems not to be reasonable, given their strongly limited set of test instances.

Ralphs et al. ([40]), who used hardware almost identical to ours, solved the instance `E-n76-k7` in about 80 hours, whereas we solved it in about 33 hours (or about 23 hours if the optimal value is available initially as an upper bound). Using a parallel computer, Ralphs et al. also solved the instances `E-n76-k8` and `E-n101-k8`. They also mention that they unsuccessfully tried to solve `B-n50-k8`, which is one of the instances that was solved for the first time by our code. More generally, with respect to those instances that are solved in [40] as well as by our code, the overall picture seems to be that the code of [40] is faster on the easier instances, whereas our code is faster on the harder instances. On this basis we consider our code to compare favourably with that of [40].

5. Conclusion

We have presented a new branch-and-cut algorithm for the CVRP. Our algorithm uses several classes of valid inequalities as cutting planes, and we have described our separation algorithms for each class in detail. For a special kind of hypotour inequalities

we have presented a separation heuristic which runs in polynomial time. Moreover, we have presented a separation heuristic for strengthened comb inequalities.

We have also given the results of computational experiments on a large number of test instances, and shown that our algorithm is competitive with others. With our branch-and-cut algorithm we were able to solve the three instances `B-n50-k8`, `B-n66-k9`, and `B-n78-k10` to proven optimality for the first time. We have also presented the best known upper bound at the time of writing for each test instance, which may be of use to other researchers.

In our view, the most pressing problem for research in this field is to understand why certain instances (such as the 'E' instances with 76 vertices) are so difficult. It is possible that there exists an unknown class of valid inequalities which would be effective for these instances. Finding such a class and devising a suitable separation algorithm for it remains a challenge.

Appendix

In Tables 13, 14 and 15 we give the optimal solutions to the three instances `B-n50-k8`, `B-n66-k9`, and `B-n78-k10`, which are solved to proven optimality for the first time by our algorithm. The vehicle capacity is 100 for each of these instances. In each case we give, for each vehicle route in the solution, the sequence in which the customers are visited, the total load (i.e., the sum of the demands serviced on that route), and the total distance travelled.

Table 13. Optimal solution to `B-n50-k8`.

Route no.	Sequence	Load	Distance
1	1 28 29 19 32	97	243
2	2 40 7 48 34	48	72
3	3 23 41 45 39 35	94	138
4	5 16 43 12	100	206
5	6 30 36 49 33 9 27 37 44	100	99
6	10 21 42 17 8 15	100	234
7	13 38 22 46 14 31	97	115
8	20 11 24 47 26 18 4 25	99	205
Total		735	1312

Table 14. Optimal solution to `B-n66-k9`.

Route no.	Sequence	Load	Distance
1	1 7 35 19 38 57 11 23 18	100	113
2	6 2 10 26 46 63 25	99	162
3	24 36 21 4 60 15 3 29 39 14 27	99	267
4	32 58 40 8 65 33	75	162
5	41 17 20 53 54 52	91	42
6	42 5 9 12 49	98	153
7	45 31 13 56 22 43 34 51	100	115
8	47 59 44 64 30 55	100	155
9	50 28 16 62 48 37 61	99	147
Total		861	1316

Table 15. Optimal solution to `B-n78-k10`.

Route no.	Sequence	Load	Distance
1	1 52 24 21 43 67 69 72 17	100	27
2	2 63 49 27 56 38 58	98	68
3	5 30 70 29 37 65	94	79
4	8 22 71 31 73 32 76 7 57 47	99	124
5	9 19 33 55 35 59 4 36 14 41 10	99	222
6	11 64 40 53 68 18 25 42	100	154
7	15 12 34 46 45 51	100	116
8	16 66 60 6 62 54 20 3 75	100	204
9	23 26 44 77	48	66
10	28 74 48 13 39 61 50	99	161
Total		937	1221

References

1. Achuthan, N.R., Caccetta, L., Hill, S.P.: “Capacitated vehicle routing problem: some new cutting planes”. Asia-Pac. J. Oper. Res. **15**, 109–123 (1998)
2. Agarwal, Y., Mathur, K., Salkin, H.M.: “A set-partitioning-based exact algorithm for the vehicle routing problem”. Networks **19**, 731–749 (1989)
3. Araque, J.R.: “Lots of combs of different sizes for vehicle routing”. Discussion paper. Center for Operations Research and Econometrics. Catholic University of Louvain, Belgium, 1990
4. Araque, J.R., Hall, L.A., Magnanti, T.L.: “Capacitated trees, capacitated routing and associated polyhedra”. Discussion paper. Center for Operations Research and Econometrics, Catholic University of Louvain, Belgium, 1990
5. Araque, J.R., Kudva, G., Morin, T.L., Pekny, J.F.: “A branch-and-cut algorithm for the vehicle routing problem”. Ann. Oper. Res. **50**, 37–59 (1994)
6. Augerat, P.: Approche Polyèdrale du Problème de Tournées de Véhicules. PhD thesis, Institut National Polytechnique de Grenoble, 1995
7. Augerat, P., Belenguer, J.M., Benavent, E., Corberán, A., Naddef, D., Rinaldi, G.:“Computational results with a branch-and-cut code for the capacitated vehicle routing problem”. Research report RR949-M. ARTEMIS-IMAG, France, 1995
8. Augerat, P., Belenguer, J.M., Benavent, E., Corberán, A., Naddef, D.: “Separating capacity constraints in the CVRP using tabu search”. Eur. J. Opl. Res. **106**, 546–557 (1998)
9. Balas, E., Ceria, S., Cornuéjols, G., Natraj, N.: “Gomory cuts revisited”. Oper. Res. Lett. **19**, 1–10 (1996)
10. Ball, M.O., Magnanti, T.L., Monma, C.L., Nemhauser, G.L.: (eds.) Network Routing. Handbooks on Operations Research and Management Science, **8**. Amsterdam: Elsevier, 1995
11. Blasum, U.: “Anwendung des Branch & Cut Verfahrens auf das kapazitierte Vehicle-Routing Problem”. PhD thesis, Universität zu Köln, 1999
12. Blasum, U., Hochstättler, W.: “Application of the branch-and-cut method to the vehicle routing problem”. Technical report. Universität zu Köln, 2002
13. Campos,V., Corberán, A., Mota, E.: “Polyhedral results for a vehicle routing problem”. Eur. J. Opl. Res. **52**, 75–85 (1991)
14. Carpaneto, G., Martello, S., Toth, P.:“Algorithms and codes for the assignment problem”. Ann. Oper. Res. **13**, 193–223 (1988)
15. Christofides, N., Eilon, S.: “An algorithm for the vehicle dispatching problem”. Oper. Res. Quarterly **20**, 309–318 (1969)
16. Christofides, N., Mingozzi, A., Toth, P.:“Exact algorithms for the vehicle routing problem based on the spanning tree and shortest path relaxations”. Math. Program. **20**, 255–282 (1981)
17. Cornuéjols, G., Harche, F.: “Polyhedral study of the capacitated vehicle routing problem”. Math. Program. **60**, 21–52, (1993)
18. Dantzig, G.B., Ramser, R.H.: “The truck dispatching problem”. Manage. Sci. **6**, 80–91 (1959)

19. Edmonds, J.: "Maximum matching and a polyhedron with 0–1 vertices". J. Res. Nat. Bur. Standards. **69B**, 125–130 (1965)
20. Fisher, M.L.: "Optimal solution of vehicle routing problems using minimum K-trees". Oper. Res. **42**, 626–642 (1994)
21. Gomory, R.E.: "An algorithm for the mixed-integer problem". Report RM-2597, Rand Corporation, 1960 (unpublished)
22. Gouveia, L.: "A result on projection for the vehicle routing problem". Eur. J. Opl. Res. **85**, 610–624 (1995)
23. Grötschel, M., Lovász, L., Schrijver, A.J.: Geometric Algorithms in Combinatorial Optimization. Springer, 1988
24. Grötschel, M., Padberg, M.W.: "On the symmetric travelling salesman problem I: inequalities". Math. Program. **16**, 265–280 (1979)
25. Grötschel, M., Padberg, M.W.: "On the symmetric travelling salesman problem II: lifting theorems and facets". Math. Program. **16**, 281–302 (1979)
26. Hadjiconstantinou, E., Christofides, N., Mingozzi, A.: "A new exact algorithm for the vehicle routing problem based on q-paths and k-shortest paths relaxations". Ann. Oper. Res. **61**, 21–43 (1996)
27. Laporte, G.: "The vehicle routing problem: an overview of exact and approximate algorithms". Eur. J. Opl. Res. **59**, 345–358, (1992)
28. Laporte, G.: "Vehicle routing". In: Dell'Amico, Maffioli, Martello (eds.) Annotated Bibliographies in Combinatorial Optimization. New York, Wiley, 1997
29. Laporte, G., Nobert, Y.: "Comb inequalities for the vehicle routing problem". Methods of Oper. Res. **51**, 271–276 (1984)
30. Letchford, A.N., Eglese, R.W., Lysgaard, J.: "Multistars, partial multistars and the capacitated vehicle routing problem". Math. Program. **94**, 21–40 (2002)
31. Martello, S., Toth, P.: Knapsack Problems: Algorithms and Computer Implementations. Chichester: Wiley, 1990
32. Miller, D.L.: "A matching-based exact algorithm for capacitated vehicle routing problems". ORSA J. Comp. **7**, 1–9 (1995)
33. Naddef, D., Rinaldi, G.: "Branch-and-cut algorithms for the capacitated VRP". In: P.Toth, D.Vigo (eds.), The Vehicle Routing Problem. SIAM Monographs on Discr. Math. Appl. **9**, 2002
34. Nemhauser, G.L., Wolsey, L.A.: Integer and Combinatorial Optimization. New York: Wiley, 1988
35. Padberg, M.W., Rao, M.R.: "Odd minimum cut-sets and b-matchings". Math. Oper. Res. **7**, 67–80 (1982)
36. Padberg, M.W., Rinaldi, G.: "Facet identification for the symmetric traveling salesman polytope". Math. Program. **47**, 219–257 (1990)
37. Padberg, M.W., Rinaldi, G.: "A branch-and-cut algorithm for the resolution of large-scale symmetric travelling salesman problems". SIAM Rev. **33**, 60–100 (1991)
38. Pereira, F.B., Tavares, J., Machado, P., Costa, E.: "GVR: a new genetic representation for the vehicle routing problem". In: M.O'Neill et al.(eds.), Proceedings of AICS 2002. Berlin: Springer-Verlag, 2002, pp. 95–102
39. Ralphs, T.K.: "Parallel branch and cut for capacitated vehicle routing". To appear in Parallel Computing
40. Ralphs, T.K., Kopman, L., Pulleyblank, W.R., Trotter, L.E.: "On the capacitated vehicle routing problem". Math. Program. **94**, 343–359 (2003)
41. Reinelt, G.: "TSPLIB: A travelling salesman problem library". ORSA J. Comp. **3**, 376–384 (1991)
URL: http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/
42. Toth, P., Vigo, D.: (eds.), The Vehicle Routing Problem. SIAM Monographs on Discr. Math. Appl. **9**, (2002)
43. Xu, J., Kelly, J.P.: "A network flow-based tabu search heuristic for the vehicle routing problem". Transportation Sci. **30**, 379–393 (1996)