Math. Program., Ser. A 106, 491–511 (2006)

Digital Object Identifier (DOI) 10.1007/s10107-005-0644-x

Ricardo Fukasawa · Humberto Longo · Jens Lysgaard · Marcus Poggi de Aragão
Marcelo Reis · Eduardo Uchoa · Renato F. Werneck

Robust Branch-and-Cut-and-Price for the Capacitated Vehicle Routing Problem

Received: November 3, 2004 / Accepted: July 30, 2005
Published online: October 12, 2005 –

Abstract. The best exact algorithms for the Capacitated Vehicle Routing Problem (CVRP) have been based on either branch-and-cut or Lagrangean relaxation/column generation. This paper presents an algorithm that combines both approaches: it works over the intersection of two polytopes, one associated with a traditional Lagrangean relaxation over q-routes, the other defined by bound, degree and capacity constraints. This is equivalent to a linear program with exponentially many variables and constraints that can lead to lower bounds that are superior to those given by previous methods. The resulting branch-and-cut-and-price algorithm can solve to optimality all instances from the literature with up to 135 vertices. This more than doubles the size of the instances that can be consistently solved.

1. Introduction

Let $G = (V, E)$ be an undirected graph with vertices $V = \{0, 1, \ldots, n\}$ and edges $E = \{1, \ldots, m\}$. Vertex 0 represents the *depot*, and each remaining vertex i represents a *client* with an associated positive demand d_i. Each edge $e \in E$ has a nonnegative length ℓ_e. Given G and two positive integers (K and C), the *Capacitated Vehicle Routing Problem* (CVRP) consists of finding routes for K vehicles satisfying the following constraints: (i) each route starts and ends at the depot, (ii) each client is visited by a single vehicle, and (iii) the total demand of all clients in any route is at most C. The goal is to minimize the sum of the lengths of all routes. This classical NP-hard problem is a natural generalization of the Travelling Salesman Problem (TSP), and has widespread application itself. The CVRP was first proposed in 1959 by Dantzig and Ramser [14] and has received close attention from the optimization community since then.

R. Fukasawa: School of Industrial and Systems Engineering, GeorgiaTech, USA.
e-mail: rfukasaw@isye.gatech.edu

H. Longo: Instituto de Informática, Universidade Federal de Goiás, Brazil.
e-mail: longo@inf.ufg.br

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

M. Poggi de Aragão · Marcelo Reis: Departamento de Informática, PUC Rio de Janeiro, Brazil.
e-mail: {poggi,mreis}@inf.puc-rio.br

E. Uchoa (✉): Departamento de Engenharia de Produção, Universidade Federal Fluminense, Brazil.
e-mail: uchoa@producao.uff.br

R. F. Werneck: Department of Computer Science, Princeton University, USA.
e-mail: rwerneck@cs.princeton.edu

Mathematics Subject Classification (2000): 20E28, 20G40, 20C20

A landmark exact algorithm for the CVRP, presented in 1981 by Christofides, Mingozzi and Toth [12], uses a Lagrangean bound from minimum q-route subproblems. A *q-route* is a walk that starts at the depot, traverses a sequence of clients with total demand at most C, and returns to the depot. Some clients may be visited more than once, so the set of valid CVRP routes is strictly contained in the set of q-routes. The resulting branch-and-bound algorithm could solve instances with up to 25 vertices, a respectful size at the time.

Several other algorithms using Lagrangean relaxation appear in the literature. Christofides et al. [12] also describe a lower bound based on k-degree center trees, which are minimum spanning trees having degree $K \leq k \leq 2K$ at the depot, plus $2K - k$ least-cost edges. Lagrangean bounds based on K-trees (sets of $n + K - 1$ edges spanning V) having degree $2K$ at the depot were used by Fisher [17] and by Martinhon, Lucena, and Maculan [31], among others. Miller [32] presented an algorithm based on minimum b-matchings having degree $2K$ at the depot and 2 on the remaining vertices. Lagrangean bounds can be improved by dualizing capacity inequalities [17, 32] and also comb and multistar inequalities [31].

Another family of exact algorithms stems from the formulation of the CVRP as a set partitioning problem by Balinski and Quandt [8]. A column covers a set of vertices S with total demand not exceeding C and has the cost of a minimum route over $\{0\} \cup S$. Unfortunately, the formulation is not practical because pricing over the exponential number of columns requires solving capacitated prize-collecting TSPs, a problem almost as difficult as the CVRP itself. Agarwal, Marthur, and Salkin [3] proposed a column generation algorithm on a modified set partitioning problem where column costs are given by a linear function over the vertices that yields a lower bound on the actual route cost. Columns with the modified cost can be priced by solving easy knapsack problems. Hadjiconstantinou et al. [20] derive lower bounds from heuristic solutions to the dual of the set partitioning formulation. The dual solutions are obtained by the so-called additive approach, combining the q-route and k-shortest path relaxations. For further information and comparative results on the algorithms mentioned above, we refer the reader to the surveys by Toth and Vigo [39, 40].

It should be noted that, since the seminal work by Desrochers, Desrosiers, and Solomon [15], column generation has been the dominant approach for the Vehicle Routing Problem with Time Windows (VRPTW). Current branch-and-price algorithms (such as [21]) can consistently solve tightly constrained instances (those with narrow time windows) with up to 100 clients. However, they often fail on less constrained instances with only 50 clients. Therefore, pure column generation has not been regarded as a promising approach for the CVRP, since it can be viewed as a particular case of the VRPTW with arbitrarily large time windows.

Recent research on the CVRP has been concentrated on the polyhedral description of the convex hull of the edge incidence vectors that correspond to K feasible routes and on the development of effective separation procedures [1, 4, 6, 7, 13, 25, 33]. In particular, Araque et al. [5], Augerat et al. [7], Blasum and Hochstättler [10], Ralphs et al. [38], Achuthan, Caccetta, and Hill [2] and Lysgaard, Letchford, and Eglese [30] describe complete branch-and-cut (BC) algorithms. These are the best exact methods currently available for the CVRP. However, the addition of several elaborate classes of cuts does not guarantee tight lower bounds, especially for large values of K ($K \geq 7$,

say). Closing the resulting duality gap usually requires exploring several nodes in the branch-and-cut tree. Even resorting to massive computational power (up to 80 processors running in parallel in a recent work by Ralphs [37, 38]) several instances with fewer than 80 vertices, including some proposed more than 30 years ago by Christofides and Eilon [11], can not be solved at all. In fact, branch-and-cut algorithms for the CVRP seem to be experiencing a "diminishing returns" stage, where substantial theoretical and implementation efforts achieve practical results that are only marginally better than those of previous works.

We present a new exact algorithm for the CVRP that seems to break through this situation. The main idea is to combine the branch-and-cut approach with the q-routes approach (which we interpret as column generation instead of the original Lagrangean relaxation) to derive superior lower bounds. Since the resulting formulation has an exponential number of both columns and rows, this leads to a branch-and-cut-and-price (BCP) algorithm. Computational experiments over the main instances from the literature show that this algorithm is very consistent, solving all instances with up to 135 vertices. Eighteen open instances are solved for the first time.

The idea of combining column and cut generation to improve lower bounds has rarely been used, since new dual variables corresponding to separated cuts may have the undesirable effect of changing the structure of the pricing subproblem, making it intractable. In the late 1990's, several researchers [9, 16, 22, 23, 41, 42] independently noted that cuts expressed in terms of variables from a suitable original formulation could be dynamically separated, translated and added to the master problem. Those cuts do not change the structure of the pricing subproblem. This observation allowed the construction of what we call *robust branch-and-cut-and-price* algorithms.

Poggi de Aragão and Uchoa [36] present a detailed discussion on this subject, including new reformulation techniques that extend the applicability of robust branch-and-cut-and-price algorithms to virtually any combinatorial optimization problem. This article on the CVRP is part of a larger effort to demonstrate that these methods lead to significant improvements on a wide variety of problems. Major advances have already been reported on three other problems: capacitated minimum spanning tree [18], generalized assignment [35] and capacitated arc routing problem [28].

This article is organized as follows. Section 2 describes the integer programming formulation we will deal with. Section 3 gives a general description of our algorithm, including its two main components: column and cut generation. Following the work of Irnich and Villeneuve [21] on the VRPTW, our column generation procedure eliminates q-routes with small cycles. The separation routines are based on the families of inequalities recently discussed by Letchford, Eglese, and Lysgaard [25, 30]. Section 4 presents an empirical analysis of our method. Final remarks are made in Section 5.

2. The New Formulation

A classical formulation for the CVRP [24] represents by x_e the number of times a vehicle traverses edge $e = \{i, j\} \in E$. The set of client vertices is denoted by $V_+ = \{1, \ldots, n\}$. Given a set $S \subseteq V_+$, let $d(S)$ be the sum of the demands of all vertices in S, and let $\delta(S)$ be the cut-set defined by S. Also, let $k(S) = \lceil d(S)/C \rceil$. Define the following polytope

in $\mathcal{R}^m$:

$$P_1 = \left\{ \begin{array}{lll} \sum\limits_{e\in\delta(\{i\})} x_e = 2 & \forall\, i \in V_+ & (1) \\ \sum\limits_{e\in\delta(\{0\})} x_e = 2\cdot K & & (2) \\ \sum\limits_{e\in\delta(S)} x_e \geq 2\cdot k(S) & \forall\, S \subseteq V_+ & (3) \\ x_e \leq 1 & \forall\, e \in E \setminus \delta(\{0\}) & (4) \\ x_e \geq 0 & \forall\, e \in E\ . & \end{array} \right.$$

Constraints (1) state that each client is visited once by some vehicle, whereas (2) states that K vehicles must leave and enter the depot. Constraints (3) are rounded capacity inequalities, which require all subsets to be served by enough vehicles. Constraints (4) enforce that each edge not adjacent to the depot is traversed at most once (edges adjacent to the depot can be used twice when a route serves only one client). The integer vectors x in P_1 define all feasible solutions for the CVRP. There are exponentially many inequalities of type (3), so the lower bound given by

$$L_1 = \min_{x\in P_1} \sum_{e\in E} \ell_e x_e$$

must be computed by a cutting plane algorithm.

Alternatively, a formulation with an exponential number of columns can be obtained by defining variables (columns) that correspond to q-routes without 2-cycles (subpaths $i \to j \to i$, $i \neq 0$). Restricting the q-routes to those without such cycles improves the formulation and does not change the complexity of the pricing [12]. Let Q be an $m \times p$ matrix where the columns are the edge incidence vectors of all p such q-routes. Let q^e_j be the coefficient associated with edge e in the j-th column of Q. Consider the following polytope in $\mathcal{R}^m$, defined as the projection of a polytope in $\mathcal{R}^{p+m}$:

$$P_2 = \text{proj}_x \left\{ \begin{array}{llll} \sum\limits_{j=1}^{p} q^e_j \cdot \lambda_j - & x_e & = 0 \ \ \forall e \in E & (5) \\ \sum\limits_{j=1}^{p} \lambda_j & & = K & (6) \\ & \sum\limits_{e\in\delta(\{i\})} x_e & = 2 \ \ \forall\, i \in V_+ & (1) \\ & x_e & \geq 0 \ \ \forall\, e \in E & \\ \lambda_j & & \geq 0 \ \ \forall\, j \in \{1,\ldots,p\}\ . & \end{array} \right.$$

Constraints (5) define the coupling between variables x and λ. Constraint (6) defines the number of vehicles to use. It can be shown that the set of integer vectors in P_2 also defines all feasible solutions for the CVRP. Due to the exponential number of variables λ, the lower bound given by

$$L_2 = \min_{x\in P_2} \sum_{e\in E} \ell_e x_e$$

must be computed using column generation or Lagrangean relaxation. Incidentally, this is exactly the bound that would be obtained by the column generation algorithm for the VRPTW described in [15] if the time windows constraints were relaxed.

The description of polyhedra associated with column generation or Lagrangean relaxation in terms of two sets of variables, λ and x, used in the definition of P_2, is called *Explicit Master* in [36]. The main contribution of this article is a formulation that amounts to optimizing over the intersection of polytopes P_1 and P_2. The Explicit Master format describes such intersection as follows:

$$P_3 = P_1 \cap P_2 = \text{proj}_x \left\{ \begin{array}{rrll} & \sum_{e\in\delta(\{i\})} x_e & = 2 & \forall\, i \in V_+ \quad (1) \\ & \sum_{e\in\delta(\{0\})} x_e & = 2\cdot K & \quad (2) \\ & \sum_{e\in\delta(S)} x_e & \geq 2\cdot k(S) & \forall\, S \subseteq V_+ \quad (3) \\ & x_e & \leq 1 & \forall\, e \in E\setminus\delta(\{0\}) \quad (4) \\ \sum_{j=1}^{p} q_j^e\cdot\lambda_j - & x_e & = 0 & \forall e \in E \quad (5) \\ \sum_{j=1}^{p} \lambda_j & & = K & \quad (6) \\ & x_e & \geq 0 & \forall\, e \in E \\ \lambda_j & & \geq 0 & \forall\, j \in \{1,\ldots,p\}\ . \end{array} \right.$$

Constraint (6) can be discarded, since it is implied by (2) and (5). Computing the improved lower bound

$$L_3 = \min_{x\in P_3} \sum_{e\in E} \ell_e x_e$$

requires solving a linear program with an exponential number of both variables and constraints. A more compact LP is obtained if every occurrence x_e in (1)–(4) is replaced by its equivalent given by (5). The resulting LP will be referred to as the *Dantzig-Wolfe Master problem* (DWM):

$$\text{DWM} = \left\{ \begin{array}{lrll} L_3 = \min & \sum_{j=1}^{p}\sum_{e\in E} \ell_e\cdot q_j^e\cdot\lambda_j & & \quad (7) \\ \text{s.t.} & \sum_{j=1}^{p}\sum_{e\in\delta(\{i\})} q_j^e\cdot\lambda_j & = 2 \quad \forall\, i \in V_+ & \quad (8) \\ & \sum_{j=1}^{p}\sum_{e\in\delta(\{0\})} q_j^e\cdot\lambda_j & = 2\cdot K & \quad (9) \\ & \sum_{j=1}^{p}\sum_{e\in\delta(S)} q_j^e\cdot\lambda_j & \geq 2\cdot k(S)\ \forall\, S \subseteq V_+ & \quad (10) \\ & \sum_{j=1}^{p} q_j^e\cdot\lambda_j & \leq 1 \quad \forall\, e \in E\setminus\delta(\{0\}) & \quad (11) \\ & \lambda_j & \geq 0 \quad \forall\, j \in \{1,\ldots,p\}\ . & \end{array} \right.$$

Capacity inequalities are not the only ones that can appear in the DWM. A generic cut $\sum_{e\in E} a_e x_e \geq b$ can be included as $\sum_{j=1}^{p}(\sum_{e\in E} a_e q_j^e)\cdot\lambda_j \geq b$. We added framed capacity, strengthened comb, multistar, partial multistar, generalized multistar and hypotour cuts, all described in detail in [25, 30].

3. The Branch-and-Cut-and-Price Algorithm

3.1. Column Generation

The reduced cost of a column (λ variable) is the sum of the reduced costs of the edges in the corresponding q-route. Let μ, ν, π, and ω be the dual variables associated with constraints (8), (9), (10), and (11), respectively. The reduced cost $\bar{c}_e$ of an edge e is given by:

$$\bar{c}_e = \begin{cases} \ell_e - \mu_i - \mu_j - \sum\limits_{S|\delta(S)\ni e} \pi_S - \omega_e & e = \{i, j\} \in E \setminus \delta(\{0\}) \\ \ell_e - \nu - \mu_j - \sum\limits_{S|\delta(S)\ni e} \pi_S & e = \{0, j\} \in \delta(\{0\}) \quad . \end{cases}$$

An additional generic cut $\sum_{j=1}^{p} \left(\sum_{e\in E} a_e q_j^e\right) \cdot \lambda_j \geq b$ in DWM, with dual variable α, contributes with the value $-a_e \cdot \alpha$ to the computation of $\bar{c}_e$.

The pricing subproblem consists of finding q-routes (columns) of minimum reduced cost. Although this problem is NP-hard, it can be solved in pseudopolynomial time. The basic data structure is a $C \times n$ matrix M. Each entry $M(d, v)$ will represent the least costly walk that reaches vertex v using total demand exactly d. The entry contains a *label* consisting of three elements: the vertex identifier (v), the cost of the walk (denoted by $\bar{c}(M(d, v))$), and a pointer to a label representing the walk up to the previous vertex. Initially, the only known label represents an empty path and has cost zero (it can be seen as entry $M(0, 0)$); all other entries are initialized with labels representing empty walks with infinite cost. From entry $M(0, 0)$, we use dynamic programming to fill the matrix, starting with lower values of d. For each row d, the algorithm goes through each entry v and, for each neighbor w of v, evaluates the extension of the walk represented by $M(d, v)$ to w. If $\bar{c}(M(d, v)) + \bar{c}_{\{v,w\}} < \bar{c}(M(d + d_w, w))$, $M(d + d_w, w)$ is updated. Eventually, we will have the most negative walk with accumulated demand at most C that arrives at each vertex v. Extending the walk to the depot (whose demand is zero), we obtain the corresponding q-route. All negative q-routes thus found (there will be at most n, one coming from each vertex) are added to the linear program. There are nC entries in the matrix, and each is processed in $O(n)$ time, so the total running time is $O(n^2C)$.

Cycle elimination. To strengthen the formulation, we could look for q-routes without cycles. Since this problem is strongly NP-hard, we settle for s-cycle-free q-routes, for small values of s.

The algorithm operates as above, using dynamic programming to fill a $C \times n$ matrix with partial walks. Labels retain the exact same meaning as before, but now each entry in the matrix contains no longer a single label, but a *bucket* of labels. Therefore, bucket $M(d, v)$ represents not only the cheapest s-cycle-free walk with total demand d that ends in v, but also alternative walks that ensure that all possible extensions from v with exactly s vertices are considered.

To formalize this notion, we use the concept of *dominance*. A label ℓ is *s-dominated* by a set of labels $\mathcal{L}$ if no label in $\mathcal{L}$ costs more than ℓ and if every path p of length s that can legally extend ℓ (i.e., without creating an s-cycle) can also extend some label of $\mathcal{L}$.

Recall that the algorithm looks for the best (shortest) s-cycle-free walk, and therefore only labels that might represent a segment of this walk need to be considered. Dominated labels will not be part of the best walk. To simplify the presentation, we assume the "best walk" is unique. What the algorithm actually requires is the existence, for every dominated label ℓ, of a minimum-cost s-cycle-free walk that does not contain ℓ. Suppose the best walk did contain a label ℓ dominated by a set $\mathcal{L}$. Label ℓ represents an s-cycle-free walk from the depot to some vertex; let the remainder of the best walk be R. The definition of dominance implies that there exists at least one label $\ell' \in \mathcal{L}$ that can be concatenated with R to create a shorter s-cycle-free route. (Note that ℓ' and R are free of s-cycles by definition; dominance just ensures that there is no s-cycle in the junction.)

Now that we have the concept of dominance, we return to the algorithm. Initially, all buckets are empty. Once an existing label is extended (in a process identical to the original algorithm), a new label is created, and the algorithm tries to insert it into the appropriate bucket. If the label is dominated by the other labels in the bucket, it is simply discarded; otherwise, it is inserted (if other labels in the bucket become dominated as a result, they are discarded).

The trickiest part of this routine is testing for dominance. Christofides, Mingozzi and Toth [12] have shown that, for $s = 2$, one only has to keep the two lower-cost labels. Given any three labels in the same bucket, the one with highest cost will be dominated by the other two. For larger values of s, deciding which labels to keep becomes significantly more complicated, as shown by Irnich and Villeneuve [21]. Our implementation follows their ideas. The reader is referred to their paper for details, but it is worth mentioning that buckets must have size at least $s!$, so the method quickly becomes impractical. For this reason, we only tried values of s up to 4.

Heuristic Acceleration. Since running the dynamic programming procedure in full is slow, it is only natural to employ heuristics to find good q-routes faster. We use three different techniques; when they are used, the algorithm still produces only s-cycle-free q-routes, but not necessarily the minimum one. Hence, when no negative route is found using heuristic acceleration, the full algorithm must be run to attest that none exists.

The first acceleration technique we use is *scaling*, which considers $g > 1$ units of demand at a time. The exact algorithm is run with a modified demand $d'_v(g) = \lceil d_v/g \rceil$ for every vertex v, and modified capacity $C' = \lfloor C/g \rfloor$ for the vehicles. The running time will be proportional to C' instead of C, but the solutions found will still be valid in the original setting.

The second technique is *sparsification*. Intuitively, small edges in the original graph are more likely to appear in the solution. With that in mind, we pre-compute a set of five disjoint spanning forests using an extension of Kruskal's algorithm similar to the "edge-greedy heuristic" described in [44]. By considering only edges in those trees and edges adjacent to the depot, the dynamic programming will be limited to a restricted set of neighbors when processing each vertex.

The third acceleration is *bucket pruning*, which consists of storing only s (instead of $s!$) labels per bucket. To minimize the number of valid s-extensions blocked, we keep only labels that differ significantly from each other. To ensure this, we use the sequence consisting of the last s vertices on the walk represented by the label. A label ℓ will be

maintained in the bucket if there is at least one position in that sequence that contains the same vertex on all labels of lower cost, but a different vertex on ℓ.

We start each pass using all three acceleration techniques, but we turn each of them on or off between LP reoptimizations depending on how successful the algorithm is. Whenever it fails to find a negative q-route using one or more acceleration techniques, it turns one of them off and tries again. On the other hand, finding a negative q-route using only a subset of the heuristics prompts another heuristic to be turned back on. Eventually the algorithm will fail with no acceleration, in which case we can safely declare that no negative q-route exists (and that column generation has converged).

3.2. Cut Generation

Heuristic separation. At first, the LP formulation includes only degree constraints; other cuts are added during the algorithm. Besides the rounded capacity cuts (10) and bound cuts (11), we also use framed capacity, strengthened comb, multistar, partial multistar, generalized multistar and hypotour cuts, all presented in [30]. Those cuts are defined over x variables. We convert a solution $\bar{\lambda}$ from DWM into a corresponding $\bar{x}$. If this solution is fractional, it is given as input to the CVRPSEP package [29]. The violated cuts found are translated back into λ variables to be introduced in the LP.

Exact separation of rounded capacity cuts. Separating inequalities (10), rounded capacity cuts, is an NP-hard problem. We implemented an exact separation procedure using the following mixed-integer model, which assumes that all demands and the vehicle capacity are integral. Let $\bar{x}$ be a fractional solution. Define y_i as a binary variable equal to 1 if vertex i belongs to set $S \subseteq V$ and w_{ij} as a variable that is equal to 1 if edge $\{i, j\}$ belongs to $\delta(S)$. For each value of M in $\{0, \ldots, \lceil d(V_+)/C \rceil - 1\}$, we solve:

$$\text{RCCSep}(M) = \begin{cases} z = \min \sum_{e \in E} \bar{x}_{ij} w_{ij} & \\ \text{s.t.} \quad w_{ij} \geq y_j - y_i & \forall \{i, j\} \in E \\ \quad\; w_{ij} \geq y_i - y_j & \forall \{i, j\} \in E \\ \quad \sum_{i \in V_+} d_i y_i \geq (M \cdot C) + 1 & \\ \quad\; y_0 = 0 & \\ \quad\; y_i \in \{0, 1\} & \forall\, i \in V_+ \\ \quad\; w_{ij} \geq 0 & \forall \{i, j\} \in E. \end{cases}$$

If $z < 2(M + 1)$, then the capacity inequality over set S is violated.

A previous version of the BCP algorithm [19] used this computationally expensive exact separation. Here we only use it to compute the exact values of bounds L_1 and L_3 and compare them with what is obtained by the heuristic separation from [29, 30] (the one we actually use during BCP). We have observed that the heuristics typically decrease separation times by two orders of magnitude without any significant loss in terms of bound quality. Wenger [43] had already observed that the bounds obtained with heuristic separation of RCCs within the branch-and-cut algorithm of Lysgaard, Letchford, and Eglese [30] are indeed very close to L_1.

3.3. Branch-and-Cut-and-Price Details

Representation of Variables and Constraints. When implementing a branch-and-cut-and-price it is necessary to have a clear distinction between variables and columns, and also between constraints and rows. Each λ variable is associated to a q-path, which is stored in a pool of variables. Every time a new cut is added, we must access all q-paths corresponding to the columns in the current LP to calculate the proper coefficients of the row to be inserted. Similarly, each constraint over x variables is stored in a pool of constraints. Every time a variable is priced, we must access each row in the current LP to determine the corresponding constraints and compute the coefficients of the column to be inserted.

The pools must be designed to allow an efficient computation of the inner product of a q-route and an x constraint. Since many such operations are performed each time a new column or row is added, a naïve implementation can significantly slow down the algorithm. Our pools are indexed using auxiliary tables. Given an edge e, we can quickly determine which columns in the LP are associated to q-paths containing e and which rows are associated to constraints having nonzero coefficient in x_e.

Dynamic Selection of Column Generation. For the CVRP, it usually pays off to combine the strengths of cut and column generation into a single algorithm. However, for some instances, the increase in the lower bound is not worth the additional time spent. In these cases pure branch-and-cut performs better. Even when branch-and-cut-and-price is clearly better, one still has to choose a value for s with a good trade-off between column generation time and bound quality.

Our algorithm tries to identify the best option for a given instance before any branching is performed. We start by solving the root node as in a pure branch-and-cut algorithm, without any column generation, recording the running time and the bound achieved. After that, the root node is solved again with branch-and-cut-and-price using column generation with $s = 2$. Then we solve it once again, changing s to 3. For the rest of the branch-and-bound tree, the algorithm picks the strategy with the best balance between running time and bound quality.

More precisely, if the time spent using $s = 2$ is less than 10 times the time spent without column generation and the bound improvement is better than 0.3%, we consider that it is worth doing column generation and move on to decide whether to do it with $s = 2$ or 3, otherwise we perform a pure branch-and-cut. If using $s = 3$ takes less time and gives a better bound than using $s = 2$, we choose $s = 3$. If $s = 3$ is slower by a factor of at most 10, we still choose it as long as the bound improves by at least 0.1%. Otherwise, we choose $s = 2$. To avoid long runs, we interrupt the computation for $s = 2$ or $s = 3$ when it reaches 10 times the time spent by the pure branch-and-cut.

It is worth mentioning that our branch-and-cut-and-price code can be easily converted into a traditional pure branch-and-cut. We simply introduce special λ variables, corresponding not to q-routes, but to each individual edge in E. No pricing is performed. The transformation from x constraints to λ constraints becomes an identity transformation.

Branching Rule. On each node the algorithm starts by adding columns until there are none with negative reduced costs, then it looks for all violated cuts. These passes are repeated until both column generation and separation fail. If the node cannot be fathomed at this point, we branch.

The branching rule is an adaptation of the rule used in [30]. We choose as branching candidates sets S such that $2 < x^*(\delta(S)) < 4$ and branch by imposing the disjunction $(x(\delta(S)) = 2) \vee (x(\delta(S)) \geq 4)$.

We use strong branching to select which set to branch on. Since computing the actual lower bound for each child node can be very time-consuming, we estimate it by performing a small number of column generation iterations. Only p candidate sets $(5 \leq p \leq \max\{10 - \text{depth}, 5\})$ are evaluated. This limits the time spent on strong branching, especially when the depth is high. Priority is given to sets with smaller values of $|x^*(\delta(S)) - 2.7|/d(S)$. We use 2.7 because constraint $x(\delta(S)) = 2$ tends to have a greater impact on the LP relaxation than $x(\delta(S)) \geq 4$, leading to an imbalance in the branch-and-bound tree. Values closer to 2 than 4 increase the impact of imposing $x(\delta(S)) \geq 4$.

Node Selection and Primal Bounds. The node selection strategy chosen was depth first search, because it requires less memory. Since the focus of our study is obtaining good dual bounds, we did not implement primal heuristics and used as initial primal bound the value of the best previously known solution plus one. Good primal heuristics would undoubtedly help solving instances for which good primal bounds are not available.

4. Computational Experiments

We tested our algorithm on all instances from series A, B, E, F, M and P, available at `www.branchandcut.org`. The tests were executed on a Pentium IV running at 2.4 GHz with 1 GB of RAM. Linear programs were solved by CPLEX 7.1. Throughout this section, we refer to the branch-and-cut-and-price algorithm with dynamic selection of column generation as Dyn-BCP and to the branch-and-cut-and-price with predetermined column generation as BCP.

Tables 1 and 2 detail the results obtained by Dyn-BCP. Columns **Root LB** and **Root Time** give the lower bound and total CPU time of the root node of the branch-and-bound tree. This running time includes the dynamic selection of column generation and the strong branching at the root. The next column represents the size s of the cycles eliminated by the column generation procedure ("–" indicates that column generation was not used). **Tree Size** represents the total number of branch-and-bound nodes explored, and **Total Time** is the total CPU time spent by the algorithm. Finally, **Prev UB** and **Our UB** indicate the best solution value available in the literature and the one obtained by our procedure. Values in bold indicate proven optimality. All instances with up to 135 vertices have been solved by our algorithm, eighteen of them for the first time. Based on partial runs, we estimate that the three remaining instances (from series M, with 151 to 200 vertices) could be solved in a few months of CPU time.

Table 3 compares different lower bounds on various instances. Lower bounds L1, L2 and L3 (defined in Section 2) are presented in the first three columns. Note that

Table 1. Results of the Dyn-BCP algorithm for the A and B instances

Instance	Root LB	Root Time (s)	s	Tree Size	Total Time (s)	Prev. UB	Our UB
A-n37-k5	664.8	16	–	8	19	**669**	**669**
A-n37-k6	932.6	30	3	74	379	**949**	**949**
A-n38-k5	716.5	12	–	52	26	**730**	**730**
A-n39-k5	816.6	107	3	11	167	**822**	**822**
A-n39-k6	822.8	39	3	11	98	**831**	**831**
A-n44-k6	934.8	52	2	6	90	**937**	**937**
A-n45-k6	938.1	52	3	11	170	**944**	**944**
A-n45-k7	1139.3	88	3	26	331	**1146**	**1146**
A-n46-k7	914.0	63	2	3	92	**914**	**914**
A-n48-k7	1069.1	72	3	8	166	**1073**	**1073**
A-n53-k7	1003.9	138	3	16	611	**1010**	**1010**
A-n54-k7	1153.9	125	3	90	1409	**1167**	**1167**
A-n55-k9	1067.4	32	3	7	84	**1073**	**1073**
A-n60-k9	1344.4	161	3	224	3080	1354	**1354**
A-n61-k9	1022.5	108	3	121	1883	**1034**	**1034**
A-n62-k8	1280.4	722	3	101	3102	1290	**1288**
A-n63-k9	1607.0	238	3	49	1046	1616	**1616**
A-n63-k10	1299.1	136	3	387	4988	1315	**1314**
A-n64-k9	1385.3	265	3	648	11254	1402	**1401**
A-n65-k9	1166.5	154	3	17	516	**1174**	**1174**
A-n69-k9	1141.4	289	3	391	7171	1159	**1159**
A-n80-k10	1754.0	1120	3	153	6464	1763	**1763**
B-n38-k6	800.2	10	–	14	14	**805**	**805**
B-n39-k5	549.0	3	–	1	3	**549**	**549**
B-n41-k6	826.4	13	–	8	18	**829**	**829**
B-n43-k6	733.7	13	–	74	29	**742**	**742**
B-n44-k7	909.0	9	–	1	9	**909**	**909**
B-n45-k5	747.5	10	–	19	16	**751**	**751**
B-n45-k6	677.5	224	3	3	279	**678**	**678**
B-n50-k7	741.0	5	–	3	6	**741**	**741**
B-n50-k8	1295.0	97	3	287	2845	**1312**	**1312**
B-n51-k7	1025.2	16	–	83	46	**1032**	**1032**
B-n52-k7	745.8	7	–	9	9	**747**	**747**
B-n56-k7	704.0	15	–	9	22	**707**	**707**
B-n57-k7	1149.2	76	–	133	168	**1153**	**1153**
B-n57-k9	1596.0	61	3	15	193	**1598**	**1598**
B-n63-k10	1479.4	231	–	501	682	**1496**	**1496**
B-n64-k9	859.3	70	–	7	86	**861**	**861**
B-n66-k9	1307.5	145	3	126	1778	**1316**	**1316**
B-n67-k10	1024.4	218	–	327	568	**1032**	**1032**
B-n68-k9	1263.0	260	3	5912	87436	1275*	**1272**
B-n78-k10	1215.2	193	3	90	1053	**1221**	**1221**

* An earlier version of our algorithm [19] found a solution of 1272 but could not prove its optimality. Using that information, K. Wenger [43] solved this instance.

calculating L1 and L3 requires the expensive exact separation of rounded capacity cuts, as described in Subsection 3.2. The instances in the table are the 41 that have at least 50 vertices and for which the L1 and L3 bounds could be computed within a reasonable amount of time. Column **LLE04** shows the lower bounds obtained in [30], which are the best available in the literature. The next three columns contain the lower bounds obtained at the root node of BCP with s-cycle elimination (for $s = 2, 3, 4$). As BCP only uses heuristic separation of rounded capacity cuts, the bound obtained for $s = 2$ is a little worse than L3 on a few instances. However, this bound is usually better than

Table 2. Results of the Dyn-BCP algorithm for the E, F, M and P instances

Instance	Root LB	Root Time (s)	s	Tree Size	Total Time (s)	Prev. UB	Our UB
E-n13-k4	247.0	0	–	1	0	**247**	**247**
E-n22-k4	375.0	0	–	1	0	**375**	**375**
E-n23-k3	569.0	0	–	1	0	**569**	**569**
E-n30-k3	533.3	7	–	6	7	**534**	**534**
E-n31-k7	378.5	4	2	2	6	**379**	**379**
E-n33-k4	834.5	14	–	5	15	**835**	**835**
E-n51-k5	518.2	51	–	8	65	**521**	**521**
E-n76-k7	670.0	264	2	1712	46520	**682**	**682**
E-n76-k8	726.5	277	2	1031	22891	**735**	**735**
E-n76-k10	817.4	354	3	4292	80722	830	**830**
E-n76-k14	1006.5	224	3	6678	48637	1021	**1021**
E-n101-k8	805.2	1068	3	11622	801963	**815**	**815**
E-n101-k14	1053.8	658	3	5848	116284	1071	**1067**
F-n45-k4	724.0	8	–	1	8	**724**	**724**
F-n72-k4	236.4	70	–	42	121	**237**	**237**
F-n135-k7	1159.9	6618	–	25	7065	**1162**	**1162**
M-n101-k10	820.0	119	–	1	119	**820**	**820**
M-n121-k7	1031.1	5594	3	40	25678	1034	**1034**
M-n151-k12	999.1	945	3	–	–	1015	–
M-n200-k16	1252.4	3168	3	–	–	–	–
M-n200-k17	1254.2	2310	3	–	–	1275	–
P-n16-k8	449.0	1	2	3	1	**450**	**450**
P-n19-k2	212.0	1	–	1	1	**212**	**212**
P-n20-k2	213.0	1	–	9	1	**216**	**216**
P-n21-k2	211.0	1	–	1	1	**211**	**211**
P-n22-k2	216.0	2	–	2	2	**216**	**216**
P-n22-k8	603.0	3	2	1	3	**603**	**603**
P-n23-k8	529.0	18	2	1	18	**529**	**529**
P-n40-k5	456.9	28	–	5	34	**458**	**458**
P-n45-k5	506.6	59	3	11	194	**510**	**510**
P-n50-k7	551.5	79	3	7	143	**554**	**554**
P-n50-k8	616.3	102	3	1084	9272	649	**631**
P-n50-k10	689.3	50	3	65	304	**696**	**696**
P-n51-k10	735.2	35	3	22	105	**741**	**741**
P-n55-k7	557.9	90	2	450	4649	**568**	**568**
P-n55-k8	579.8	42	2	196	1822	**588**	**588**
P-n55-k10	681.4	107	3	1556	9076	699	**694**
P-n55-k15	972.8	251	3	398	1944	993	**989**
P-n60-k10	738.9	126	3	52	570	756	**744**
P-n60-k15	962.8	118	3	76	442	1033	**968**
P-n65-k10	786.0	159	3	23	422	792	**792**
P-n70-k10	814.5	292	3	1752	24039	834	**827**
P-n76-k4	588.8	363	–	59	572	**593**	**593**
P-n76-k5	616.8	273	–	3399	14546	**627**	**627**
P-n101-k4	678.5	1055	–	23	1253	**681**	**681**

L3 due to the presence of other families of cuts. As expected, the bounds for $s = 3$ and $s = 4$ are even better. Column **OPT** contains optimal solution values. The last line in the table shows the average gap obtained by each bound with respect to the optimal solutions (considering only the instances in the table). The experimental results show that bound L_2 is quite poor, worse than L_1 on most instances. Bound L_3 is much better than L_1 and L_2. This indicates that the most important ingredient of our algorithm is not column generation or cut generation, but the combination of both approaches.

Table 3. Comparison of lower bounds

Instance	Lower Bounds							OPT
	L1	L2	L3	LLE04	$s = 2$	$s = 3$	$s = 4$	
A-n53-k7	996.6	978.5	1002.2	998.7	1002.2	1003.9	1004.1	**1010**
A-n54-k7	1130.7	1114.0	1150.0	1135.3	1150.3	1153.2	1155.5	**1167**
A-n55-k9	1055.9	1025.4	1066.4	1058.3	1066.4	1067.2	1067.6	**1073**
A-n60-k9	1316.5	1305.6	1341.6	1319.6	1341.6	1344.4	1344.9	**1354**
A-n61-k9	1004.8	996.8	1018.6	1010.2	1018.8	1022.4	1023.3	**1034**
A-n62-k8	1244.1	1222.7	1274.1	1251.7	1273.2	1280.1	1280.9	**1288**
A-n63-k9	1572.2	1564.8	1603.5	1580.7	1603.5	1606.4	1608.6	**1616**
A-n63-k10	1262.2	1267.4	1294.5	1266.6	1294.2	1299.1	1302.8	**1314**
A-n64-k9	1340.1	1353.3	1378.9	1351.6	1378.8	1385.2	1389.1	**1401**
A-n65-k9	1151.1	1133.0	1163.4	1155.2	1164.5	1166.6	1168.6	**1174**
A-n69-k9	1108.9	1113.2	1138.4	1114.4	1138.7	1140.8	1143.8	**1159**
A-n80-k10	1699.9	1712.2	1749.7	1709.6	1750.1	1753.8	1755.4	**1763**
B-n50-k7	740.0	664.8	741.0	741.0	741.0	741.0	741.0	**741**
B-n50-k8	1279.2	1217.5	1291.8	1281.1	1291.8	1295.1	1302.0	**1312**
B-n51-k7	1024.6	918.4	1025.9	1025.6	1025.9	1026.4	1026.6	**1032**
B-n52-k7	745.0	640.2	746.3	746.0	746.4	746.4	746.4	**747**
B-n56-k7	703.4	606.9	704.5	705.0	704.5	705.0	705.0	**707**
B-n57-k7	1148.6	1058.5	1150.9	1150.1	1150.9	1151.6	1152.3	**1153**
B-n57-k9	1586.7	1511.5	1595.2	1589.2	1595.2	1596.0	1596.0	**1598**
B-n63-k10	1478.9	1418.4	1484.2	1481.0	1484.2	1486.7	1487.2	**1496**
B-n64-k9	858.5	769.3	860.1	860.5	860.2	860.4	860.5	**861**
B-n66-k9	1295.2	1223.1	1302.6	1298.5	1303.6	1307.3	1308.4	**1316**
B-n67-k10	1023.8	984.5	1026.4	1024.8	1026.4	1026.8	1027.2	**1032**
B-n68-k9	1256.8	1163.9	1261.5	1258.1	1261.6	1263.0	1263.5	**1272**
B-n78-k10	1202.3	1124.5	1212.5	1205.6	1212.6	1215.0	1215.9	**1221**
E-n51-k5	514.5	512.9	518.0	519.0	519.1	519.1	519.0	**521**
E-n76-k7	661.4	663.3	668.4	666.4	669.9	670.4	670.5	**682**
E-n76-k8	711.2	716.7	725.1	717.9	726.0	726.4	726.8	**735**
E-n76-k10	789.5	811.4	816.5	799.9	816.8	816.8	817.3	**830**
E-n76-k14	948.1	999.6	1004.8	969.6	1004.8	1006.5	1007.4	**1021**
E-n101-k8	796.4	786.4	801.8	802.6	803.7	804.1	804.3	**815**
E-n101-k14	1008.3	1045.1	1051.6	1026.9	1051.6	1053.5	1053.8	**1067**
M-n101-k10	819.5	798.1	820.0	820.0	820.0	820.0	820.0	**820**
M-n121-k7	1009.7	1013.0	1030.9	1017.4	1031.2	1031.8	1032.0	**1034**
P-n50-k8	596.9	612.5	615.3	602.1	615.7	616.2	616.9	**631**
P-n55-k10	646.7	677.2	680.1	662.1	680.0	681.3	681.3	**694**
P-n55-k15	895.1	963.0	967.5	906.7	967.5	972.8	972.8	**989**
P-n60-k10	708.3	733.5	737.2	718.4	737.2	738.7	739.0	**744**
P-n60-k15	903.3	955.6	961.2	929.8	961.2	962.8	963.5	**968**
P-n65-k10	756.5	779.8	785.2	767.1	785.2	785.9	786.9	**792**
P-n70-k10	786.9	808.3	812.7	795.6	813.4	814.3	814.8	**827**
AvgGAP	**2.92%**	**4.76%**	**1.02%**	**2.26%**	**0.97%**	**0.82%**	**0.74%**	—

Table 4 presents aggregate results for the root node using four different versions of BCP: a pure branch-and-cut (indicated by BC in the table) and branch-and-cut-and-price with elimination of 2-, 3-, and 4-cycles. We consider in our calculations only the 36 instances with at least 50 vertices for which Dyn-BCP chose to use column generation and found the optimal solution. To avoid biasing the results towards larger instances, all values presented are geometric means.

For each variant, we show the mean time to solve the root node, as well as the mean time spent in the three most important phases of the algorithm: column generation, separation, and LP solving. For reference, we show the number of columns and rows added

Table 4. Root node statistics for different variants of BCP. Values are geometric means over all instances with at least 50 vertices that were solved to optimality using branch-and-cut-and-price

	BC	$s = 2$	$s = 3$	$s = 4$
Column generation time (s)	0.0	4.0	5.3	11.3
Separation time (s)	4.7	2.1	2.0	1.8
LP solving time (s)	6.6	4.7	3.9	3.7
Total time (s)	12.1	12.7	12.9	18.3
Columns added	0	4338	4074	4087
Rows added	1053	156	137	129
Gap (%)	3.19	1.08	0.89	0.79
CG iterations	0.0	141.1	133.7	131.6
Passes	37.2	8.5	8.0	7.7

by column generation and separation, respectively (the original columns and rows are not considered). We also show the mean number of iterations of column generation, as well as the number of passes performed by the algorithm. For BCP, each pass is defined as the application of column generation—until it converges—followed by a call to the separation routines; for the pure branch-and-cut, a pass is defined as a call to the separation routines. Finally, we present the gap between the lower bound found by the algorithm and the optimal solution.

The table makes it clear that, on instances where Dyn-BCP chooses to perform branch-and-cut-and-price, even the simplest form of column generation (which eliminates 2-cycles only) reduces the gap by a factor of almost three when compared to the pure branch-and-cut, while leaving the mean time to solve the root node almost unchanged. Eliminating 3-cycles further reduces the gap, with a minimal increase in running time. Eliminating 4-cycles, on the other hand, significantly increases the total running time, while decreasing the mean gap by a smaller factor. That is why Dyn-BCP only chooses between $s = 2$ and $s = 3$ (or no column generation at all). Still, a few instances (like B-n50-k8) are better solved with $s = 4$.

Table 5 presents the same pieces of information as Table 3, but for ten individual instances selected so as to illustrate different aspects of the behavior of BCP. These results show that only considering means can be misleading, since the performance of a BCP (when compared to a pure branch-and-cut) is heavily dependent on the instance. In most cases, it does improve the lower bound dramatically with only a minor increase (and sometimes even a reduction) in the running time. This behavior can be seen on A-n63-k10, A-n80-k10, B-n50-k8, E-n76-k14, and E-n101-k14. Other instances, such as M-n121-k7, B-n68-k9, and E-n101-k8, also see a remarkable increase in the lower bound quality, but with significantly higher running times—BCP is still a good choice, but not clearly superior to BC. A third class is formed by instances for which the BCP not only has a marginal effect on the bounds, but also takes an unacceptably long time to run (due to column generation convergence problems). This happens with P-n101-k4 and F-n72-k4; for the latter, we could not compute the bounds for $s = 3$ or $s = 4$, even allowing several hours of computation. Fortunately, those instances are precisely the ones that are solved very well by a pure BC. This fact makes Dyn-BCP a very consistent algorithm.

Table 5. Detailed root node statistics for a set of representative instances

		Operations				Time (s)				Gap
Instance	s	Columns	Rows	Iter.	Passes	CG	Sep.	LP	Total	(%)
A-n63-k10	–	0	1137	0	41	0.0	3.4	7.8	11.3	3.95
	2	4776	169	157	9	4.5	2.2	4.9	11.8	1.51
	3	4576	170	138	7	5.1	1.6	4.3	11.0	1.14
	4	4283	117	125	6	9.4	1.0	2.6	13.0	0.85
A-n80-k10	–	0	1566	0	42	0.0	7.5	24.0	31.8	3.57
	2	7716	229	206	8	12.4	1.8	16.1	30.6	0.74
	3	9240	228	219	10	18.4	3.3	21.1	43.0	0.52
	4	8707	172	208	9	39.5	3.4	17.1	60.2	0.43
B-n50-k8	–	0	1220	0	38	0.0	1.6	4.8	6.5	2.50
	2	3625	223	163	9	3.1	1.6	5.3	10.1	1.54
	3	3450	187	158	9	4.5	2.8	4.1	11.5	1.30
	4	3189	168	143	7	8.8	1.6	3.2	13.7	0.76
B-n68-k9	–	0	1103	0	60	0.0	3.5	6.2	9.8	1.19
	2	6478	212	206	8	8.5	1.9	13.8	24.4	0.82
	3	5716	227	189	8	9.9	1.2	11.3	22.6	0.71
	4	5389	208	203	11	26.0	2.0	9.4	37.6	0.67
E-n76-k14	–	0	1366	0	38	0.0	9.8	16.5	26.5	7.13
	2	3665	91	82	5	2.3	2.6	1.3	6.2	1.59
	3	3872	66	84	6	3.1	2.8	1.3	7.2	1.42
	4	3517	75	79	7	6.3	4.6	1.2	12.1	1.33
E-n101-k8	–	0	1119	0	48	0.0	25.7	22.0	48.3	1.93
	2	15349	335	356	16	41.3	10.1	105.5	158.5	1.38
	3	16838	290	357	16	58.9	9.5	107.5	177.3	1.34
	4	14348	296	330	14	99.0	6.6	90.3	197.1	1.31
E-n101-k14	–	0	2576	0	82	0.0	19.1	60.8	80.6	5.38
	2	7843	150	129	9	9.9	3.1	7.3	20.5	1.44
	3	6433	130	119	9	11.0	6.2	5.6	23.0	1.26
	4	7262	157	130	8	22.6	2.5	7.5	32.8	1.23
F-n72-k4	–	0	313	0	66	0.0	2.3	1.7	4.1	0.34
	2	31415	261	2178	18	4546.2	3.5	630.9	5187.8	0.74
M-n121-k7	–	0	2916	0	76	0.0	21.5	142.8	166.0	2.32
	2	39253	355	1244	13	334.7	3.6	625.1	970.9	0.30
	3	40729	344	1128	12	368.1	3.7	550.5	927.8	0.26
	4	42830	216	1146	9	715.9	3.2	578.0	1302.1	0.21
P-n101-k4	–	0	648	0	42	0.0	7.2	8.3	16.1	0.39
	2	227271	472	14272	28	5954.5	16.9	21543.4	27929.6	0.37
	3	379023	505	20599	37	15991.0	19.0	63765.2	80432.0	0.39
	4	228294	460	11317	21	8446.0	11.5	31751.2	40540.0	0.37

It can be noted that branch-and-cut-and-price usually achieves the greatest improvements over pure branch-and-cut when the ratio n/K, the average number of clients per vehicle, is smallest. In such cases, q-routes without small cycles are very close to being actual routes, so one can expect the duality gap to be reduced. Figure 1, created from the instances of Table 3, illustrates this observation. The horizontal axis represents the ratio n/K. The value plotted is $(s3 - lle)/(opt - lle)$, where $s3$ is the bound of the BCP with $s = 3$, lle is the LLE04 bound and opt is the optimal solution value. This ratio represents the reduction of the duality gap relative to the LLE04 gap. The value is zero whenever both methods produce the same result and one when the BCP closes the gap in full. Circles represent instances that were solved by our algorithm for the first time, while crosses indicate instances that had already been solved.

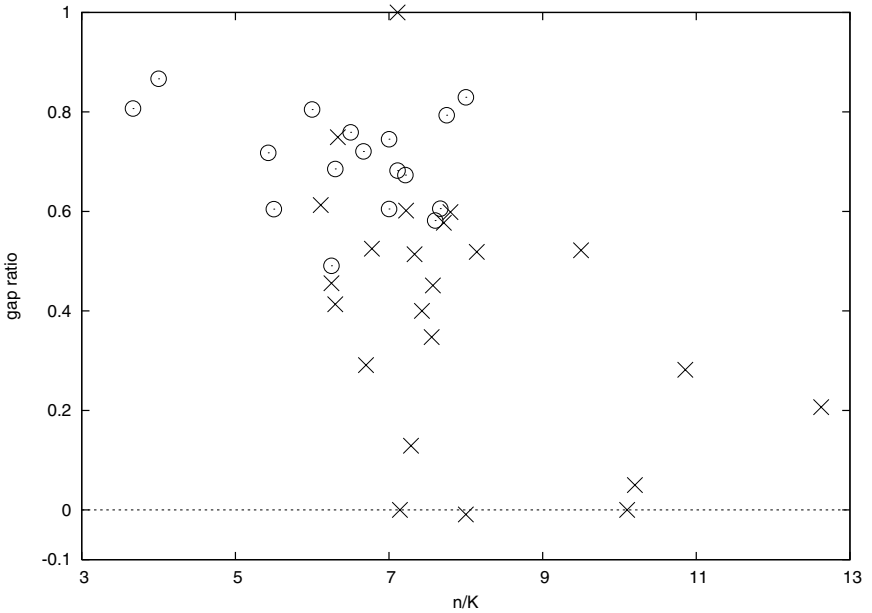

Fig. 1. Gap improvement of BCP with $s = 3$ ($s3$) with respect to [30] (lle). The graph represents $\frac{s3-lle}{opt-lle}$ as a function of $\frac{n}{K}$. Circles represent instances solved to optimality for the first time and crosses the ones already closed

5. Conclusion

Figures 2, 3, and 4 display the optimal solutions of instances E-n76-k10, E-n76-k14 and E-n101-k14. Each client is represented as a box containing its demand. We consider the solution of these instances for the first time a significant achievement. The E instances have been the main benchmark of CVRP algorithms since [11], appearing in dozens of published articles. Until this work, those three instances were considered very far from being solved, as attested by the comments in the conclusion of [30]:

> 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.

This paper has shown that combining known cuts with column generation is a more practical way of improving CVRP algorithms than searching for increasingly complicated families of cuts.

This does not mean that our BCP algorithm would not benefit from further polyhedral research on the CVRP. Currently, rounded capacity cuts are highly effective—their efficient heuristic separation contributes significantly to the good overall performance. We did try all other families of cuts for the CVRP known to be effective in a pure BC: framed capacities, generalized capacities, strengthened combs, multistars, and extended

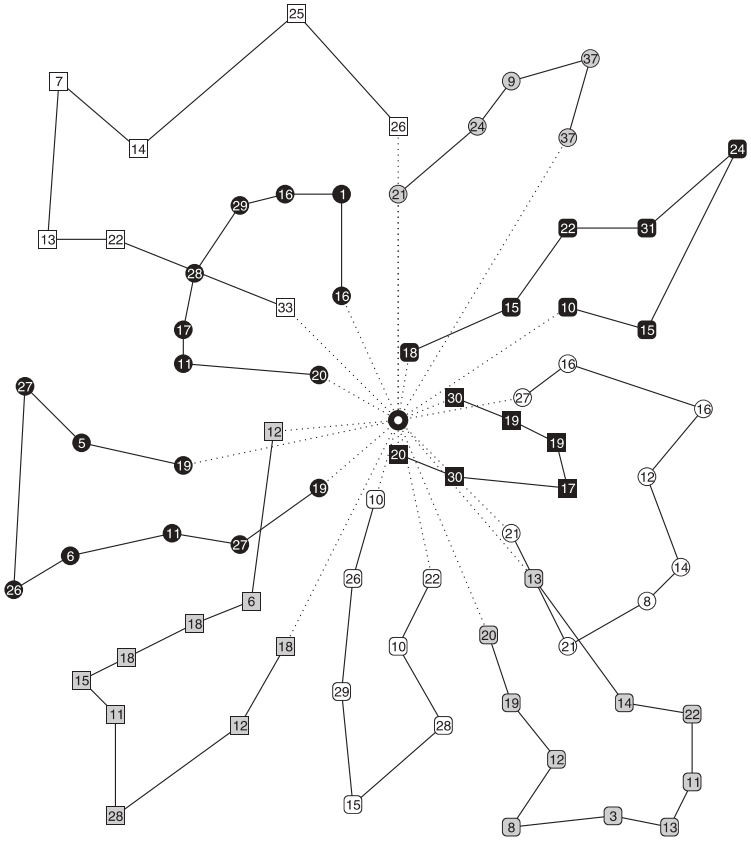

Fig. 2. Optimal solution to E-n76-k10 (vehicle capacity = 140)

hypotours. Surprisingly, however, their effect was quite modest in practice—only on E-n101-k8 the root bound increased by more than two units. Therefore, any improvement to the cutting part of our BCP will indeed require new families of inequalities. It is interesting to note that even new facet-defining inequalities for the CVRP polyhedron are not guaranteed to be helpful within the BCP framework. It is possible that such inequalities also define facets of P_2, the polyhedron induced by the column generation over the q-routes. In this case, since $P_3 \subseteq P_2$, those inequalities would never cut fractional solutions. For example, Letchford and Salazar [27] have recently proved that generalized large multistar inequalities would be useless in our BCP, since they do not cut P_2.

In the short term, improvements to the BCP are more likely to come from column generation. The goal is to devise efficient ways of pricing over a set of columns as close to cycle-free q-routes as possible. The s-cycle elimination approach we chose was reasonably effective, at least for $s \leq 4$. But other alternatives to further restrict the q-routes can also be tried. One could, for instance, forbid q-routes that visit vertices in a small set S more than once. This set could be dynamically chosen in order to eliminate q-routes with long cycles that are particularly harmful to bound quality.

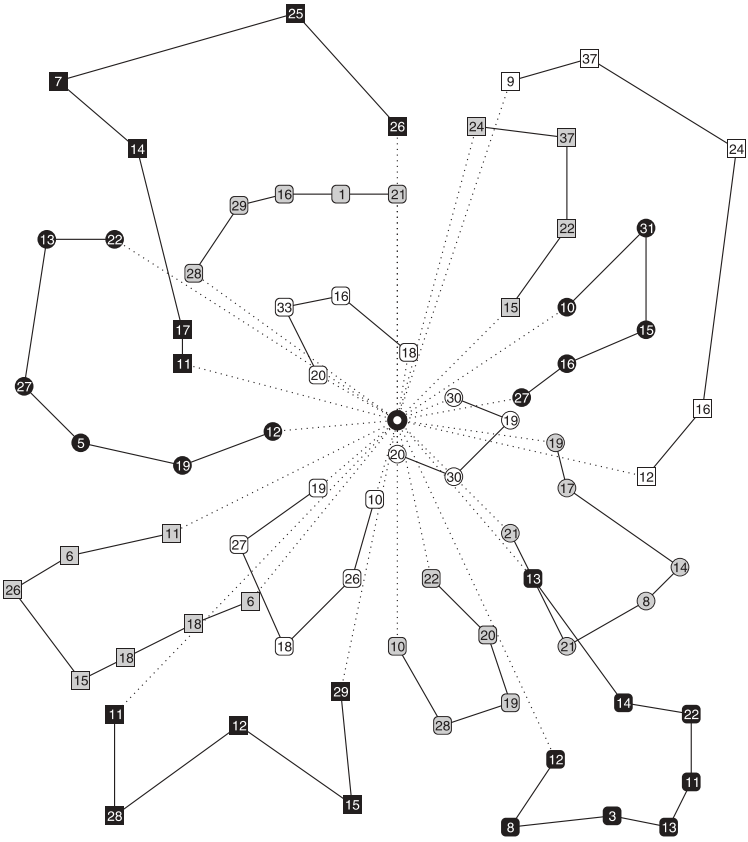

Fig. 3. Optimal solution to E-n76-k14 (vehicle capacity = 100)

In principle, several of the combinatorial relaxations of the CVRP—such as k-degree center trees, K-trees, or b-matchings—could also be used instead of (or even in addition to) q-routes. We think they are unlikely to have a noticeable impact, for two reasons. First, among the relaxations mentioned above, only q-routes take client demands and vehicle capacities into account. These numerical elements are precisely what makes the CVRP much harder in practice than a pure graph problem such as the TSP. It seems that known families of inequalities for the CVRP, mostly inspired by previous work on the TSP, cannot cope well with this numerical aspect. The inequalities implicitly given by P_2 can do better. The second reason for not using the alternatives above is that only q-routes lead to a pricing problem that is superpolynomial, but still practical. The polyhedra associated with polynomial pricing problems can be fully described and efficiently separated, which usually makes a pure BC (instead of BCP) a faster alternative. For instance, we can separate over the b-matching polyhedron in a very efficient way [34, 26]. Separation over the q-route polyhedron, on the other hand, is possible only implicitly, with column generation.

Even if we leave column and cut generation as they are, we can still accelerate the algorithm through better integration. The task of designing, coding and fine-tuning a

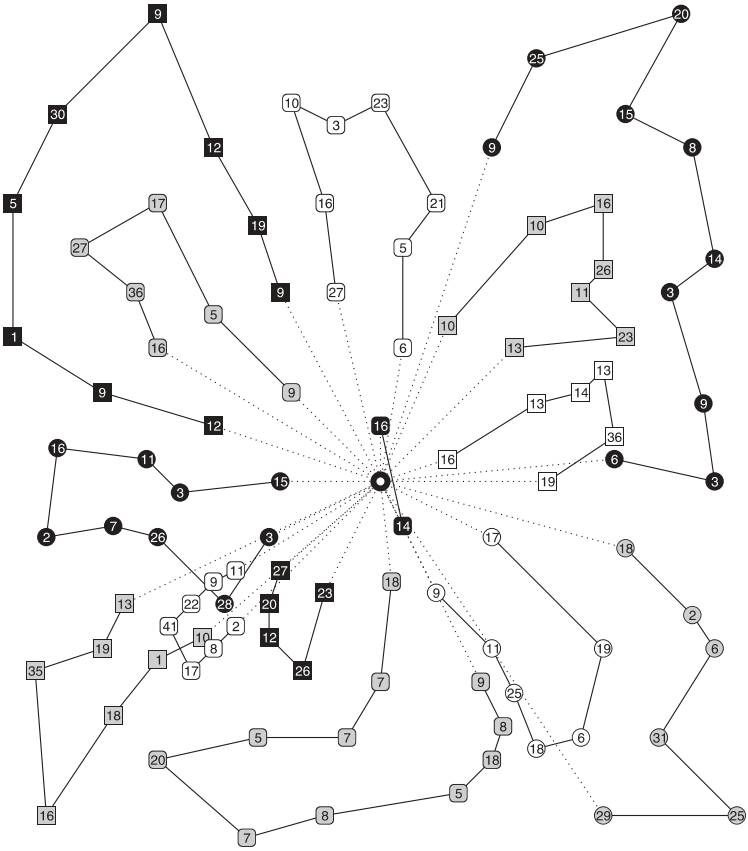

Fig. 4. Optimal solution to E-n101-k14 (vehicle capacity = 112)

high-performance branch-and-cut or branch-and-price procedure is complex, requiring experience to choose among several possible strategies, and, often, some computational tricks. BCP multiplies the possibilities and, therefore, the difficulties. We believe that there is still plenty of room for such improvements in our BCP.

The general approach described in this article may be a good alternative for several VRP variants other than the CVRP. This has already been shown to be true for the capacitated arc routing problem [28]. In some cases, when the variant can be viewed as a CVRP with additional constraints (such as maximum route length or time windows), one only needs to include the new constraints into the dynamic programming pricing routine in order to have a working code. However, to get the full benefits from the BCP mechanism, one should also change the separation procedures in order to find the stronger cuts obtained when the additional constraints are taken into account.

Acknowledgements. RF was supported by NSF grant DMI-0245609, having started this work while at Gapso Inc., Brazil. HL was was supported by CAPES. MPA was funded by CNPq grant 300475/93-4. MR was supported by CAPES. EU received support from Mestrado em Engenharia de Produção-UFF and CNPq grant 304533/02-5. RW was supported by the Aladdin Project, NSF grant CCR-9626862

References

1. Achuthan, N., Caccetta, L., Hill, S.: Capacited vehicle routing problem: Some new cutting planes. Asia-Pacific J. Oper. Res. **15**, 109–123 (1998)
2. Achuthan, N., Caccetta, L., Hill, S.: An improved branch-and-cut algorithm for the capacitated vehicle routing problem. Transportation Sci. **37**, 153–169 (2003)
3. Agarwal, Y., Mathur, K., Salkin, H.: A set-partitioning based exact algorithm for the vehicle routing problem. Networks **19**, 731–739 (1989)
4. Araque, J., Hall, L., Magnanti, T.: Capacitated trees, capacitated routing and polyhedra. Technical Report SOR-90-12. Princeton University, 1990
5. Araque, J., Kudva, G., Morin, T., Pekny, J.: 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 véhicles. PhD thesis, Institut National Polytechnique de Grenoble, 1995
7. Augerat, P., Belenguer, J., Benavent, E., Corberán, A., Naddef, D., Rinaldi, G.: Computational results with a branch and cut code for the capacitated vehicle routing problem. Technical Report 949-M, Université Joseph Fourier, Grenoble, France, 1995
8. Balinski, M., Quandt, R.: On an integer program for a delivery problem. Oper. Res. **12**, 300–304 (1964)
9. Barnhart, C., Hane, C., Vance, P.: Using branch-and-price-and-cut to solve origin-destination integer multicommodity flow problems. Oper. Res. **40**, 318–326 (2000)
10. Blasum, U., Hochstättler, W.: Application of the branch and cut method to the vehicle routing problem. Technical Report ZPR2000-386, Zentrum fur Angewandte Informatik Köln, 2000
11. Christofides, N., Eilon, S.: An algorithm for the vehicle-dispatching problem. Oper. Res. Q. **20**, 309–318 (1969)
12. Christofides, N., Mingozzi, A., Toth, P.: Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations. Math. Prog. **20**, 255–282 (1981)
13. Cornuéjols, G., Harche, F.: Polyhedral study of the capacitated vehicle routing problem. Math. Prog. **60**, 21–52 (1993)
14. Dantzig, G., Ramser, R.: The truck dispatching problem. Management Science **6**, 80–91 (1959)
15. Desrochers, M., Desrosiers, J., Solomon, M.: A new optimization algorithm for the vehicle routing problem with time windows. Oper. Res. **40**, 342–354 (1992)
16. Felici, G., Gentile, C., Rinaldi, G.: Solving large MIP models in supply chain management by branch & cut. Technical report, Istituto di Analisi dei Sistemi ed Informatica del CNR, Italy, 2000
17. Fisher, M.: Optimal solution of vehicle routing problem using minimum k-trees. Oper. Res. **42**, 626–642 (1994)
18. Fukasawa, R., Poggi de Aragão, M., Porto, O., Uchoa, E.: Robust branch-and-cut-and-price for the capacitated minimum spanning tree problem. In: Proc. of the International Network Optimization Conference. Evry, France, 2003, pp. 231–236
19. Fukasawa, R., Reis, M., Poggi de Aragão, M., Uchoa, E.: Robust branch-and-cut-and-price for the capacitated vehicle routing problem. Technical Report RPEP Vol.3 no.8, Universidade Federal Fluminense, Engenharia de Produção, Niterói, Brazil, 2003
20. Hadjiconstantinou, E., Christofides, N., Mingozzi, A.: A new exact algorithm from the vehicle routing problem based on q-paths and k-shortest paths relaxations. In: Laporte, G., Michel Gendreau (eds.), Freight Transportation, **61** in Annals of Operations Research. Baltzer Science Publishers, 1995, pp. 21–44
21. Irnich, S., Villeneuve, D.: The shortest path problem with resource constraints and k-cycle elimination for $k \geq 3$, 2003. To appear in JNFORMS Journal on computing. Available at `http://www.dpor.rwth-aachen.de/de/publikationen/pdf/or_2003-01.pdf`.
22. Kim, D., Barnhart, C., Ware, K., Reinhardt, G.: Multimodal express package delivery: A service network design application. Transportation Science **33**, 391–407 (1999)
23. Kohl, N., Desrosiers, J., Madsen, O., Solomon, M., Soumis, F.: 2-Path cuts for the vehicle routing problem with time windows. Transportation Science **33**, 101–116 (1999)
24. Laporte, G., Norbert, Y.: A branch and bound algorithm for the capacitated vehicle routing problem. Oper. Res. Spektrum **5**, 77–85 (1983)
25. Letchford, A., Eglese, R., Lysgaard, J.: Multistars, partial multistars and the capacitated vehicle routing problem. Math. Prog. **94**, 21–40 (2002)
26. Letchford, A., Reinelt, G., Theis, D.: A faster exact separation algorithm for blossom inequalities. In: Proceedings of the X IPCO, volume 3064 of Lecture Notes in Computer Science. New York, 2004, pp. 196–205
27. Letchford, A.N., Salazar, J.J.: Projection results for vehicle routing. Math. Prog. 2004, To appear
28. Longo, H., Poggi de Aragão, M., Uchoa, E.: Solving capacitated arc routing problems using a transformation to the CVRP. Computers & Operations Research, 2004, To appear

29. Lysgaard, J.: CVRPSEP: A package of separation routines for the capacitated vehicle routing problem, 2003. Available at `www.asb.dk/~lys`
30. Lysgaard, J., Letchford, A., Eglese, R.: A new branch-and-cut algorithm for the capacitated vehicle routing problem. Math. Prog. **100** (2), 423–445 (2004)
31. Martinhon, C., Lucena, A., Maculan, N.: Stronger k-tree relaxations for the vehicle routing problem. European J. Oper. Res. **158** (1), 56–71 (2004)
32. Miller, D.: A matching based exact algorithm for capacitated vehicle routing problems. ORSA J. Comput. **7**, 1–9 (1995)
33. Naddef, D., Rinaldi, G.: Branch-and-cut algorithms for the capacitated VRP. In: Toth, P., Vigo, D. (eds.) The Vehicle Routing Problem, chapter 3. SIAM, 2002, pp. 53–84
34. Padberg, M., Rao, M.: Odd minimum cut-sets and b-matchings. Math. Oper. Res. **7** (1), 67–80 (1982)
35. Pigatti, A.: Modelos e algoritmos para o problema de alocação generalizada e aplicações. Master's thesis, Pontifí cia Universidade Católica do Rio de Janeiro, Brazil, July 2003
36. Poggi de Aragão, M., Uchoa, E.: Integer program reformulation for robust branch-and-cut-and-price. In: Annals of Mathematical Programming in Rio. Búzios, Brazil, 2003, pp. 56–61
37. Ralphs, T.: Parallel branch and cut for capacitated vehicle routing. Parallel Computing **29**, 607–629 (2003)
38. Ralphs, T., Kopman, L., Pulleyblank, W., Trotter, L. Jr.: On the capacitated vehicle routing problem. Math. Prog. **94**, 343–359 (2003)
39. Toth, P., Vigo, D.: Models, relaxations and exact approaches for the capacitated vehicle routing problem. Discrete Applied Mathematics **123**, 487–512 (2002)
40. Toth, P., Vigo, D.:The Vehicle Routing Problem. Monographs on Discrete Mathematics and Applications. SIAM, 2002
41. Van den Akker, J., Hurkens, C., Savelsbergh, M.: Time-indexed formulation for machine scheduling problems: column generation. INFORMS J. on Computing **12**, 111–124 (2000)
42. Vanderbeck, F.: Lot-sizing with start-up times. Management Science **44**, 1409–1425 (1998)
43. Wenger, K.M.: Generic Cut Generation Methods for Routing Problems. PhD thesis, Institute of Computer Science, University of Heidelberg, 2003
44. Werneck, R.F., Setubal, J.C.: Finding minimum congestion spanning trees. ACM J. of Experimental Algorithmics, **5**, 2000