Math. Program., Ser. A (2008) 112:443–472
DOI 10.1007/s10107-006-0043-y

FULL LENGTH PAPER

Robust branch-cut-and-price for the Capacitated Minimum Spanning Tree problem over a large extended formulation

Eduardo Uchoa · Ricardo Fukasawa · Jens Lysgaard · Artur Pessoa · Marcus Poggi de Aragão · Diogo Andrade

Received: 25 May 2006 / Accepted: 17 August 2006 / Published online: 5 October 2006
© Springer-Verlag 2006

Abstract This paper presents a robust branch-cut-and-price algorithm for the Capacitated Minimum Spanning Tree Problem (CMST). The variables are associated to q-arbs, a structure that arises from a relaxation of the capacitated prize-collecting arborescence problem in order to make it solvable in pseudo-polynomial time. Traditional inequalities over the arc formulation, like Capacity Cuts, are also used. Moreover, a novel feature is introduced in such kind of algorithms: powerful new cuts expressed over a very large set of variables are added, without increasing the complexity of the pricing subproblem or the size of the

E. Uchoa · A. Pessoa
Departamento de Engenharia de Produção, Universidade Federal Fluminense, Niterói, Brazil
e-mail: uchoa@producao.uff.br

A. Pessoa
e-mail: artur@producao.uff.br

R. Fukasawa (✉)
H. Milton Stewart School of Industrial and Systems Engineering, GeorgiaTech, Atlanta, GA, USA
e-mail: rfukasaw@isye.gatech.edu

J. Lysgaard
Department of Business Studies, Aarhus School of Business, Aarhus, Denmark
e-mail: lys@asb.dk

M. Poggi de Aragão
Departamento de Informática, PUC Rio de Janeiro, Rio de Janerio, Brazil
e-mail: poggi@inf.puc-rio.br

D. Andrade
RUTCOR, Rutgers University, Piscataway, NJ, USA
e-mail: dandrade@rutcor.rutgers.edu

LPs that are actually solved. Computational results on benchmark instances from the OR-Library show very significant improvements over previous algorithms. Several open instances could be solved to optimality.

Mathematics Subject Classification (1991) 20E28 · 20G40 · 20C20

1 Introduction

Let $G = (V, E)$ be an undirected graph with vertices $V = \{0, 1, \ldots, n\}$ and $m = |E|$ edges. Vertex 0 is the *root*. Each remaining vertex i is associated with a positive integer demand d_i. Root demand d_0 is defined as zero. Each edge $e \in E$ has a nonnegative cost c_e. Given a positive integer C greater than or equal to the maximum demand, the *Capacitated Minimum Spanning Tree* (CMST) problem consists of finding a minimum cost spanning tree for G such that the total demand of the vertices in each subtree hanging from the root does not exceed C. Figure 1 shows an example of a CMST solution. This NP-hard problem has important applications in network design, but it also receives attention from the optimization community for being a nice example of an "easy to state, hard to solve" problem. In fact, while minimum cost spanning trees can be obtained in almost linear time by greedy algorithms [15], some quite small CMST instances can not be solved by sophisticated branch-and-cut codes. Part of the literature on the CMST only address the *unitary demands* (UD) case, where all $d_i = 1$. In this article, we are considering general demands.

Several exact algorithms have been proposed for the CMST, including [9,16,20–22,24,25,32,40]. On UD instances, the best results were obtained by Gouveia and Martins using branch-and-cut algorithms over the hop-indexed formulations [20–22]. Such formulations yield quite good lower bounds, but lead to very large LPs having $O(Cm)$ variables and constraints. They are only practical for small values of C. The best results on non-UD instances were obtained by branch-and-cut over the arc formulation [25] or by Lagrangean relaxation enhanced by cuts [9]. Such algorithms rely on cuts found by polyhedral investigation [4,18,19,26]. However, the lower bounds so obtained are often not very tight. Benchmark instances from the OR-Library with 50 vertices and $C = 200$ can not be solved by any such method. Several heuristic algorithms have also been proposed for the CMST. On non-UD instances the best results are clearly those by Ahuja, Orlin and Sharma [1,2]. Good heuristics for UD instances include [2,3,33,37–39]. The algorithm proposed in this article is mainly an exact method. Moreover, it can also be combined with known heuristic techniques to provide high-quality solutions for instances that can not be solved to optimality.

Like many combinatorial problems, the CMST can be formulated as a set-partitioning problem. The rows would correspond to the non-root vertices and the columns to subtrees with degree 1 at the root and capacity not exceeding C. This formulation is not practical, since pricing over the exponential number of variables would require the solution of a strongly NP-hard problem (see Sect. 2). The exact algorithm proposed in this article utilizes variables that are associated

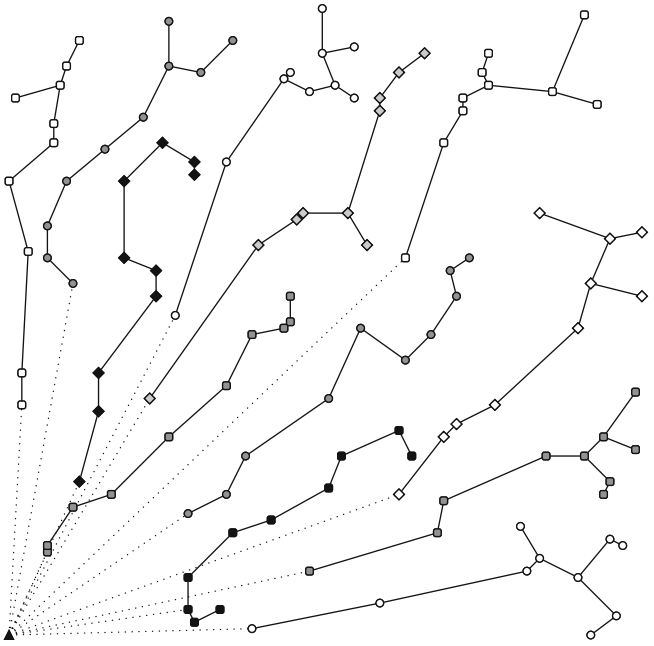

Fig. 1 Optimal Capacitated Minimum Spanning Tree (CMST) solution for the OR-Library instance te120-1-10, with $n = 120$ and $C = 10$ and unit demands. The root is represented by a triangle

to q-arbs, a structure that strictly contains the above mentioned subtrees. This relaxation allows the pricing problem to be solved in pseudo-polynomial time by dynamic programming, but it is still strong enough to capture some of the knapsack structure of the CMST. Besides, some families of cuts are added to provide a stronger formulation. The resulting branch-cut-and-price (BCP) algorithm was tested over the main instances from the literature and turned out to be very consistent. All instances with up to 100 vertices were solved, and most instances with up to 160 vertices were solved too. Moreover, the best upper bounds found in the literature were improved for most instances that could not be solved.

For some time, the combination of the branch-and-cut and branch-and-price techniques was not considered to be practical [7], since the new dual variables corresponding to separated cuts would have the undesirable effect of changing the structure of the pricing subproblem, making it intractable. However, in the late 1990s, several researchers [6,13,27,28,41,42] 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 last property defines what we call *robust branch-cut-and-price* algorithms.

Poggi de Aragão and Uchoa [36] present a discussion on this subject, stating that robust BCP algorithms could bring major algorithmic improvements on a wide variety of problems. This article on the CMST can be viewed as part of

a larger effort to support this statement [8,14,30,35]. However, it also brings an original contribution to the literature of such algorithms, by showing the possibility and the effectiveness of separating cuts expressed over a pseudo-polynomially large set of variables in a robust way.

Section 2 describes the integer programming formulations we will deal with and the families of cuts utilized. Section 3 describes our procedures, exact and heuristic, for the pricing of q-arbs. Section 4 presents efficient separation procedures for traditional Capacity Cuts and for a new family of cuts over the extended set of variables. Section 5 describes the primal heuristics embedded in the exact algorithm. Section 6 presents an empirical analysis of the overall branch-cut-and-price algorithm, making comparisons with other related approaches. Some additional remarks are made in the last section.

2 Formulations and valid inequalities

2.1 Formulations with arc variables

Although the CMST is defined on an undirected graph, it is preferable to use a directed formulation [25]. Specifically, the most effective formulations for the CMST work over a directed graph $G_D = (V, A)$, where A has a pair of opposite arcs (i, j) and (j, i) for each edge $e = \{i, j\} \in E$, excepting edges $\{0, i\}$ adjacent to the root, which are transformed into a single arc $(0, i)$. The arc costs correspond to the original edge costs. Now, in graph G_D, one looks for a minimum cost capacitated spanning arborescence directed from the root to each other vertex. Such an arborescence corresponds to a minimum cost capacitated spanning tree in the original graph G.

The arc (also known as two-index) CMST formulation uses binary variables x_a to indicate whether arc a belongs to the optimal solution. Denote the set of non-root vertices by $V_+ = \{1, \ldots, n\}$. For any set $S \subseteq V$, we let $d(S) = \sum_{i \in S} d_i$, $k(S) = \lceil d(S)/C \rceil$, $A(S) = \{(i, j) \in A : i, j \in S\}$, $\delta^-(S) = \{(i, j) \in A : i \in V \setminus S, j \in S\}$, and $\delta^+(S) = \{(i, j) \in A : i \in S, j \in V \setminus S\}$. Let $x(A') = \sum_{a \in A'} x_a$ for any $A' \subseteq A$. The formulation follows:

$$\text{Minimize} \quad \sum_{a \in A} c_a x_a \tag{1a}$$

$$\text{S.t.}$$

$$x(\delta^-(\{i\})) = 1 \qquad (\forall\, i \in V_+), \tag{1b}$$

$$x(\delta^-(S)) \geq k(S) \qquad (\forall\, S \subseteq V_+), \tag{1c}$$

$$x_a \in \{0, 1\} \qquad (\forall\, a \in A). \tag{1d}$$

The *In-Degree* constraints (1b) state that exactly one arc must enter each non-root vertex. *Capacity Cuts* (1c) state that at least $k(S)$ arcs must enter each set S. Instead of (1c), Hall [25] actually used in her branch-and-cut algorithm the

equivalent *Generalized Subtour Elimination* constraints

$$x(A(S)) \leq |S| - k(S) \quad (\forall S \subseteq V_+). \tag{2}$$

Another useful family of constraints are the *Root Cutset inequalities*. Define $S_\alpha = \{i \in V \setminus S : k(S \cup \{i\}) = k(S)\}$ and $S_\beta = (V \setminus S) \setminus S_\alpha$. Note that the root always belongs to S_α. The Root Cutset inequalities are

$$\tfrac{k(S)+1}{k(S)} x(\delta^-(S) \cap \delta^+(S_\alpha)) + x(\delta^-(S) \cap \delta^+(S_\beta)) \geq k(S) + 1 \quad (\forall S \subseteq V_+). \tag{3}$$

Those constraints are a strengthening of Capacity Cuts, based on the observation that if at least one of the subtrees covering S comes from a higher demand vertex in S_β, at least $k(S) + 1$ subtrees must enter S. Other families of valid inequalities that can potentially improve the arc formulation are known, including several variants of the so-called *Multistar* constraints [4,18, 19,25]. Even with all such inequalities, branch-and-cut algorithms over the arc formulation fail on many instances with just 50 vertices.

It seems that currently known valid inequalities are having trouble to capture the knapsack-like aspect of the CMST, related to the demands and capacities. We would like to obtain stronger formulations by introducing additional variables devised to capture that aspect. A natural set-partitioning formulation for the CMST would have variables corresponding to subtrees having degree 1 at the root and not exceeding the total demand C. However, this formulation is unpractical, since pricing over the exponential number of such variables requires the solution of a strongly NP-hard problem. The directed version of the well-known Prize Collecting Steiner Tree Problem has a linear reduction to the problem of finding a minimum cost (uncapacitated) arborescence in a directed graph where the arcs have both positive and negative costs [11]. One can easily change that reduction to prove that the problem of finding a minimum cost tree in an undirected graph is also strongly NP-hard.

Christofides, Mingozzi and Toth [10] faced a similar problem on the Capacitated Vehicle Routing Problem (CVRP). Instead of working with actual routes, which would lead to an intractable subproblem, they defined the q-route relaxation. A *q-route* is a walk that starts at the depot vertex, traverses a sequence of client vertices with total demand at most Q, 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. Minimum cost q-routes can be found in $O(n^2Q)$ time by dynamic programming. Recently, it was found that a formulation combining the q-routes with known valid inequalities for the edge (two-index) CVRP formulation is significantly stronger than previous formulations [14].

In order to achieve analogous results on the CMST, we introduce the concept of q-arbs, an arborescence-like structure directed from the root, having degree 1 at the root and with total demand at most C, but allowing some vertices (and even arcs) to appear more than once. More precisely,

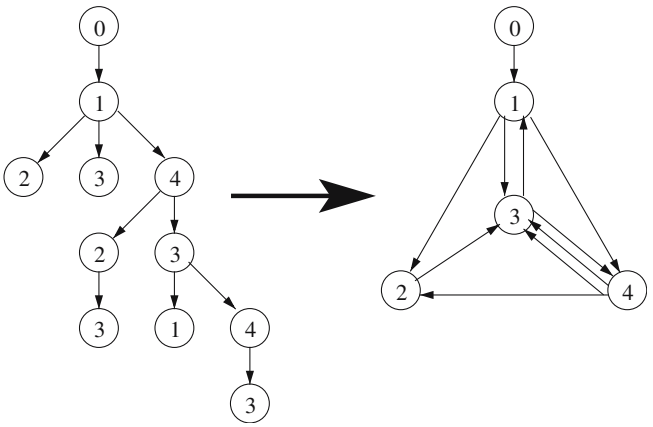

Fig. 2 A q-arb and its corresponding multigraph

Definition 1 *A* q-arb rooted at a vertex $i \neq 0$ *can be:*

- *The vertex i alone. In this case, the q-arb demand is* d_i.
- *The vertex i connected to k other q-arbs rooted at distinct vertices* $v_1, \ldots, v_k$ *by arcs* $(i, v_j) \in A$ *for* $j = 1, \ldots, k$. *The demand of this q-arb is* d_i *plus the demand of its k sub-q-arbs and must not exceed C.*

Finally, a q-arb rooted at *0, or just a* q-arb*, is a q-arb rooted at a vertex* $i \neq 0$ *plus an arc* $(0, i) \in A$.

On the left side of Fig. 2, an example of a q-arb over an UD instance with $C = 10$ is shown. The right of the same figure shows the multigraph over G_D that would be obtained by "folding" this q-arb. In this case, arc $(4, 3)$ appears twice.

A CMST formulation with an exponential number of variables and constraints can now be defined. Number all possible q-arbs from 1 to p. Define q_a^j as the number of times arc a appears in jth q-arb.

$$\text{Minimize} \quad \sum_{a \in A} c_a x_a \tag{4a}$$

S.t.

$$\sum_{j=1}^{p} q_a^j \lambda_j - x_a = 0 \quad (\forall\, a \in A), \tag{4b}$$

$$x(\delta^-(\{i\})) = 1 \quad (\forall\, i \in V_+), \tag{4c}$$

$$x(\delta^-(S)) \geq k(S) \quad (\forall\, S \subseteq V_+), \tag{4d}$$

$$\lambda_j \geq 0 \quad (j = 1, \ldots, p), \tag{4e}$$

$$x_a \in \{0, 1\} \quad (\forall\, a \in A). \tag{4f}$$

This formulation includes all variables and constraints from the arc formulation, but new constraints (4b) impose that x must be a weighted sum of arc-incidence

vectors of q-arbs. This restriction leads to a significantly stronger formulation. Pricing the λ variables requires the solution of minimum cost q-arb problems, which can be solved in pseudo-polynomial time.

When solving the linear relaxation of (4) by column and row generation, a more compact Master LP is obtained if every occurrence x_a in (4c)–(4d) is replaced by its equivalent given by (4b). The resulting LP will be referred to as the *Dantzig–Wolfe Master* (DWM):

$$\text{Minimize} \quad \sum_{j=1}^{p} \left(\sum_{a \in A} c_a q_a^j \right) \lambda_j \tag{5a}$$

S.t.

$$\sum_{j=1}^{p} \left(\sum_{a \in \delta^-(\{i\})} q_a^j \right) \lambda_j = 1 \qquad (\forall\, i \in V_+), \tag{5b}$$

$$\sum_{j=1}^{p} \left(\sum_{a \in \delta^-(S)} q_a^j \right) \lambda_j \geq k(S) \qquad (\forall\, S \subseteq V_+), \tag{5c}$$

$$\lambda_j \geq 0 \qquad (j = 1, \ldots, p). \tag{5d}$$

The reduced cost of a λ variable is the sum of the reduced costs of the arcs in the corresponding q-arb. Let ω and π be the dual variables associated with constraints (5b) and (5c), respectively. The reduced cost $\bar{c}_a$ of an arc a is given by:

$$\bar{c}_a = c_a - \omega_j - \sum_{S | \delta^-(S) \ni a} \pi_S \qquad (\forall a = (i,j) \in A). \tag{6}$$

Capacity Cuts are not the only ones that can appear in the DWM. A generic cut $\sum_{a \in A} \alpha_a x_a \geq b$ can be included as $\sum_{j=1}^{p} (\sum_{a \in A} \alpha_a q_a^j) \lambda_j \geq b$. This cut contributes to the computation of $\bar{c}_a$ with the value $-\alpha_a \beta$, where β is the new dual variable. The addition of cuts to the DWM affects the reduced costs but not the structure of the subproblem.

The possibility of adding such extra cuts naturally leads to the question of which cuts to add. Computational experiments have shown that also adding Root cutset inequalities only improve bounds modestly, while multistars are not even violated. This is consistent with previous experience with a BCP algorithm on the CVRP [14]. In that case, several complex families of cuts known to be effective in a pure BC algorithm were separated: framed capacities, generalized capacities, strengthened combs, CVRP multistars, and extended hypotours. Surprisingly, however, the resulting bounds were not significantly better than those obtained by only separating Capacity Cuts. A possible explanation for that (on the CVRP and on the CMST) is that most such cuts are already implicitly given by the combinatorial structure used in the column generation (q-routes and q-arbs, respectively). This explanation recently received some theoretical support. Letchford and Salazar [29] proved that generalized large multistar inequalities (for the CVRP) are indeed implied by the q-route definition.

In order to improve significantly over the bounds given by (5), we searched for other families of cuts, radically different from those currently used on the arc formulation.

2.2 Introducing capacity-indexed variables

Gouveia [17] presented a very interesting capacity-indexed formulation for the CMST. Let binary variables x_a^d indicate that arc $a = (i, j)$ belongs to the optimal arborescence and that the total demand of all vertices in the sub-arborescence rooted in j is exactly d.

$$\text{Minimize} \qquad \sum_{a \in A} c_a \sum_{d=1}^{C} x_a^d \tag{7a}$$

S.t.

$$\sum_{a \in \delta^-(\{i\})} \sum_{d=1}^{C} x_a^d = 1 \qquad (\forall\, i \in V_+), \tag{7b}$$

$$\sum_{a \in \delta^-(\{i\})} \sum_{d=1}^{C} \mathrm{d} x_a^d - \sum_{a \in \delta^+(\{i\})} \sum_{d=1}^{C} \mathrm{d} x_a^d = d_i \qquad (\forall\, i \in V_+), \tag{7c}$$

$$x_a^d \in \{0, 1\} \qquad (\forall\, a \in A;\ d = 1, \ldots, C). \tag{7d}$$

Equations (7b) are in-degree constraints and Eqs. (7c) are *Capacity–Balance* contraints that both prevents cycles and sub-arborescences with total demand greater than C. Note that variables x_{ij}^d with $d > C - d(i)$ can be removed. To provide a more simple and precise notation of this formulation, we define a directed multigraph $G_C = (V, A_C)$, where A_C contains arcs $(i, j)^d$, for each $(i, j) \in A, d = 1, \ldots, C - d(i)$. When working with variables x_a^d it is assumed that $\delta^-(\cdot)$ and $\delta^+(\cdot)$ are subsets of A_C. For example, equations (7c) will be written as:

$$\sum_{a^d \in \delta^-(\{i\})} \mathrm{d} x_a^d - \sum_{a^d \in \delta^+(\{i\})} \mathrm{d} x_a^d = d_i \qquad (\forall\, i \in V_+). \tag{8}$$

This formulation has only $2n$ constraints, but $O(Cm)$ variables. It can be shown that the capacity-indexed formulation is equivalent to a much more compact single-flow formulation in terms of bounds obtained by their linear relaxation [17]. Therefore, using the capacity-indexed formulation directly in a branch-and-bound algorithm is not interesting. However, this formulation may be useful in a branch-and-cut approach. Of course, since x_a can be defined as the sum of the x_a^d variables, for all existing d, any inequality valid for the arc formulation could be used in that algorithm. But the potential advantage of the capacity-indexed formulation is to allow the derivation and separation of new families of cuts defined over this pseudo-polynomially large extended variable space.

Recently, Gouveia and Saldanha-da-Gama [23] used another capacity-indexed formulation for the Capacitated Concentrator Location Problem, obtaining good results. The drawback of working with those kind of formulations is that the size of the LPs to be solved increases proportionally with the value of C. This is only practical for small values of capacity. In the CMST case, that approach could not be applied on most non-UD instances.

The capacity-indexed formulation can be naturally combined with the q-arbs, providing a new stronger formulation. Define q_a^{dj} as the number of arcs a carrying exactly d units of capacity in the j-th q-arb.

$$\text{Minimize} \quad \sum_{a^d \in A} c_a x_a^d \tag{9a}$$

S.t.

$$\sum_{j=1}^{p} q_a^{dj} \lambda_j - x_a^d = 0 \quad (\forall\, a^d \in A_C), \tag{9b}$$

$$\sum_{a^d \in \delta^-(\{i\})} x_a^d = 1 \quad (\forall\, i \in V_+), \tag{9c}$$

$$\sum_{a^d \in \delta^-(S)} x_a^d \geq k(S) \quad (\forall\, S \subseteq V_+), \tag{9d}$$

$$\lambda_j \geq 0 \quad (j = 1, \ldots, p), \tag{9e}$$

$$x_a^d \in \{0,1\} \quad (\forall\, a^d \in A_C). \tag{9f}$$

It can be noted that Capacity–Balance equalities (8) are already implied by the definition of q-arbs together with (9b). Eliminating the x variables, we can write the Dantzig–Wolfe Master as:

$$\text{Minimize} \quad \sum_{j=1}^{p} \left(\sum_{a^d \in A_C} q_a^{dj} c_a \right) \lambda_j \tag{10a}$$

S.t.

$$\sum_{j=1}^{p} \left(\sum_{a^d \in \delta^-(\{i\})} q_a^{dj} \right) \lambda_j = 1 \quad (\forall\, i \in V_+), \tag{10b}$$

$$\sum_{j=1}^{p} \left(\sum_{a^d \in \delta^-(S)} q_a^{dj} \right) \lambda_j \geq k(S) \quad (\forall\, S \subseteq V_+), \tag{10c}$$

$$\lambda_j \geq 0 \quad (j = 1, \ldots, p). \tag{10d}$$

This LP and (5) are exactly the same. But now it is clear that a generic cut $\sum_{a^d \in A_C} \alpha_a^d x_a^d \geq b$ can be included as $\sum_{j=1}^{p} (\sum_{a^d \in A_C} \alpha_a^d q_a^{dj}) \lambda_j \geq b$. This cut contributes to the computation of reduced cost $\bar{c}_a^d$ with the value $-\alpha_a^d \beta$, where β is the new dual variable.

This reformulation presents some remarkable features in a branch-cut-and-price context. It allows the introduction of new cuts over the capacity-indexed

variables, even for large values of C, without having to explicitly introduce any new variables. This means that the size of the LPs that are actually solved is basically unchanged. Moreover, those new cuts are robust with respect to the pricing of q-arbs. This means that computing a minimum q-arb using reduced costs $\bar{c}_a^d$ can still be done in pseudo-polynomial time, basically by the same dynamic programming algorithm.

2.3 Extended capacity cuts

We introduce a new family of cuts over the capacity-indexed variables. Let $S \subseteq V_+$ be a set of vertices. Summing the Equalities (8) corresponding to each $i \in S$, we get the *Capacity–Balance Equation over S*:

$$\sum_{a^d \in \delta^-(S)} \mathrm{d} x_a^d - \sum_{a^d \in \delta^+(S)} \mathrm{d} x_a^d = d(S) \ . \tag{11}$$

It can be noted that those equations are always satisfied by the solutions of (10) (translated to the x_a^d space by (9b)). Nevertheless, they can be viewed as the source of a rich family of cuts.

Definition 2 *An* Extended Capacity Cut (*ECC*) *over S is any inequality valid for P*(*S*)*, the polyhedron given by the convex hull of the* 0–1 *solutions of* (11).

The traditional Capacity Cuts (1c) could be derived only from the above definition: for a given S relax (11) to $\geq$, divide both sides by C and round coefficients up. Remember that $\delta^+(S)$ contains no arc with capacity C, so all such coefficients are rounded to zero. All coefficients corresponding to $\delta^-(S)$ are rounded to one. Therefore Capacity Cuts are ECCs. A slightly more complex reasoning shows that:

Proposition 1 *The Root Cutset inequalities* (3) *are ECCs.*

Proof For any $S \subseteq V_+$, the following inequality is clearly valid for $P(S)$:

$$\sum_{a^d \in \delta^-(S)} \mathrm{d} x_a^d \geq d(S). \tag{12}$$

Define $d^* = d(S) - C(k(S) - 1) - 1$. If at least one variable x_a^d with $d \leq d^*$ is set to one, we still need $k(S)$ additional variables set to one to satisfy (12). Otherwise, if all variables set to one have $d > d^*$, we need at least $k(S)$ such variables to satisfy (12). Hence

$$\frac{k(S)+1}{k(S)} \sum_{a^d \in \delta^-(S) : d > d^*} x_a^d + \sum_{a^d \in \delta^-(S) : d \leq d^*} x_a^d \geq k(S) + 1 \tag{13}$$

is also valid for $P(S)$ and dominates (3). □

The above results could be interpreted as saying that it is better to only separate ECCs, forgetting Capacity Cuts or Root Cutset inequalities. This interpretation is misleading, since it may be easier in practice to separate those important special cases of ECCs. In fact, our proposed BCP does utilize a specific separation of CCs. However, those results can be interpreted in a different way: if a violated Capacity Cut or Root Cutset inequality is found, one can usually perform a lifting of some coefficients (in the extended space) and actually introduce (13) in the LP.

In this article we only work with *Homogeneous Extended Capacity Cuts* (HECCs), a subset of the ECCs where all entering variables with the same capacity have the same coefficients, the same happening with the leaving variables. For a given set S, define aggregated variables y^d and z^d as follows:

$$y^d = \sum_{a^{d'} \in \delta^-(S) : d' = d} x_a^d \quad (d = 1, \ldots, C), \tag{14}$$

$$z^d = \sum_{a^{d'} \in \delta^+(S) : d' = d} x_a^d \quad (d = 1, \ldots, C-1). \tag{15}$$

The Capacity–Balance equation over those variables is:

$$\sum_{d=1}^{C} \mathrm{d}y^d - \sum_{d=1}^{C-1} \mathrm{d}z^d = d(S)\ . \tag{16}$$

For each possible pair of values of C and $D = d(S)$, we may define the polyhedron $P(C, D)$ induced by the integral solutions of (16). The inequalities that are valid for those polyhedra are HECCs. We used two approaches to obtain such inequalities:

- For small values of C, up to 10, we can actually compute the facets of $P(C, D)$, for different values of D, and store them in tables for posterior separation. The separation procedure must only choose suitable sets S and check if one of those facets is violated.
- For larger values of C, after selecting suitable sets S, the separation procedure tries to obtain violated HECCs by the following sequence of operations: relax the Eq. (16) corresponding to S to $\geq$, multiply all coefficients by a rational constant $r = a/b$, apply integer rounding and check if the resulting inequality is violated. Several such constants are tried for each set S.

We now present an example to illustrate why it can be much easier to find a violated cut over the capacity-indexed extended variables than over the traditional arc variables. Figure 3 displays part of a fractional x_a^d solution of a CMST UD instance with $C = 4$, over a set $S = \{1, 2, 3\}$. This fractional solution is obtained from the q-arb formulation, already including all Capacity Cuts. The set S is being covered by three different q-arbs, each one with associated λ variable equal to 1/2. The first q-arb enters at vertex 1 with capacity 4 (arc a) and leaves the set at vertex 2 (arc d) with capacity 2. The second q-arb

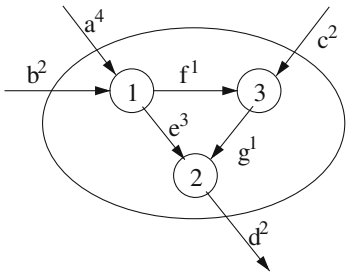

Fig. 3 Part of a fractional solution containing a violated Homogeneous Extended Capacity Cut (HECC)

enters at vertex 1 (arc b) with capacity 2 and does not leave the set. The third q-arb enters at vertex 3 (arc c) with capacity 2 and does not leave the set. The Capacity–Balance equality over the non-zero variables entering and leaving S is:

$$4x_a^4 + 2x_b^2 + 2x_c^2 - 2x_d^2 = 3.$$

As this equation has no 0–1 solution, there must be some violated ECC over S. In this case, applying the rounding with multiplier $r = 1/2$, a violated HECC is found:

$$2y^4 + 2y^3 + y^2 + y^1 - z^3 - z^2 \geq 2.$$

On the other hand, it is impossible to cut this fractional solution in the x_a space by only looking at the variables entering and leaving S, and even by also looking at those inside S. This is true because the incidence vector formed by $x_a = x_b = x_c = x_d = x_e = x_f = x_g = 1/2$ and all the remaining variables in $(\delta^-(S) \cup \delta^+(S) \cup A(S))$ equal to 0 is a convex combination of two valid CMST solutions: the first with one arborescence covering S using arcs $\{a, e, f\}$; the second solution covering S with two arborescences, using arcs $\{c, g, d\}$ and $\{b\}$. Of course, there must be some violated cut over the x_a space. But such a cut is likely to be much more complex to identify and separate.

3 Column generation

We now consider the problem of generating columns on the Dantzig–Wolfe Master (10). Each λ variable in that LP is associated to a q-arb rooted at 0, as in Definition 1. Define q_a^{dT} as the number of arcs a carrying exactly d units of capacity in the q-arb T. The reduced cost of a q-arb T is $\bar{c}(T) = \sum_{a^d \in A_C} q_a^{dT} \bar{c}_a^d$, where $\bar{c}_a^d$ is defined as in Subsect. 2.2. The pricing problem is finding a q-arb of minimum reduced cost.

Exact pricing

Given the reduced costs of all arcs in A_C, our exact pricing algorithm uses dynamic programming to find columns with minimum reduced costs. For that, we define the following subproblem. For $i, j \in V_+$ and $1 \leq d \leq C$, find the minimum reduced cost q-arb rooted at i, with demand d, such that $k \leq j$ for every child k of the root vertex. Let us denote the optimum q-arb for this subproblem by $T^*(i, j, d)$. Observe that the minimum reduced cost solution for our pricing problem must be one of the q-arbs obtained by connecting the root vertex 0 to the sub-q-arb $T^*(i, n, d)$ with the arc $(0, i)^d$, for all $i \in V_+$ and $d = 1, \ldots, C$.

Next, we show the recursion used by our dynamic programming algorithm. Assume that $\bar{c}(T^*(i, j, d)) = \infty$ whenever $0 < d < d_i$ and that $\bar{c}(T^*(i, j, d_i)) = 0$ for all $i, j \in V_+$.

$$\bar{c}(T^*(i,j,d)) = \min\{\bar{c}(T^*(i,j-1,d)), \tag{17}$$

$$\bar{c}_{ij}^{d-d_i} + \bar{c}(T^*(j,n,d-d_i)), \tag{18}$$

$$\min_{q=d_j,\ldots,d-d_i-1} \{\bar{c}(T^*(i,j-1,d-q)) + \bar{c}_{ij}^{q} + \bar{c}(T^*(j,n,q))\} \tag{19}$$

In the previous recursion, (17) represents the case where the root of $T^*(i, j, d)$ has no outgoing arc to vertex j. In this case, all the capacity d of $T^*(i, j, d)$ is used by $T^*(i, j-1, d)$. On the other hand, (18) represents the case where the root of $T^*(i, j, d)$ has no outgoing arc to a vertex numbered from 1 to $j-1$. In this case, all the capacity $d - d_i$ transferred through the outgoing arcs from the root of $T^*(i, j, d)$ is used by the sub-q-arb rooted at j. Finally, (19) represents the case where the capacity d of $T^*(i, j, d)$ is split between $T^*(i, j-1, d)$ and the sub-q-arb rooted at j.

Observe that it takes $O(C)$ time to solve each dynamic programming stage. Since we have $O(Cm)$ subproblems, the overall time complexity of this algorithm is $O(C^2 m)$. For each $i \in V_+$, we select a q-arb rooted at i and connect it to the root vertex. Hence, the reduced cost of the resulting column that corresponds to vertex i is given by

$$\min_{d=1,\ldots,C} \{\bar{c}_{0i}^{d} + \bar{c}(T^*(i,n,d))\}.$$

From these columns, we use the ones with negative reduced costs.

Heuristic pricing

In order to avoid calling the above expensive pricing too many times, we devised a heuristic pricing algorithm, based on a similar, but more restricted, dynamic programming recursion. Since it uses the same subproblem definition, let us denote the corresponding sub-optimal q-arbs by $T^h(i, j, d)$ for all $i, j \in V_+$ and $d = 1, \ldots, C$, that is, $T^h(i, j, d)$ denotes the sub-optimal q-arb rooted at i, with

demand d, such that $k \leq j$ for every child k of the root vertex. Our heuristic pricing algorithm uses the following recursion:

$$\begin{aligned}\bar{c}(T^h(i,j,d)) = \min\{&\bar{c}(T^h(i,j-1,d)),\\ &\bar{c}_{ij}^{d-d_i} + \bar{c}(T^h(j,n,d-d_i)),\\ &\bar{c}(T^h(i,j-1,d-q_1)) + \bar{c}_{ij}^{q_1} + \bar{c}(T^h(j,n,q_1)),\\ &\bar{c}(T^h(i,j-1,d-q_2)) + c_{ij}^{q_2} + \bar{c}(T^h(j,n,q_2))\},\end{aligned} \tag{20}$$

where

$$q_1 = \arg \min_{q=d_j,\ldots,d-d_i-1} \{\bar{c}(T^h(i,j-1,d-q))\}$$

and

$$q_2 = \arg \min_{q=d_j,\ldots,d-d_i-1} \{\bar{c}_{ij}^{q} + \bar{c}(T^h(j,n,q))\}.$$

The only difference between the previous recursion and the exact one is that we consider only two choices for the case where the capacity d of $T^h(i,j,d)$ is split between $T^h(i,j-1,d)$ and the sub-q-arb rooted at j: the choice that minimizes $\bar{c}(T^h(i,j-1,d-q))$ and the choice that minimizes $\bar{c}_{ij}^{q} + \bar{c}(T^h(j,n,q))$. In this case, $q_1 = q_1(i,j,d)$ and $q_2 = q_2(i,j,d)$ are the capacities used by the sub-q-arb rooted at j for the first and the second choices, respectively. Now, observe that we can write recursions for both $q_1(i,j,d)$ and $q_2(i,j,d)$ as follows:

$$q_1(i,j,d) = \arg \min_{q=q_1(i,j,d-1)+1,d_j} \{\bar{c}(T^h(i,j-1,d-q))\}$$

and

$$q_2(i,j,d) = \arg \min_{q=q_2(i,j,d-1),d-d_i-1} \{\bar{c}_{ij}^{q} + \bar{c}(T^h(j,n,q))\}.$$

Hence, if we maintain two tables with the values of $q_1(i,j,d)$ and $q_2(i,j,d)$ already calculated, we can calculate each new value for these tables in $O(1)$ time. The overall table calculation uses $O(Cm)$ time. Using this approach, one can solve (20) in $O(Cm)$.

Our heuristic uses a sparsification method in addition to the technique described before. In this case, the base instance graph is replaced by a sparse graph (V, A_-) obtained as follows. For each $(i,j),(j,i) \in A$ with $i,j \in V_+$, we define an edge $e = \{i,j\} \in E$ with cost $c_e = \min\{c_{(i,j)}, c_{(j,i)}\}$. Then we calculate the 20 edge-disjoint spanning trees with minimum total cost over the graph (V_+, E). The set of arcs A_- contains all $(i,j),(j,i)$ such that $e = \{i,j\}$ belongs to one of the constructed spanning trees. It also contains $(0,i)$ for all $i \in V_+$. In this case, the

tables $q_1(i,j,d)$, $q_2(i,j,d)$ and $T^h(i,j,d)$ only need be initialized for $(i,j) \in A_-$. Since $|A_-| = O(n)$, our heuristic runs in $O(Cn)$ time.

4 Cut separation

4.1 Capacity Cuts

We now consider the problem of separating Capacity Cuts in the arc CMST formulation. Specifically, given a solution $\bar{x}$ satisfying (1b) and $0 \le x_a \le 1 \,\forall\, a \in A$, the problem is to identify one or more sets S for which (1c) is violated (or to prove that no such S exists). For this separation problem we have implemented three heuristics which are described below. Throughout, *checking* a cut means that we generate the cut if it is violated.

Our first heuristic is based on connected components. We compute the connected components $S_1, \ldots, S_p$ in $\bar{G}_E = (V_+, \bar{E})$, where $\bar{E} = \{e = \{i,j\} \in E(V_+) : \bar{x}_{ij} + \bar{x}_{ji} > 0\}$. Then, for every $i = 1, \ldots, p$ we check the Capacity Cut for S_i and for $V_+ \setminus S_i$. Finally we check the Capacity Cut for the union of those components S_i for which $\sum_{j \in S_i} \bar{x}_{0j} = 0$.

If this heuristic fails to find any violated Capacity Cuts, we proceed by using our last two heuristics. All of these take as input a *shrunk* support graph, which is obtained by iteratively shrinking a vertex set $S \subset V_+$ into a single supervertex s having demand $d(S)$; arcs (s,j) and (j,s) in the shrunk graph are given the weights $\sum_{i \in S} \bar{x}_{ij}$ and $\sum_{i \in S} \bar{x}_{ji}$, respectively.

The shrinking is called *safe* if and only if it holds that for any $S \subseteq V_+$, the violation of the Capacity Cut for S is no larger than that for the union of those supervertices which are intersected by S. A sufficient condition for safe shrinking is stated formally in Proposition 2.

Proposition 2 *For separation of capacity cuts, it is safe to shrink a vertex set $S \subset V_+$ if $\bar{x}(\delta^-(S)) \le 1$ and $\bar{x}(\delta^-(R)) \ge 1 \,\forall R \subset S$.*

Proof Let $T \subset V_+$ be such that $T \cap S$, $T \setminus S$ and $S \setminus T$ are all nonempty. We show that, under the conditions stated, the Capacity Cut for $S \cup T$ is violated by at least as much as that for T. This holds if and only if $k(S \cup T) - x(\delta^-(S \cup T)) \ge k(T) - x(\delta^-(T))$. It is trivially true that $k(S \cup T) \ge k(T)$, so it suffices to show that $x(\delta^-(S \cup T)) \le x(\delta^-(T))$. It follows from the submodularity of the cut function $\bar{x}(\delta^-(S))$ [34, p. 660] that $\bar{x}(\delta^-(T)) - \bar{x}(\delta^-(S \cup T)) \ge \bar{x}(\delta^-(S \cap T)) - \bar{x}(\delta^-(S))$. Since $\bar{x}(\delta^-(S \cap T)) \ge 1$ and $\bar{x}(\delta^-(S)) \le 1$, it follows that $\bar{x}(\delta^-(S \cup T)) \le \bar{x}(\delta^-(T))$. □

In our code we iteratively shrink vertex sets of cardinality 2 and 3 until no such set satisfies the shrinking condition. Each non-root vertex in the shrunk graph is referred to as a *supervertex*, even if it represents only one original vertex.

Our second heuristic is based on the separation of *fractional* Capacity Cuts, which are obtained from (1c) by replacing the right-hand side with $d(S)/C$.

The motivation for considering Fractional Capacity Cuts lies in the fact that these can be separated exactly in polynomial time using max-flow computations. Specifically, the *Fractional Capacity Cut* can be written in the following form:

$$x(\delta^-(S)) + d(V_+ \setminus S)/C \geq d(V_+)/C \quad \forall S \subseteq V_+. \tag{21}$$

The right-hand side in (21) is constant, so a maximally violated Fractional Capacity Cut is obtained by identifing a set S that minimizes the left-hand side of (21).

The separation is done on a graph G' which is constructed as follows (to avoid introducing further notation we assume that no shrinking has been performed). The vertex set of G' is $V \cap \{n+1\}$. For each arc $(i,j) \in A$ with $\bar{x}_{ij} > 0$, G' contains the arc (i,j) with capacity $\bar{x}_{ij}$. Further, G' contains arc $(i, n+1)$ with capacity q_i/C for $i = 1, \ldots, n$. After computing the maximum flow from the source 0 to the sink $n+1$ in G', the set S is the set of vertices (excluding $n+1$) on the same side of the minimum cut as vertex $n+1$.

In our implementation we used a refined version of this in order to obtain several sets S, by fixing certain vertices on the source or sink side of the minimum cut. Specifically, setting the capacity of arc $(0,i)$ (arc $(i, n+1)$) to infinity fixes vertex i on the source side (sink side) of the minimum cut. Using this construction, we run the separation once for each supervertex being fixed on the sink side, and using a simple heuristic for fixing other supervertices on the source side in order to avoid generating previously generated sets. Finally, we check each of the corresponding Capacity Cuts.

Our third heuristic is a greedy construction heuristic, which is repeated once for each supervertex used as seed. Starting by setting S equal to the seed supervertex, we iteratively add to S the supervertex which implies the smallest slack (largest violation) of the Capacity Cut for the resulting S, subject to the restriction that this S has not been generated before. During the procedure we check the Capacity Cut for each generated set.

Finally, we note that our separation heuristics for capacity cuts are to a great extent inspired by [31]. Indeed, our three separation heuristics are closely related to the first three separation heuristics for rounded capacity cuts in [31], and our proposition 2 is closely related to proposition 1 in [31].

4.2 Extended Capacity Cuts

We divided the separation of Extended Capacity Cuts into two parts: (i) the generation of candidate sets S, and (ii) the search for violated cuts by considering the equation (11) corresponding to a given S. For the first part we chose a rather simply strategy, which is independent of the procedures in Sect. 4.1, namely enumeration of sets $S \subseteq V_+$ up to a given cardinality. In order to avoid an early combinatorial explosion, we restrict the search to sets S which are connected with respect to the support graph $\bar{G}_E$. As this graph is usually sparse, the computational savings are huge, allowing fast complete enumeration up to a respectable size of 10, even on larger instances. It is possible to show that such

connectivity restriction is safe, i.e., we are actually not losing any violated ECC. The proof of Lemma 1 is left to the appendix.

Lemma 1 *Let N_1 and N_2 be a non-trivial partition of the set $N = \{1, \ldots, m\}$; b_1, b_2 and b be real numbers such that $b_1 + b_2 = b$; and α be a m-dimensional vector of real numbers. Define polyhedra P, P_1 and P_2 as the convex hull of* 0–1 *solutions of $\sum_{j \in N} \alpha_j x_j = b$, $\sum_{j \in N_1} \alpha_j x_j = b_1$ and $\sum_{j \in N_2} \alpha_j x_j = b_2$, respectively. Let $\bar{x}$, $\bar{x}^1$ and $\bar{x}^2$ be m-dimensional vectors such that $\bar{x}^1$ and $\bar{x}^2$ are the projections of $\bar{x}$ on the subspaces generated by the canonical vectors $\{e_j | j \in N_1\}$ and $\{e_j | j \in N_2\}$, respectively. If $\bar{x}^1 \in P_1$ and $\bar{x}^2 \in P_2$ then $\bar{x} \in P$.*

Proposition 3 *For separation of ECCs, it is safe to consider only the sets whose induced subgraph of $\bar{G}_E$ is connected.*

Proof Let $\hat{S}$ be a subset of V_+ such that $\hat{S} = S_1 \cup S_2$, where S_1 and S_2 are disjoint subsets and there is no edge $(i, j) \in \bar{E}$ with $i \in S_1$ and $j \in S_2$. Let also $P(S)$ be the polyhedron given by the convex hull of 0–1 solutions of (11), for $S = \hat{S}, S_1, S_2$. Here, we consider that the dimension of $P(S)$ also includes the variables that do not appear in (11).

We must show that the existence of a violated ECC over the subset $\hat{S}$ implies a violated cut either over S_1 or over S_2. This is equivalent to showing that if $\bar{x}$ belongs to both $P(S_1)$ and $P(S_2)$, then it also belongs to $P(\hat{S})$, which follows immediately from Lemma 1. □

4.2.1 Facet HECCs

Each valid inequality, which is valid for the polyhedron $P(C, D)$ as defined in relation to (16), is a valid Homogeneous Extended Capacity Cut over a set S such that $d(S) = D$. We used the functions from the Parma Polyhedra Library (PPL) to compute the facets of some of those polyhedra. Interestingly, they do not have many facets for small values of C. For example,

$$P(5,6) = \left\{ \begin{array}{rrrrrrrrrr} y^1 & +2y^2 & +3y^3 & +4y^4 & +5y^5 & -z^1 & -2z^2 & -3z^3 & -4z^4 & = 6, \\ y^1 & +y^2 & +y^3 & +y^4 & +y^5 & & & & & \geq 2 \\ y^1 & +2y^2 & +2y^3 & +2y^4 & +3y^5 & & & -z^3 & -2z^4 & \geq 4 \\ 2y^1 & +2y^2 & +3y^3 & +4y^4 & +4y^5 & & -z^2 & -2z^3 & -2z^4 & \geq 6 \\ & & & & & & & & (y,z) & \geq 0 \end{array} \right\}.$$

It can be seen that one of the three non-trivial facets of $P(5, 6)$ is a Capacity Cut.

4.2.2 Rounded HECCs

The above procedure for separating facet HECCs is not suitable for larger values of C. For those cases, we implemented a procedure for separation over a particular subclass of HECCs, namely those that can be obtained from

(16) by applying integer rounding. In particular, we define a Rounded HECC (RHECC) as an inequality of the following form:

$$\sum_{d=1}^{C} \lceil rd \rceil y^d - \sum_{d=1}^{C-1} \lfloor rd \rfloor z^d \geq \lceil rd(S) \rceil \ , \tag{22}$$

where $0 < r \leq 1$.

In relation to reducing the computational effort of separating RHECCs, the *Farey sequence* plays a particular role. In particular, the Farey sequence F_C is the set of irreducible rational numbers a/b with $0 \leq a \leq b \leq C$ and arranged in increasing order (see [43]). We use the notation $F_C = (a_0/b_0, a_1/b_1, \ldots, a_p/b_p)$, where $a_0 = 0$ and $b_0 = a_p = b_p = 1$.

A basic observation is that if rd is integer in any left-hand side term of the RHECC, then r must be one of the numbers in F_C. This leads to the dominance relation expressed by Lemma 2.

Lemma 2 *For any multiplier r' satisfying $a_{i-1}/b_{i-1} < r' < a_i/b_i$ for a given $i \in [1;p]$, the RHECC obtained by using the multiplier r' is dominated by the RHECC obtained by using the multiplier $r = a_i/b_i$.*

Proof Using multiplier r' implies that all coefficients on the left-hand side are fractional before rounding, which means that each term is unchanged or strengthened if r' is replaced by r. Indeed, we have that $\lceil rd \rceil = \lceil r'd \rceil$ and $\lfloor rd \rfloor \geq \lfloor r'd \rfloor$ in any term on the left-hand side, and the right-hand side is not smaller for r than for r'. □

As such, among all possible values of the multiplier r, the search for a maximally violated RHECC can be reduced significantly by observing Proposition 4.

Proposition 4 *A most violated RHECC can be obtained by using a multiplier $r \in F_C$.*

Proof It follows from Lemma 2 that the multipliers in F_C collectively dominate the entire interval]0;1]. □

Since there are $O(C^2)$ multipliers in F_C (asymptotically, $|F_C|$ is approximately $0.3C^2$ [43]), a straightforward implementation of a separation routine which checks all RHECCs for a given $S \subset V_+$ would run in $O(C^3)$ time. However, in our implementation we are able to check all RHECCs in $O(C^2)$ time, as described in the following.

Considering any two consecutive multipliers r_i and r_{i+1}, we have that $\lceil r_{i+1}d \rceil - \lceil r_i d \rceil \in \{0, 1\}$ for any $d = 1, \ldots, C$. That is, in the two RHECCs, the two coefficients for any y-variable are equal or differ by one. A perfectly similar observation is made wrt. the coefficients of the z-variables.

Independently of S, we can therefore build a matrix of *cofficient increments*, i.e., a matrix D_{ij} with $|F_C|$ rows and $2C-1$ columns, such that for $i = 1, \ldots, p$, D_{ij} is the jth coefficient for multiplier a_i/b_i *minus* the jth coefficient for multiplier

a_{i-1}/b_{i-1}. The entries D_{0j} are all zero, corresponding to the multiplier 0/1. We note that the D-matrix contains only $O(C^2)$ nonzero entries.

Then, for a given fractional solution and a given S, we can in $O(C^2)$ time, by using a sparse matrix representation of D, compute the value of $\Delta_i = \text{LHS}_i - \text{LHS}_{i-1}$ for all $i = 1, \ldots, p$, where LHS_i is the value of the left-hand side of (22) using the multiplier a_i/b_i. Finally, a single pass in $O(|F_C|)$ time through all multipliers, using the accumulated Δ-values for the left-hand sides, is sufficient for finding the most violated RHECC.

Finally, we note that instead of using the integer rounding function leading to (22), it is possible to also use other subadditive functions to get stronger HECCs. We did some preliminary experiments along these lines but found that the extra computational effort involved was not justified in terms of improved results. Another issue left to future research is the use of non-homogeneous ECCs. Indeed, Eq. (11) may be viewed as a 0–1 knapsack equation from which lifted cover inequalities may be derived. In this context there is also the possibility of considering Generalized Upper Bounds based on (7b).

4.2.3 Generating Candidate Sets

Regarding the choice of sets S, we have implemented a depth-first search routine which enables us to enumerate all connected subgraphs of the support graph.

Our ECC separation on UD instances with $C \leq 10$ consists of checking each non-trivial facet of $P(C, |S|)$, for all sets S with cardinality up to 10 and connected in $\bar{G}_E$.

For ECC separation on UD instances with $C > 10$, and on non-UD instances, we invoke this routine repeatedly, each time with a given maximum depth, and such that we only check the RHECCs for those sets whose cardinality is equal to the maximum depth. Specifically, starting from a maximum depth of two, we first enumerate all connected subgraphs of cardinality two. Then we increase the maximum depth by one and repeat the search. This is done until one of three stopping conditions is satisfied: (i) All sets of cardinality 10 have been enumerated, (ii) The RHECCs have been checked for n^2 sets, or (iii) we have found 50 violated RHECCs.

5 Primal heuristics

We propose a primal heuristic to be embedded in the BCP. We implement the Esau-Williams constructive heuristic [12] and three local searches [2,3,38]. After solving every BCP node, we use the resulting LP relaxation to get pseudo-costs, favoring arcs with larger values of x_a. Those pseudo-costs are used in the constructive heuristic. Then, the local searches are applied using the correct costs, completing a try.

The pseudo-cost of an arc a is defined as $c'_a = c_a(1 - \alpha_a x_a)$, where α_a must be between zero and one. On each BCP node, there are four tries, with different

ranges of α values, randomly selected from the intervals $[1, 1]$, $[0.8, 1]$, $[0.6, 1]$, and $[0.5, 0.8]$, respectively. We also use the idea of "averaging" the values of x_a over successive node relaxations, in order to find out which arcs consistently belong to fractional solutions. We maintain $\bar{x}_a$ as a weighted average of past values of x_a. At the end of the root node, $\bar{x}_a$ is initialized with the current fractional solution. After each subsequent node solution, $\bar{x}_a$ is updated to $\beta\bar{x}_a + (1 - \beta)x_a$. The idea of taking convex combinations of primal solutions has been previously used to improve the convergence of subgradient methods [5]. In the reported tests, $\beta = 0.875$. Then we use the values of $\bar{x}_a$ to get pseudo-costs in two more tries, with values of α_a from the intervals $[1, 1]$, and $[0.5, 0.8]$, respectively. The overall idea is getting valuable information from the relaxations to guide the heuristics, but also add some degree of randomization to diversify the search.

Three local searches are applied to each solution obtained by the constructive heuristic. In all of them, the current solution is represented as a set of subtrees, each subtree being connected to the root vertex by a single arc. Given the set of vertices V_i of a subtree T_i, T_i is always the minimum cost tree that spans the vertices in V_i. Each local search consists in finding *moves* of sets of vertices from one subtree to another. Whenever the set of vertices of a subtree changes, the corresponding minimum spanning tree is recomputed. In this case, if the degree of the root vertex in the obtained subtree is greater than one, then this subtree is split so that each remaining subtree has only one arc from the root. The *moves* are selected so that the resulting tree satisfies the Capacity Cuts.

The first applied local search [3] searches for vertex exchanges between two subtrees, that is, for every two vertices v_i and v_j respectively in the subtrees T_i and T_j, it tries to move v_i to T_j and v_j to T_i. The second local search [38] uses a similar approach. For every two vertices v_i and v_j respectively in the subtrees T_i and T_j, it tries to move the subtree rooted at v_i to T_j and the subtree rooted at v_j to T_i. Both the first and the second local search are very fast but less effective than the third one. The third local search is a multi-exchange search [2] that generalizes the first two. In this search, each allowed transformation is represented by a sequence of sets of vertices $(S_1, \ldots, S_k)$, where S_i contains either a single vertex or the subtree rooted at a vertex, for $i = 1, \ldots, k$. Each set of vertices S_i in the sequence must belong to a different subtree T_i. In this case, the corresponding transformation consists in moving S_i to T_{i+1}, for $i = 1, \ldots, k - 1$, and S_k to T_1. In order to allow non-cyclic transformations, a sequence may contain an empty set of vertices. The procedure used to search for an improving sequence of sets of vertices uses a dynamic programming technique described in [2]. The complexity of searching for sequences with at most K sets is $O(n^3K2^K)$, but it is usually much faster in practice. In our experiments, we used K ranging from 3 to 5, the smaller values on instances where column and cut generation are faster. Therefore, the time spent on heuristics is a small fraction (usually less than 10%) of the total time to solve the node.

6 Computational results

The computational experiments were performed on the instances available at the OR-Library, the same ones as those used in the recent literature:

- The 45 non-UD cm instances correspond to complete graphs with 50, 100 or 200 vertices and capacities 200, 400 or 800. Costs and demands are integers generated from a uniform distribution [1, 100]. Instance names reflects their sizes and capacities, for example, cm50r1-200 is the first instance of a series of 5 instances with 50 vertices and capacity 200.
- There are three kinds of UD instances, all obtained from a random placement of non-root vertices in a 100×100 grid. The root has a central position, coordinates $(50, 50)$, in the tc instances. The te instances have the root in an eccentric position, coordinates $(0, 0)$. Finally, the root is distant from the other vertices on td instances, coordinates $(-201, -201)$. The instance costs correspond to truncated Euclidean distances. Each series, with up to 5 instances of the same kind, corresponds to graphs with 41, 81, 121 or 161 vertices and capacities 5, 10 or 20. The root vertex is not counted in the naming of the instances, for example, tc120-1-5 is the first instance of a series of kind tc having 121 vertices and capacity 5. Even disregarding the tc40s and te40s, which are easy by today standards, there are still 81 such UD instances.

All reported runs were done on Pentium 3 GHz machines, using CPLEX 9.0 to solve linear programs.

6.1 Comparison of lower bounds

The first two tables present a detailed comparison of lower bounding schemes over a sample of 30 instances. These instances were chosen to illustrate what happens on similar UD and non-UD instances for different values of C. Bound L_1 is obtained by the separation of CCs on the arc formulation. This bound is the root node bound in a BC over (1). Bound L_2 is obtained by only pricing q-arbs, as in a branch-and-price. Bound L_3 corresponds to the combination of q-arbs and CCs described in (5). Bound L_4 corresponds to the root node of our proposed BCP algorithm, obtained by also separating HECCs, as described in previous sections. Those bounds are compared with the best available in the literature. Average computation times are also reported. Some inferences can be drawn from Tables 1 and 2:

- Bound L_3 is already comparable or superior to previous known bounds on all those instances, but bound L_4 is significantly better. Moreover, L_4 is the only bound that produces stable integrality gaps over a wide range of C values, both on UD and non-UD instances.
- The proposed BCP algorithm would require large times on the UD instances with $C = 20$ and unacceptable times on the non-UD instances with $C = 800$. This behavior is due to the column generation part, whose convergence

Table 1 Comparison of lower bounds on the te80 instances

Instance	PrevLB	L1	L2	L3	L4	Opt
te80-1-5	2543.08	2405.20	2524.42	2541.54	2544.00	**2544**
te80-2	2537.12	2399.81	2516.37	2536.44	2545.03	**2551**
te80-3	2600.21	2438.80	2575.24	2602.42	2605.12	**2612**
te80-4	2546.20	2402.47	2529.03	2547.42	2551.08	**2558**
te80-5	2466.83	2339.96	2450.25	2463.87	2468.72	**2469**
Avg gap (%)	**0.32**	**5.86**	**1.09**	**0.33**	**0.16**	
Avg time (s)	786	70.1	1.77	6.46	20.9	
te80-1-10	1646.09	1586.52	1618.22	1644.65	1655.35	**1657**
te80-2	1609.95	1549.44	1589.57	1608.71	1624.71	**1639**
te80-3	1666.58	1614.13	1643.41	1667.45	1675.73	**1687**
te80-4	1627.22	1579.36	1601.98	1627.89	1629.00	**1629**
te80-5	1595.90	1553.43	1577.63	1595.65	1600.80	**1603**
Avg gap (%)	**0.84**	**4.04**	**2.24**	**0.86**	**0.36**	
Avg time (s)	3760	19.1	7.30	25.3	223.8	
te80-1-20	1256	1256.00	1214.99	1261.30	1264.11	**1275**
te80-2	1207	1201.80	1172.97	1202.98	1217.05	**1224**
te80-3	1257	1259.00	1218.18	1261.20	1263.72	**1267**
te80-4	1249	1246.50	1196.79	1252.61	1256.15	**1265**
te80-5	1230	1233.00	1178.67	1233.89	1236.92	**1240**
Avg gap (%)	**1.15**	**1.19**	**4.61**	**0.94**	**0.53**	
Avg time (s)	8469	2.10	31.5	199.4	638.8	

For $C = 5$ and $C = 10$, previous best bounds from [22], their times on AMD 1GHz machine. For $C = 20$, previous best bounds from [21], Pentium 180 MHz

depends on the average number of vertices in each subtree, becoming problematic when the subtrees are large. This is expected and consistent with the experience found in the general column generation literature. Column generation is a technique that allows the solution of LPs with a huge number of variables. However, it is quite reasonable that the number of iterations to solve such LPs still depends on "how huge" they are. There are many more possible q-arbs when $C = 20$ than when $C = 5$. When $C = 800$, the problem is particularly serious due to the worst-case complexity of the pricing, being quadratic over C.

Happily, the instances where the BCP performs poorly are exactly the ones where a BC over the arc formulation performs well. In such cases, bound L_1 can be computed quickly and is not much worse than L_3. It is interesting to note that bound L_4 improves significantly over L_3 even on such instances. However, the elevated computational times make the use of L_4 unattractive. In order to have a consistent overall method, we also implemented that pure BC. In [14], for each particular instance, the so-called Dyn-BCP algorithm tried to obtain both BC and BCP root bounds and times, in order to decide how it should continue. In this paper there was no need for such an algorithm since the best algorithm for each benchmark series was quite obvious.

Table 2 Comparison of lower bounds on the cm50 instances

Instance	PrevLB	L1	L2	L3	L4	Opt
cm50r1-200	1039.4	1034.04	1073.29	1093.76	1097.90	**1098**
cm50r2	913.1	915.58	937.70	962.40	971.60	**974**
cm50r3	1131.1	1139.62	1154.12	1180.03	1185.78	**1186**
cm50r4	768.9	774.21	758.82	792.33	797.73	**800**
cm50r5	867.7	867.48	894.94	918.00	922.79	**928**
Avg gap (%)	**5.32**	**5.10**	**3.48**	**0.82**	**0.22**	
Avg time (s)	560	3.01	7.38	31.9	115.8	
cm50r1-400	645.4	651.79	652.41	669.65	676.70	**679**
cm50r2	608.8	620.00	592.70	627.45	631.00	**631**
cm50r3	704.3	714.17	678.10	721.92	726.15	**732**
cm50r4	543.5	545.38	489.96	551.73	561.63	**564**
cm50r5	586.5	588.80	565.05	599.00	609.66	**611**
Avg gap (%)	**3.64**	**3.02**	**7.60**	**1.49**	**0.36**	
Avg time (s)	605	0.74	74.9	320.9	2251	
cm50r1-800	482.6	492.50	456.61	493.06	494.69	**495**
cm50r2	502.4	508.00	444.37	508.18	510.09	**513**
cm50r3	522.3	529.00	470.23	529.67	530.91	**532**
cm50r4	468.9	471.00	373.98	471.00	471.00	**471**
cm50r5	477.4	484.00	429.48	485.39	491.21	**492**
Avg gap (%)	**1.77**	**0.73**	**13.21**	**0.62**	**0.20**	
Avg time (s)	711	0.32	817	6679	38715	

Previous best bounds from [9], their times on Pentium III 933 MHz machine. Hall [25] did not run those instances, her method would probably give results close to column L_1

6.2 Complete runs

Tables 3 and 4 present BCP results over the 81 instances where this algorithm performs better than the BC. Results of this latter algorithm on the remaining 45 instances are shown in Table 5. The assignment of series to algorithms turned out to be clear, the instances from Tables 3 and 4 (even from the tc80-5, te80-10 or cm50-200 series) can not be solved by the BC in reasonable time. Columns **RootT**, **Nodes** and **TotalT** are the root time, number of nodes explored, and total algorithm time. All such times are in seconds. The runs were aborted without optimality on 26 larger/harder instances. In those cases, the number of nodes and times at that moment are reported. Column **PrevUB** gives the best upper bounds found in the literature, their compilation is available in [33] on UD instances and in [1,2] on non-UD. Values in bold are proven optima. Column **Ours** gives the upper bounds produced by our runs, bold numbers indicate that the instance was solved to proven optimality. Underlined values indicate improved upper bounds. Some additional information and remarks:

- Adding HECCs in the BCP increases root times by a factor between 2 and 10. This factor is usually smaller in non-root nodes. Since those cuts lead to good gap reductions, as shown in columns L_3 and L_4, the overall gains can

be large. For example, te80-2-10 and te80-3-10 would not have been solved without the new cuts.
- Having a good relaxation proved to be very valuable in guiding primal heuristics. Our local searches were proposed by other authors, but they consistently obtain better results in less tries when embedded in the BCP. For example, on cm200r1-200 the embedded heuristic obtains a solution of 1032 already on the first try, after solving the root node. We found that a stand-alone GRASP using the same searches takes more time (and makes hundreds of tries) to get a solution of similar quality. Several improving solutions are found at the next nodes. The reported solution with value 994 corresponds to an improvement of more than 2% over the previous best solution of 1017.

7 Conclusion

This paper presents an effective branch-cut-and-price algorithm for the CMST. Some general conclusions from this experience can be derived, reinforcing ideas obtained from previous experiences with the same kind of algorithms on other problems:

- On many problems with an underlying knapsack-like aspect, the combination of simple cuts with column generation over combinatorial structures devised to capture that aspect is a more practical way of improving bounds than searching for increasingly complicated families of cuts. This can be true even when the bounds provided by that column generation alone are not good, i.e. pure branch-and-price is not an option.
- In order to have a robust algorithm (in the usual meaning of the word), it is usually necessary to combine cut and column generation in a robust way (in the technical meaning employed in this article). The complexity of the pricing subproblem should be controlled. If the natural combinatorial structure for the column generation leads to a strongly NP-hard problem, one must find a relaxation of that structure that is guaranteed to be tractable, at least when the numbers involved have a reasonable size. On the other hand, the pricing subproblem should not be too easy. Subproblems that can be solved in polynomial time do not currently properly capture knapsack-like aspects that are hard to be captured by cuts.
- Even if most instances of a problem benefits from the BCP approach, some instances are still much better solved by a traditional BC algorithm.

When subproblems of pseudo-polynomial complexity are being solved by dynamic programming algorithms, there is a new idea to be explored. One may introduce up to one variable to each possible choice inside the dynamic programming. Each such variable, and there are a pseudo-polynomially large number of them, corresponds to a very specific choice. It is possible that cuts expressed over such detailed set of variables are easier to be identified and separated.

Table 3 Branch-cut-and-price results over 56 UD instances

Instance	L3	L4	RootT	Nodes	TotalT	PrevUB	Ours
tc80-1-5	1098.2	1099.0	10.7	1	10.7	**1099**	**1099**
tc80-2	1097.4	1099.3	15.5	5	32.4	**1100**	**1100**
tc80-3	1070.5	1071.8	14.5	3	17.0	**1073**	**1073**
tc80-4	1074.9	1077.0	13.4	11	57.3	**1080**	**1080**
tc80-5	1278.8	1282.5	14.5	46	219.9	**1287**	**1287**
te80-1-5	2541.5	2544.0	17.3	1	17.3	**2544**	**2544**
te80-2	2536.4	2545.0	21.1	37	194.2	**2551**	**2551**
te80-3	2602.4	2605.1	18.9	39	199.4	**2612**	**2612**
te80-4	2547.4	2551.1	22.9	97	508.7	**2558**	**2558**
te80-5	2463.9	2468.7	24.6	1	24.6	**2469**	**2469**
te80-1-10	1644.7	1655.4	316.3	7	474.4	**1657**	**1657**
te80-2	1608.7	1624.7	378.2	1105	29134	1639	**1639**
te80-3	1667.5	1675.7	212.1	767	22349	1687	**1687**
te80-4	1627.9	1629.0	52.6	1	52.6	**1629**	**1629**
te80-5	1595.6	1600.8	161.9	4	254.8	**1603**	**1603**
td80-1-5	6057.1	6064.6	22.2	86	321.9	**6068**	**6068**
td80-2	6011.4	6018.5	20.7	2	23.3	**6019**	**6019**
td80-3	5983.3	5991.3	16.4	20	92.0	**5994**	**5994**
td80-4	6006.8	6011.7	17.2	13	51.2	**6012**	**6012**
td80-5	5966.7	5975.7	36.6	3	43.1	**5977**	**5977**
td80-1-10	3205.9	3217.6	544.6	303	21326	3236	**3223**
td80-2	3190.2	3201.3	288.2	25	1424	3206	**3205**
td80-3	3190.3	3199.6	429.6	5067	204175	3212	**3212**
td80-4	3190.6	3199.3	538.3	15	1343	3204	**3203**
td80-5	3167.0	3177.3	493.5	13	1151	3184	**3180**
tc120-1-5	1280.5	1287.6	24.0	19	87.5	**1291**	**1291**
tc120-2	1180.2	1185.8	43.6	223	1185	1189	**1189**
tc120-3	1115.9	1121.9	22.0	25	170.0	1124	**1124**
tc120-4	1121.0	1124.5	40.0	5	54.6	**1126**	**1126**
tc120-5	1153.3	1157.1	31.4	4	58.6	**1158**	**1158**
tc120-1-10	889.8	898.8	493.8	273	15448	904	**904**
tc120-2	744.2	750.7	648.9	3873	174827	757	**756**
tc120-3	707.6	714.9	724.3	12536	499685	725	**722**
tc120-4	716.0	722.0	1300	1	1300	**722**	**722**
tc120-5	743.5	753.9	521.3	4554	191195	762	**761**
te120-1-5	2193.1	2197.0	34.6	1	34.6	**2197**	**2197**
te120-2	2123.6	2129.4	41.4	725	6006	2137	**2134**
te120-3	2073.3	2076.7	40.0	35	317.0	**2079**	**2079**
te120-4	2149.3	2154.3	64.7	159	1178	2161	**2158**
te120-5	2011.6	2015.2	56.0	9	146.8	2021	**2017**
te120-1-10	1315.7	1324.6	665.4	120	7091	1329	**1329**
te120-2	1210.2	1219.5	1131	4013	240541	1229	**1225**
te120-3	1179.9	1188.3	760.4	2328	175670	1197	1195
te120-4	1198.5	1223.3	934.7	2328	163866	1234	1230
te120-5	1153.1	1161.6	1815	25	6998	1166	**1164**
te120-1-20	908.1	912.1	1424	1692	830173	920	**920**
te120-2	769.5	773.7	1414	507	227423	789	785
te120-3	739.7	741.6	1197	528	246816	755	749
te120-4	757.7	762.4	2162	555	247754	774	773
te120-5	737.1	740.8	1516	344	225517	755	746

Table 3 continued

Instance	L3	L4	RootT	Nodes	TotalT	PrevUB	Ours
tc160-1-5	2061.7	2074.4	93.2	13	280.6	2081	**2077**
tc160-1-10	1287.6	1309.6	1228	2102	303095	1319	1319
tc160-1-20	948.5	953.8	4104	23	26531	960	960
te160-1-5	2779.7	2785.7	112.8	44	938.0	2790	**2789**
te160-1-10	1623.6	1637.4	2499	1348	206063	1646	1645
te160-1-20	1086.6	1089.8	4692	165	226346	1098	1098

Table 4 Branch-cut-and-price results over 25 cm instances

Instance	L3	L4	RootT	Nodes	TotalT	PrevUB	Ours
cm50r1-200	1093.8	1097.7	137.1	1	137.1	1098	**1098**
cm50r2	962.9	971.7	130.8	5	239.5	974	**974**
cm50r3	1180.0	1185.6	131.6	2	151.7	1186	**1186**
cm50r4	792.3	798.0	80.2	3	120.6	800	**800**
cm50r5	918.0	922.7	131.1	30	733.6	928	**928**
cm100r1-200	501.6	506.8	273.5	85	7082	516	**509**
cm100r2	577.9	579.7	152.0	471	28025	596	**584**
cm100r3	531.9	533.5	287.9	885	82637	541	**540**
cm100r4	428.6	430.7	166.3	99	7837	437	**435**
cm100r5	412.8	414.7	180.6	251	16592	425	**418**
cm100r1-400	249.3	250.8	5510	11	18489	252	**252**
cm100r2	275.4	276.5	2971	12	14653	278	**277**
cm100r3	234.2	236.0	6931	2	8135	236	**236**
cm100r4	215.6	217.8	4591	7	12188	219	**219**
cm100r5	219.8	221.9	2751	15	18032	223	**223**
cm200r1-200	970.7	974.0	1442	83	33542	1017	994
cm200r2	1170.5	1175.0	1484	27	17426	1221	1188
cm200r3	1298.7	1303.1	1045	76	31358	1365	1313
cm200r4	898.6	902.9	1660	44	20397	927	917
cm200r5	922.2	926.7	1246	63	32316	965	948
cm200r1-400	379.2	382.6	18913	30	234387	397	391
cm200r2	460.8	465.0	21270	41	223765	478	476
cm200r3	540.6	545.1	22741	36	215625	560	559
cm200r4	378.7	382.0	16539	89	302842	392	389
cm200r5	407.3	411.1	21335	65	228001	420	418

The experiments here described on the CMST are encouraging, the addition of one such family of cuts, the HECCs, could reduce substantially the integrality gaps, without making the resolution of a node much slower. Those improvements are consistent in all kinds of tested instances, UD or non-UD, small or large capacities. Several hard instances would not have been solved without the new cuts.

Table 5 Branch-and-cut results over the remaining 45 instances

Instance	L1	RootT	Nodes	TotalT	PrevUB	Ours
tc80-1-10	877.0	1.21	153	40.4	**888**	**888**
tc80-2	876.3	0.64	1	0.64	**877**	**877**
tc80-3	870.4	1.00	630	17.0	**878**	**878**
tc80-4	861.3	1.03	193	57.3	**868**	**868**
tc80-5	1000.0	1.21	4	2.14	**1002**	**1002**
tc80-1-20	834.0	0.39	2	0.50	**834**	**834**
tc80-2	820.0	0.31	8	0.51	**820**	**820**
tc80-3	824.0	0.12	1	0.12	**824**	**824**
tc80-4	820.0	0.26	1	0.26	**820**	**820**
tc80-5	916.0	0.23	1	0.23	**916**	**916**
te80-1-20	1256.0	2.45	226	99.0	1275	**1275**
te80-2	1206.8	3.48	243	69.8	1224	**1224**
te80-3	1259.0	3.67	43	9.90	**1267**	**1267**
te80-4	1246.5	1.75	194	41.3	1265	**1265**
te80-5	1233.0	2.43	12	3.62	**1240**	**1240**
td80-1-20	1256.0	2.45	32	6.81	1833	**1832**
td80-2	1206.8	3.48	844	423.4	1830	**1829**
td80-3	1259.0	3.67	204	86.0	1839	**1839**
td80-4	1246.5	1.75	1192	1647.8	1834	**1834**
td80-5	1233.0	2.43	428	95.3	1829	**1826**
tc120-1-20	763.5	3.73	27	9.59	768	**768**
tc120-2	565.4	7.68	16	13.8	569	**569**
tc120-3	529.4	10.8	157	101.7	537	**536**
tc120-4	561.3	6.73	7436	7392	572	**571**
tc120-5	574.2	4.87	200	67.7	581	**581**
cm50r1-400	651.8	0.87	905	233.1	681	**679**
cm50r2	620.0	0.87	216	39.9	631	**631**
cm50r3	714.1	0.92	1007	272.7	732	**732**
cm50r4	545.4	0.23	207	11.0	564	**564**
cm50r5	588.8	0.65	1188	279.3	611	**611**
cm50r1-800	492.5	0.50	87	4.06	495	**495**
cm50r2	508.0	0.75	267	30.0	513	**513**
cm50r3	529.0	0.68	7	1.12	532	**532**
cm50r4	471.0	0.14	1	0.14	**471**	**471**
cm50r5	484.0	0.35	170	7.68	492	**492**
cm100r1-800	181.7	2.93	5	4.26	182	**182**
cm100r2	177.7	8.03	106	75.8	179	**179**
cm100r3	173.5	3.34	92	29.2	175	**175**
cm100r4	181.0	2.95	117	43.2	183	**183**
cm100r5	184.0	4.18	116	60.4	186	**186**

Acknowledgements E.U. received support from Mestrado em Engenharia de Produção-UFF and CNPq grant 304533/02-5. R.F. was supported by NSF grant DMI-0245609 and ONR grant N00014-03-1-0040, having started this work while at Gapso Inc., Brazil. A.P. was supported by a CAPES-PRODOC grant. M.P.A. was funded by CNPq grant 300475/93-4.

Appendix

Proof of Lemma 1

Since $\bar{x}^k \in P_k$, for $k = 1, 2$, we may write $\bar{x}^k = \sum_{i=1}^{q_k} \lambda_i^k \hat{x}^{i,k}$, where $\hat{x}^{i,k}$ is a 0–1 vector that belongs to P_k, $\lambda_i^k \geq 0$, and $\sum_{i=1}^{q_k} \lambda_i^k = 1$. Moreover, since $\left(\bar{x}^1\right)' \bar{x}^2 = 0$, we have that $\left(\hat{x}^{i,1}\right)' \hat{x}^{j,2} = 0$, for all $i = 1, \ldots, q_1$ and $j = 1, \ldots, q_2$. As a result, $\hat{x}^{i,1} + \hat{x}^{j,2}$ is a 0–1 vector that belongs to P. Hence, we only need to write $\bar{x}^1 + \bar{x}^2$ as a convex combination of $\hat{x}^{i,1} + \hat{x}^{j,2}$, for $i = 1, \ldots, q_1$ and $j = 1, \ldots, q_2$. First, observe that

$$\sum_{i=1}^{q_1} \sum_{j=1}^{q_2} \lambda_i^1 \lambda_j^2 \left(\hat{x}^{i,1} + \hat{x}^{j,2}\right) = \sum_{i=1}^{q_1} \lambda_i^1 \left(\sum_{j=1}^{q_2} \lambda_j^2\right) \hat{x}^{i,1} + \left(\sum_{i=1}^{q_1} \lambda_i^1\right) \sum_{j=1}^{q_2} \lambda_j^2 \hat{x}^{j,2}$$
$$= \sum_{i=1}^{q_1} \lambda_i^1 \hat{x}^{i,1} + \sum_{j=1}^{q_2} \lambda_j^2 \hat{x}^{j,2} = \bar{x}^1 + \bar{x}^2.$$

Moreover, we have that $\sum_{i=1}^{q_1} \sum_{j=1}^{q_2} \lambda_i^1 \lambda_j^2 = \sum_{i=1}^{q_1} \lambda_i^1 \left(\sum_{j=1}^{q_2} \lambda_j^2\right) = 1$. □

References

1. Ahuja, R., Orlin, J., Sharma, D.: Multi-exchange neighborhood search structures for the capacitated minimum spanning tree problem. Math. Program. **91**, 71–97 (2001)
2. Ahuja, R., Orlin, J., Sharma, D.: A composite very large-scale neighborhood structure for the capacitated minimum spanning tree problem. Oper. Res. Lett. **31**(3), 185–194 (2003)
3. Amberg, A., Domschke, W., Voss, S.: Capacitated minimum spanning trees: Algorithms using intelligent search. Comb. Optim. Theory Pract. **1**, 9–39 (1996)
4. Araque, J., Hall, L., Magnanti, T.: Capacitated trees, capacitated routing, and associated polyhedra. Technical Report OR232-90, MIT, Operations Research Center (1990)
5. Barahona, F., Anbil, R.: The volume algorithm: producing primal solutions with a subgradient method. Math. Program. **87**, 385–399 (2000)
6. 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)
7. Barnhart, C., Johnson, E.L., Nemhauser, G.L., Savelsbergh, M.W.P., Vance, P.H.: Branch-and-price: column generation for solving huge integer programs. Oper. Res. **46**, 316–329 (1998)
8. Belov, G., Letchford, A., Uchoa, E.: A node-flow model for 1D stock cutting: Robust branch-cut-and-price. Tech. Rep. RPEP Vol.5 no.7, Universidade Federal Fluminense, Engenharia de Produção, Niterói, Brazil (2005)
9. Bergson, J.: Uma heurística lagrangeana para o problema da árvore geradora capacitada de custo mínimo. Master's thesis, Universidade Federal do Rio de Janeiro (2002)
10. Christofides, N., Mingozzi, A., Toth, P.: Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations. Math. Program. **20**, 255–282 (1981)
11. Duhamel, C., Gouveia, L., Moura, P., Souza, M.C.: Models and heuristics for a minimum arborescence problem. Tech. Rep. LIMOS/RR-04-13, Institut Supérieur d'Informatique, de Modélisation et de leurs Applications (2004). URL: http://www.isima.fr/limos/publi/RR-04-13.pdf
12. Esau, L., Williams, K.: On teleprocessing system design part ii: a method for approximating the optimal network. IBM Syst. J. **5**, 142–147 (1966)
13. Felici, G., Gentile, C., Rinaldi, G.: Solving large MIP models in supply chain management by branch & cut. Tech. rep., Istituto di Analisi dei Sistemi ed Informatica del CNR, Italy (2000)

14. Fukasawa, R., Longo, H., Lysgaard, J., Poggi de Aragão, M., Reis, M., Uchoa, E., Werneck, R.F.: Robust branch-and-cut-and-price for the capacitated vehicle routing problem. Math. Program. **106**, 491–511 (2006)
15. Gabow, H.N., Galil, Z., Spencer, T., Tarjan, R.E.: Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. Combinatorica **6**(2), 109–122 (1986)
16. Gavish, B.: Formulations and algorithms for the capacitated minimal directed tree problem. J. ACM **30**, 118–132 (1983)
17. Gouveia, L.: A 2n-constraint formulation for the capacitated minimal spanning tree problem. Oper. Res. **43**, 130–141 (1995)
18. Gouveia, L., Hall, L.: Multistars and directed flow formulations. Networks **40**, 188–201 (2002)
19. Gouveia, L., Lopes, M.: The capacitated minimum spanning tree problem: On improved multistar constraints. Eur. J. Oper. Res. **160**, 47–62 (2005)
20. Gouveia, L., Martins, P.: The capacitated minimal spanning tree problem: an experiment with a hop-indexed model. Ann. Oper. Res. **86**, 271–294 (1999)
21. Gouveia, L., Martins, P.: A hierarchy of hop-indexed models for the capacitated minimum spanning tree problem. Networks **35**, 1–16 (2000)
22. Gouveia, L., Martins, P.: The capacitated minimum spanning tree problem: revisiting hop-indexed formulations. Comput. Oper. Res. **32**, 2345–2452 (2005)
23. Gouveia, L., Saldanha-da-Gama, F.: On the capacitated concentrator location problem: a reformulation by discretization. Comput. Oper. Res. **33**, 1242–1258 (2006)
24. Gzara, F., Goffin, J.L.: Exact solution of the centralized network design problem on directed graphs. Networks **45**, 181–192 (2005)
25. Hall, L.: Experience with a cutting plane algorithm for the capacitated spanning tree problem. INFORMS J. Comput. **8**, 219–234 (1996)
26. Hall, L., Magnanti, T.: A polyhedral intersection theorem for capacitated spanning trees. Math. Oper. Res. **17** (1992)
27. Kim, D., Barnhart, C., Ware, K., Reinhardt, G.: Multimodal express package delivery: a service network design application. Transport. Sci. **33**, 391–407 (1999)
28. Kohl, N., Desrosiers, J., Madsen, O., Solomon, M., Soumis, F.: 2-Path cuts for the vehicle routing problem with time windows. Transport. Sci. **33**, 101–116 (1999)
29. Letchford, A.N., Salazar, J.J.: Projection results for vehicle routing. Math. Program. **105**, 251–274 (2006)
30. Longo, H., Poggi de Aragão, M., Uchoa, E.: Solving capacitated arc routing problems using a transformation to the CVRP. Comput. Oper. Res. **33**, 1823–1837 (2006)
31. Lysgaard, J., Letchford, A., Eglese, R.: A new branch-and-cut algorithm for the capacitated vehicle routing problem. Math. Program. **100**, 423–445 (2004)
32. Malik, K., Yu, G.: A branch and bound algorithm for the capacitated minimum spanning tree problem. Networks **23** (1993)
33. Martins, P.: Enhanced second order algorithm applied to the capacitated minimum spanning tree problem. Comput. Oper. Res. (2006). (To appear)
34. Nemhauser, G., Wolsey, L.: Integer and combinatorial optimization. Wiley New York (1988)
35. Pigatti, A., Poggi de Aragão, M., Uchoa, E.: Stabilized branch-and-cut-and-price for the generalized assignment problem. In: Annals of GRACO'05, Electronic Notes in Discrete Mathematics, vol. 19, pp. 389–395 (2005)
36. Poggi de Aragão, M., Uchoa, E.: Integer program reformulation for robust branch-and-cut-and-price. In: Annals of Mathematical Programming in Rio, pp. 56–61. Búzios, Brazil (2003)
37. Rolland, E., Patterson, R., Pirkul, H.: Memory adaptive reasoning and greedy assignment techniques for the capacitated minimum spanning tree problem. J. Heuristics **5**, 159–180 (1999)
38. Sharaiha, Y., Gendreau, M., Laporte, G., Osman, I.: A tabu search algorithm for the capacitated shortest spanning tree problem. Networks **29**, 161–171 (1997)
39. Souza, M., Duhamel, C., Ribeiro, C.: A GRASP heuristic for the capacitated minimum spanning tree problem using a memory-based local search strategy. In: M. Resende, J. de Sousa (eds.) Metaheuristics: Computer decision-making, pp. 627–658. Kluwer Academic Publishers, Dordrecht (2003)
40. Toth, P., Vigo, D.: An exact algorithm for the capacitated shortest spanning arborescence. Ann. Oper. Res. **61**, 121–141 (1995)

41. Van den Akker, J., Hurkens, C., Savelsbergh, M.: Time-indexed formulation for machine scheduling problems: column generation. INFORMS J. Comput. **12**, 111–124 (2000)
42. Vanderbeck, F.: Lot-sizing with start-up times. Manage. Sci. **44**, 1409–1425 (1998)
43. Weisstein, E.: Farey sequence. From MathWorld–A Wolfram Web Resource (2006). URL: http://mathworld.wolfram.com/FareySequence.html