EURO J Comput Optim (2018) 6:143–184
https://doi.org/10.1007/s13675-017-0091-5

ORIGINAL PAPER

# The impact of filtering in a branch-and-cut algorithm for multicommodity capacitated fixed charge network design

**Mervat Chouman**[1] · **Teodor Gabriel Crainic**[2] · **Bernard Gendron**[3]

Received: 25 September 2015 / Accepted: 24 October 2017 / Published online: 13 November 2017


**Abstract** In this paper, we present a state-of-the-art branch-and-cut (B&C) algorithm for the multicommodity capacitated fixed charge network design problem (MCND). This algorithm combines bounding and branching procedures inspired by the latest developments in mixed-integer programming (MIP) software tools. Several filtering methods that exploit the structure of the MCND are also developed and embedded within the B&C algorithm. These filtering methods apply inference techniques to forbid combinations of values for some variables. This can take the form of adding cuts, reducing the domains of the variables, or fixing the values of the variables. Our experiments on a large set of randomly generated instances show that an appropriate selection of filtering techniques allows the B&C algorithm to perform better than the variant of the algorithm without filtering. These experiments also show that the B&C algorithm, with or without filtering, is competitive with a state-of-the-art MIP solver.

✉ Bernard Gendron
Bernard.Gendron@cirrelt.ca

Mervat Chouman
Mervat.Chouman@cirrelt.ca

Teodor Gabriel Crainic
TeodorGabriel.Crainic@cirrelt.net

1 College of Business, Effat University, Jeddah, Saudi Arabia

2 Department of Management and Technology, School of Management Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT), Université du Québec à Montréal, Montreal, QC, Canada

3 Département d'informatique et recherche opérationnelle Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT), Université de Montréal, Montreal, QC, Canada

**Keywords** Mixed-integer programming · Multicommodity capacitated fixed charge network design · Branch-and-cut · Filtering

**Mathematics Subject Classification** 90B10 · 90C11

## 1 Introduction

We present a branch-and-cut algorithm (B&C) to solve the multicommodity capacitated fixed charge network design problem (MCND), following the development of a specialized cutting-plane method described in Chouman et al. (2017). In this last paper, several valid inequalities, separation routines and modeling alternatives are presented and analyzed, the cutting-plane procedure being embedded within a state-of-the-art mixed-integer programming (MIP) solver. In the present paper, our aim is to develop a B&C method tailored for the problem that includes not only the cuts and separation routines from Chouman et al. (2017), but also *filtering* methods that exploit the structural properties of the problem. In general, such filtering methods apply inference techniques to forbid combinations of values for some variables, and proceed by adding cuts, reducing the domains of the variables, or fixing the values of the variables. Filtering methods are widely used in constraint programming (Hooker 2002), while in MIP, they arise within preprocessing routines (Savelsbergh 1994) and domain reduction tests based on reduced cost information. As such, filtering methods are an integral part of state-of-the-art MIP solvers (Atamtürk and Savelsbergh 2005).

To the best of our knowledge, along with the cutting-plane approach (Chouman et al. 2017) that constitutes the foundation for this work, the present paper is one of the few attempts at solving optimally the MCND, following earlier contributions based on Benders decomposition (Costa et al. 2009, 2012), column generation (Gendron and Larose 2014) and Lagrangian relaxation approaches (Crainic et al. 1999, 2001; Gendron and Crainic 1994; Holmberg and Yuan 2000; Kliewer and Timajev 2005; Sellmann et al. 2002). Heuristic methods have also been proposed for computing feasible solutions (Crainic et al. 2000, 2004; Crainic and Gendreau 2002; Ghamlouche et al. 2003, 2004; Hewitt et al. 2010; Katayama et al. 2009; Rodríguez-Martín and Salazar-González 2010). Typically, instances with few commodities (say, in the order of 10) can be solved to optimality in reasonable time by state-of-the-art MIP solvers, while instances with many commodities (more than 100) are very hard to solve to optimality (in Chouman et al. (2017), an average gap of 1.93% is reported for 57 difficult instances that are still unsolved after 2 hours of computing time). However, even for these instances, very good (often optimal) upper bounds are obtained by the cited heuristic methods. In our developments, we will therefore focus on the exact solution of these difficult large-scale instances, assuming that near-optimal solutions are readily available.

While a B&C algorithm is often the method of choice for the exact solution of network design problems similar to the MCND (Aardal 1998; Aardal et al. 1995; Atamtürk 2002; Atamtürk and Rajan 2002; Barahona 1996; Bienstock et al. 1998; Bienstock and Günlük 1996; Gabrel et al. 1999; Günlük 1999; Leung and Magnanti 1989; Magnanti et al. 1993, 1995; Ortega and Wolsey 2003; Raack et al. 2011), there

are no systematic studies regarding the behavior and performance of filtering methods in B&C algorithms for network design problems. Our main goal is to address this issue. We proceed by first presenting the basic features of the B&C algorithm we propose for the MCND, i.e., bounding and branching procedures inspired by Chouman et al. (2017) and by the latest developments in MIP software tools (Achterberg et al. 2005; Atamtürk and Savelsbergh 2005). We then develop a number of filtering methods that exploit the structure of the MCND and analyze their performance using the proposed B&C algorithm.

Our contributions are threefold:

- We develop a tailored B&C algorithm for the MCND. The implementation of this algorithm combines the cutting-plane method from Chouman et al. (2017) with an adaptation of the reliability branching rule introduced in Achterberg et al. (2005).
- We develop several filtering methods that are embedded within the B&C algorithm. These filtering methods are based either on duality arguments or on the detection of infeasible solutions. With the exception of the classical LP-based reduced cost fixing technique, they all exploit the structure of the MCND. Hence, to the best of our knowledge, state-of-the-art MIP solvers do not perform these filtering methods.
- By performing experiments on a set of 196 randomly generated instances used in other studies on the MCND, we show the efficiency and the effectiveness of both the B&C algorithm and the filtering methods. Specifically, our computational results illustrate that an appropriate selection of filtering techniques and their associated parameters provides notable improvements over the B&C algorithm without filtering. Furthermore, we also show that the B&C algorithm, with or without filtering, is competitive with a state-of-the-art MIP solver.

The paper is organized as follows. Section 2 presents the main features of the B&C algorithm, namely the valid inequalities and their separation routines, the cutting-plane procedure and the branching rule. Section 3 describes the filtering methods, while Sect. 4 summarizes the overall B&C algorithm. In Sect. 5, we present the results of extensive computational experiments on a large set of instances. Section 6 summarizes our findings and discusses avenues for future research.

## 2 Main features of the branch-and-cut algorithm

We describe the MCND and the formulation used within the B&C algorithm in Sect. 2.1. In Sect. 2.2, we present the valid inequalities and the separation routines performed at every node of the B&C tree by the cutting-plane procedure. The latter is summarized in Sect. 2.3, while the branching rule used in the B&C algorithm is described in Sect. 2.4.

### 2.1 Problem formulation

Given a directed network $G = (V, A)$, with $V$ the set of nodes and $A$ the set of arcs, we let $K$ be the set of commodities, each commodity $k$ having one origin, $O(k)$, and one destination, $D(k)$, with a demand $d^k > 0$ between the two nodes. We associate

with each arc $(i, j)$ the per unit routing cost $c_{ij} \geq 0$, the fixed cost $f_{ij} \geq 0$, and the capacity $u_{ij} > 0$. We assume that capacities and demands take integer values. Two types of variables are used to formulate the MCND: the continuous flow variable $x_{ij}^k$, which represents the flow of commodity $k$ on arc $(i, j)$, and the binary design variable $y_{ij}$, which equals 1 when arc $(i, j)$ is used, and 0, otherwise. Given these definitions, the MCND can be formulated as follows:

$$Z = \min \sum_{k \in K} \sum_{(i,j) \in A} c_{ij} x_{ij}^k + \sum_{(i,j) \in A} f_{ij} y_{ij} \tag{1}$$

$$\sum_{j \in V_i^+} x_{ij}^k - \sum_{j \in V_i^-} x_{ji}^k = \begin{cases} d^k, & \text{if } i = O(k), \\ -d^k, & \text{if } i = D(k), \\ 0, & \text{otherwise,} \end{cases} \quad i \in V, k \in K, \tag{2}$$

$$\sum_{k \in K} x_{ij}^k \leq u_{ij} y_{ij}, \quad (i, j) \in A, \tag{3}$$

$$0 \leq x_{ij}^k \leq b_{ij}^k, \quad (i, j) \in A, k \in K, \tag{4}$$

$$y_{ij} \in \{0, 1\}, \quad (i, j) \in A, \tag{5}$$

where $b_{ij}^k = \min\{u_{ij}, d^k\}$, $V_i^+ = \{j \in V | (i, j) \in A\}$ and $V_i^- = \{j \in V | (j, i) \in A\}$. Constraints (2) represent the flow conservation equations for each node and each commodity. Relations (3) ensure that the flow on each arc does not exceed its capacity; they also play the role of forcing constraints, since they ensure that no flow is allowed on an arc unless the fixed cost on the arc is incurred. Constraints (4) and (5) define the domains of the flow and design variables, respectively. Note that $b_{ij}^k$ can be any valid upper bound on the amount of flow of commodity $k$ on arc $(i, j)$. The model can thus integrate commodity-dependent capacities, although we only assume a capacity $u_{ij}$ on each arc $(i, j)$ that binds the flow of all commodities on the arc. Similarly, we assume that the routing costs do not depend on the commodities, although it would be easy to handle commodity-dependent costs in our model.

To characterize the status of the binary design variables at each node of the B&C tree, $A_1$ and $A_0$ denote the sets of *open and closed arcs*, respectively, i.e., the arcs fixed to 1 and to 0 by branching and variable fixing; $A_{01} = A \setminus (A_1 \cup A_0)$ denotes the set of *free arcs*. The restricted problem considered at each node then consists of model (1)–(5) to which we add the constraints $y_{ij} = 0$, $(i, j) \in A_0$, and $y_{ij} = 1$, $(i, j) \in A_1$. The cutting-plane procedure strengthens the linear programming (LP) relaxation of this restricted problem by adding inequalities that are valid for model (1)–(5), but violated by the solution of the current LP relaxation. These inequalities are presented next.

### 2.2 Valid inequalities and separation

Our cutting-plane procedure exploits the valid inequalities that are shown to be the most useful in Chouman et al. (2017). We use two classes of valid inequalities, the strong and knapsack inequalities, which are described in the next subsections, along with their respective separation algorithms. Chouman et al. (2017) also use flow cover/pack

inequalities (Atamtürk 2001; Gu et al. 1999b; Louveaux and Wolsey 2007; Padberg et al. 1985; Roy and Wolsey 1987). Although these inequalities are effective in improving the lower bounds, they provide similar bound improvements, on most instances, than the combination of strong and knapsack inequalities. Since their separation is significantly more expensive computationally, we have decided not to use them in our cutting-plane procedure.

#### *2.2.1 Strong inequalities*

The following inequalities, in a similar way as constraints (3), play the role of forcing constraints, since they also forbid any flow to circulate on an arc that is not part of the selected design:

$$x_{ij}^k \leq b_{ij}^k y_{ij}, \quad (i,j) \in A, k \in K. \tag{6}$$

Adding these so-called *strong inequalities* to the model significantly improves the quality of the LP relaxation lower bound (Chouman et al. 2017; Crainic et al. 1999; Gendron and Crainic 1994). Adding *a priori* all these inequalities to the LP relaxation yields very large models that frequently exhibit degeneracy. We add them in a dynamic way, identifying only those that are violated by the solution of the current LP relaxation. Their separation is trivial, as it suffices to scan each arc and each commodity to identify all violated inequalities.

#### *2.2.2 Knapsack inequalities*

Assuming $S \subset V$ is a non-empty subset of $V$ and $\bar{S} = V \backslash S$ is its complement, we note the corresponding cutset $(S, \bar{S}) = \{(i,j) \in A \mid i \in S, j \in \bar{S}\}$ and its associated commodity subset $K(S, \bar{S}) = \{k \in K \mid O(k) \in S, D(k) \in \bar{S}\}$. We then have the following valid inequality, which is obtained by combining the flow conservation equations (2) with the capacity constraints (3):

$$\sum_{(i,j)\in(S,\bar{S})} u_{ij} y_{ij} \geq d_{(S,\bar{S})}, \tag{7}$$

where $d_{(S,\bar{S})} = \sum_{k\in K(S,\bar{S})} d^k$. This inequality simply states that there should be enough capacity on the arcs of the cutset $(S, \bar{S})$ to satisfy the total demand that must flow from $S$ to $\bar{S}$. By complementing the $y$ variables, i.e., replacing $y_{ij}$ by $1 - y_{ij}$, the cutset inequality reduces to a 0-1 knapsack structure.

The well-known cover inequalities for the 0-1 knapsack structure (Balas 1975; Hammer et al. 1975; Wolsey 1975) are based on the following definition: $C \subseteq (S, \bar{S})$ is a *cover* if the total capacity of the arcs in $(S, \bar{S})\backslash C$ does not cover the demand, i.e., $\sum_{(i,j)\in(S,\bar{S})\backslash C} u_{ij} < d_{(S,\bar{S})}$. For every cover $C \subseteq (S, \bar{S})$, the following *cover inequality* is valid for the MCND:

$$\sum_{(i,j)\in C} y_{ij} \geq 1. \tag{8}$$

In addition to the cover inequalities, we use the so-called minimum cardinality inequalities Martello and Toth (1997). To define these inequalities, we assume the capacities of the arcs in $(S, \bar{S})$ are sorted in non-increasing order: $u_{a(t)} \geq u_{a(t+1)}$, where $a(t) \in (S, \bar{S})$, $t = 1, ..., |(S, \bar{S})|$ $(u_{a(t+1)} = u_{a(t)})$. This allows us to compute the least number of arcs in $(S, \bar{S})$ that must be opened in any feasible solution: $l_{(S,\bar{S})} = \max \{h \mid \sum_{t=1,...,h} u_{a(t)} < d_{(S,\bar{S})}\} + 1$. We then derive the *minimum cardinality inequality*:

$$\sum_{(i,j)\in(S,\bar{S})} y_{ij} \geq l_{(S,\bar{S})}. \tag{9}$$

The generation of knapsack inequalities is based on single-node cutsets, i.e., for each cutset $(S, \bar{S})$, $S$ is an origin or $\bar{S}$ is a destination for at least one commodity. Methods to generate cutsets $(S, \bar{S})$ with $|S| > 1$ are developed and tested in Chouman et al. (2017), where it is observed that, for most instances, the single-node cutsets are responsible for most of the lower bound improvement.

For each single-node cutset, we try to generate one violated cover inequality and one violated minimum cardinality inequality. Initially, some $y$ variables are fixed to either 0 or 1, using the LP relaxation solution. Two different variable fixing strategies are used, depending on the type of inequality we try to generate, cover or minimum cardinality (details can be found in Chouman et al. (2017)). Thus, we obtain in this way two restricted cutsets, one that is used to derive a cover inequality, the other to generate a minimum cardinality inequality. The cover inequality is obtained by the separation routine described in Chouman et al. (2017); Gu et al. (1998, 1999a). To generate the minimum cardinality inequality, we simply sort the arcs in the corresponding restricted cutset and then derive the minimum number of arcs to be opened. For each of the two inequalities thus obtained, a sequential lifting procedure is applied to obtain an inequality that is valid for the original cutset, and therefore also for the MCND. The same lifting procedure is used for the two inequalities, cover and minimum cardinality. If any of the resulting valid inequalities is violated by the solution of the current LP relaxation solution, it is added to the LP relaxation.

### 2.3 Cutting-plane procedure

As explained above, the cutting-plane procedure is a simpler variant of the method described in Chouman et al. (2017), since it generates strong and knapsack inequalities, but no flow cover/pack inequalities. At each node of the B&C tree, it starts by solving the LP relaxation of the current formulation, defined by the current status of the arcs, open, closed or free, and by the cuts added so far. Subsequently, it alternates between the generation of cuts and the solution of the current LP relaxation.

The cutting-plane procedure performs the following steps, where $Z^*$ is the objective value of the best known feasible solution and $\delta$ is a parameter that measures the minimum bound improvement between two consecutive LPs that is required to continue the procedure (we use $\delta = 0.1$ as in Chouman et al. (2017)):

1. $Z^l_{last} \leftarrow 0$.
2. Solve the LP relaxation; let $Z^l$ be the LP optimal value ($Z^l = +\infty$ if the LP is infeasible), and $\bar{y}$ the LP design solution.

3. If $\bar{y}$ is integral or $Z^l \geq Z^*$ or $Z^l - Z^l_{last} \leq \delta$, then stop.
4. Try to generate cuts.
5. If some cuts are found, then $Z^l_{last} \leftarrow Z^l$ and go to 2.

The B&C algorithm manages two types of cuts: *global cuts*, which are valid at any node of the B&C tree, and *local cuts*, which are valid only at the current node and at all its descendants. When a node is handled immediately after its parent, the LP relaxation is simply reoptimized after taking into account the additions made by branching and filtering. When a node is obtained from backtracking in the B&C tree, the LP relaxation is built by considering the LP solution from its parent and by adding global and local cuts violated by this solution.

The strong inequalities are generated at all nodes and managed as global cuts. The knapsack inequalities are generated only at the root node and are therefore managed as global cuts. Other global and local cuts are generated by the filtering methods described in Sect. 3.

### 2.4 Branching rule

When branching is performed, the set of free arcs with fractional $\bar{y}$ values, denoted $\bar{A}_{01}$, is non-empty, i.e., $\bar{A}_{01} = \{(i, j) \in A_{01} | 0 < \bar{y}_{ij} < 1\} \neq \emptyset$. In a classical way, the branching rule selects one arc from this set, say $a^* \in \bar{A}_{01}$, and generates the 0-child and the 1-child defined by removing $a^*$ from $A_{01}$ and by adding it to $A_0$ and to $A_1$, respectively. To select $a^*$, we use a variant of *reliability branching* (Achterberg et al. 2005), a rule that combines the strengths of two other branching rules, pseudo-cost branching and strong branching.

To define these different branching rules, we use the following notation. When branching on an arc $a$, we define the increase in the LP bounds from the parent node to the 0-child and the 1-child as $\Delta^0_a$ and $\Delta^1_a$, respectively. We also define the corresponding *per unit* increase in the LP bounds from the parent to its children as follows: $\rho^h_a = \frac{\Delta^h_a}{g^h_a}$, $h = 0, 1$, where $g^0_a = \bar{y}_a$ and $g^1_a = 1 - \bar{y}_a$. Assume that, after branching on arc $a$, the increase in the LP bounds from the parent node to the 0-child and the 1-child have been computed $n^0_a$ and $n^1_a$ times; we can then define the *average per unit* increase in the LP bounds from the parent node to its children as $\bar{\rho}^h_a$, $h = 0, 1$ (i.e., the average value of $\rho^h_a$ over the $n^h_a$ times arc $a$ has been selected for branching and the increase in the LP bound from the parent to its $h$-child has been computed).

Pseudo-cost branching (Benichou et al. 1971) is based on computing and storing the values $\bar{\rho}^h_a$, $h = 0, 1$, for each arc $a$. This branching rule selects the free arc $a^*$ such that

$$a^* = \arg \max_{a \in \bar{A}_{01}} \{\min(g^0_a \bar{\rho}^0_a, g^1_a \bar{\rho}^1_a)\}.$$

In this formula, $g^h_a \bar{\rho}^h_a$, $h = 0, 1$, represent estimates of the increase in the LP bounds from the current node to the children that would be obtained by selecting arc $a$ for branching. Initially, no values of LP bound increases, i.e., $\Delta^h_a$, $h = 0, 1$, are available;

hence, we simply set $\bar{\rho}_a^h = 1$, $h = 0, 1$, for each arc $a$. The selected arc is then the one with the most fractional value $\bar{y}_a$, i.e., with $\bar{y}_a$ closest to 0.5.

An alternative is strong branching (Applegate et al. 1995), which is based on computing estimates of the LP bound increases $\Delta_a^h$, $h = 0, 1$, prior to branching. This rule amounts to look at the effect of selecting arc $a$ by adding to the current LP relaxation the constraints $y_a = 0$ and $y_a = 1$ in order to evaluate $\Delta_a^0$ and $\Delta_a^1$, respectively. This is performed by reoptimizing the current LP relaxation with the added constraint through a few iterations of the dual simplex method. The strong branching rule then selects the free arc $a^*$ such that

$$a^* = \arg \max_{a \in \bar{A}_{01}} \{\min(\Delta_a^0, \Delta_a^1)\}.$$

The idea behind reliability branching (Achterberg et al. 2005) is to perform strong branching at the beginning of the exploration to obtain reliable LP bound increase estimates and then to switch to pseudo-cost branching for the rest of the exploration. More precisely, assuming a free arc $a^*$ is selected by the pseudo-cost branching rule, if $\min(n_{a^*}^0, n_{a^*}^1) < \eta$, where $\eta \geq 0$ is a parameter, then the pseudo-costs associated with arc $a^*$ are considered unreliable, and the pseudo-cost estimates are replaced by the strong branching estimates of LP bound increases. When $\eta = 0$, reliability branching reduces to pseudo-cost branching, while if $\eta = +\infty$, reliability branching reduces to strong branching.

Our implementation of reliability branching works as follows. We first select the free arc $a^* \in \bar{A}_{01}$ according to the pseudo-cost branching rule. If $a^*$ is not reliable, i.e., $\min(n_{a^*}^0, n_{a^*}^1) < \eta$, then the arcs $a \in \bar{A}_{01}$ with unreliable pseudo-costs are sorted in non-increasing order of $\min(g_a^0 \bar{\rho}_a^0, g_a^1 \bar{\rho}_a^1)$. Using that particular order, we keep as a candidate for branching the arc $a^*$ that achieves so far the maximum of $\min(\Delta_a^0, \Delta_a^1)$, where $\Delta_a^h$, $h = 0, 1$, are computed with the dual simplex method (limited to 100 iterations). If that candidate is not updated for $\lambda$ successive attempts, we select $a^*$ for branching. The branching procedure thus performs the following steps (we use $\eta = 8$ and $\lambda = 4$ as in Achterberg et al. (2005)):

1. $a^* \leftarrow \arg\max_{a \in \bar{A}_{01}} \{\min(g_a^0 \bar{\rho}_a^0, g_a^1 \bar{\rho}_a^1)\}$.
2. If $\min(n_{a^*}^0, n_{a^*}^1) \geq \eta$, then stop.
3. Let $m \leftarrow 0$, $s^* \leftarrow 0$ and sort the arcs of $(\bar{A}_{01} \cap \{a \in A | \min(n_a^0, n_a^1) < \eta\})$ in non-increasing order of $\min(g_a^0 \bar{\rho}_a^0, g_a^1 \bar{\rho}_a^1)$.
4. For all $a \in (\bar{A}_{01} \cap \{a \in A | \min(n_a^0, n_a^1) < \eta\})$ (sorted):
   (a) $m \leftarrow m + 1$.
   (b) Compute $\Delta_a^0$ and update $n_a^0$ and $\bar{\rho}_a^0$ (unless the LP is infeasible); if $Z^l + \Delta_a^0 \geq Z^*$, then fix arc $a$ to value 1, i.e., transfer $a$ from $A_{01}$ to $A_1$.
   (c) If arc $a$ has not been fixed to value 1 in the previous step, then compute $\Delta_a^1$ and update $n_a^1$ and $\bar{\rho}_a^1$; if $Z^l + \Delta_a^1 \geq Z^*$, then fix arc $a$ to value 0, i.e., transfer $a$ from $A_{01}$ to $A_0$.
   (d) If arc $a$ has not been fixed to value 0 or 1 in the previous steps and if $\min(\Delta_a^0, \Delta_a^1) > s^*$, then $a^* \leftarrow a$, $s^* \leftarrow \min(\Delta_a^0, \Delta_a^1)$ and $m \leftarrow 0$.
   (e) If $m \geq \lambda$, then stop.

Note that a filtering method is already embedded into the strong branching loop in steps (4b) and (4c). In both steps, we use the fact that $Z^l + \Delta_a^h$ is a lower bound on the restriction of the MCND defined by $y_a = h$, $h = 0, 1$. Therefore, if this lower bound exceeds the best known upper $Z^*$ on the optimal value of the MCND, we can fix variable $y_a$ to the value $1 - h$. We see other examples of similar filtering methods in the next section.

## 3 Filtering methods

Filtering methods are applied at every node of the B&C tree. The general idea is to exclude solutions that cannot be optimal, given the current status of the design variables, i.e., the partition of the set of arcs into $A_0$, $A_1$ and $A_{01}$. The solutions are excluded through the addition of cuts that are generally local (i.e., valid only for the node and its descendants), but that can be global in some cases. Special types of cuts are worth noting: *bound reduction* consists in decreasing (increasing) the upper (lower) bound on a single variable, while *variable fixing*, a special case of bound reduction, assigns a value to a single variable (such cuts are heavily used in the field of constraint programming).

A common approach in filtering methods is to deduce from the addition of a constraint $\mathcal{C}$ the impossibility of finding an optimal solution that satisfies simultaneously $\mathcal{C}$ and the constraints that define the current B&C node. Hence, constraint $\neg\mathcal{C}$, the complement of $\mathcal{C}$, can be added to cut all solutions that satisfy $\mathcal{C}$. To infer that the addition of $\mathcal{C}$ cannot lead to an optimal solution, we generally compute a lower bound $Z^l(\mathcal{C})$ on the optimal value of the restricted problem derived from the addition of $\mathcal{C}$. If $Z^l(\mathcal{C}) \geq Z^*$, where $Z^*$ is the value of the best known feasible solution, we can conclude that no optimal solution can be found when constraint $\mathcal{C}$ is added. A particular case of this test arises when we can deduce that no feasible solution can be obtained when $\mathcal{C}$ is added, since this case can be reduced to $Z^l(\mathcal{C}) = +\infty$.

Thus, to perform efficient and effective filtering methods, we: (1) derive lower bounds that are quickly computed based on duality arguments; (2) investigate sources of infeasibility to try to detect them as early as possible when exploring the B&C tree. The next three sections are dedicated to duality-based filtering techniques: the LP-based reduced cost fixing, the Lagrangian-based reduced cost fixing and the reduced cost bound reduction, which are presented in Sects. 3.1, 3.2, and 3.3, respectively. Then, we describe three feasibility-based filtering techniques: the generation of combinatorial Benders cuts, the connectivity-based filtering procedure, and the capacity-based filtering methods, which are presented in Sects. 3.4, 3.5, and 3.6, respectively.

### 3.1 LP-based reduced cost fixing

The reduced costs $\overline{f}_{ij}$ derived from the LP relaxation can be used to perform variable fixing. Indeed, for each non-basic variable $y_{ij}$ at value $\overline{y}_{ij} \in \{0, 1\}$ and such that $(i, j) \in A_{01}$, we have $\overline{f}_{ij} \leq 0$ if $\overline{y}_{ij} = 1$, and $\overline{f}_{ij} \geq 0$ if $\overline{y}_{ij} = 0$. If we add the constraint $y_{ij} = (1 - \overline{y}_{ij})$, then $Z^l + |\overline{f}_{ij}|$ is a lower bound on the optimal value of the resulting problem, using standard LP duality theory. Therefore, if $Z^l + |\overline{f}_{ij}| \geq$

$Z^*$, then we can fix $y_{ij}$ to value $\overline{y}_{ij}$. These tests are carried out immediately after performing the cutting-plane procedure by scanning all non-basic design variables. This filtering technique is common to all general purpose LP-based B&C algorithms and is performed by state-of-the-art MIP solvers. The next filtering method, however, exploits the particular structure of the MCND.

### 3.2 Lagrangian-based reduced cost fixing

At any node of the B&C tree, characterized by the sets $A_0$, $A_1$ and $A_{01}$, we consider the Lagrangian relaxation of the flow conservation equations, known as the *knapsack relaxation* (Gendron and Crainic 1994). Our objective is to use reduced costs derived from this Lagrangian relaxation to perform variable fixing, with the potential of delivering results that are different than those obtained when performing LP-based reduced cost fixing. More precisely, we consider the Lagrangian relaxation with respect to the formulation restricted by $A_0$, $A_1$, $A_{01}$, and defined by (1)–(5), plus the strong inequalities (6). Denoting $\pi = (\pi_i^k)_{i\in V}^{k\in K}$ the vector of Lagrange multipliers associated with the flow conservation equations, we then obtain the following Lagrangian subproblem:

$$
\begin{aligned}
Z_{LR}^l = & \sum_{k\in K}(\pi_{D(k)}^k - \pi_{O(k)}^k)d^k \\
& + \min \sum_{(i,j)\in A_{01}\cup A_1} \left\{ \sum_{k\in K}(c_{ij} + \pi_i^k - \pi_j^k)x_{ij}^k + f_{ij}y_{ij} \right\} \\
\sum_{k\in K} x_{ij}^k \le\ & u_{ij}y_{ij}, \quad (i,j) \in A_{01}\cup A_1, \\
0 \le\ & x_{ij}^k \le b_{ij}^k y_{ij}, \quad (i,j) \in A_{01}\cup A_1, k \in K, \\
y_{ij} =\ & 1, \quad (i,j) \in A_1, \\
y_{ij} \in\ & \{0,1\}, \quad (i,j) \in A_{01}.
\end{aligned}
$$

This problem can be solved by first considering, for each arc $(i, j) \in A_{01} \cup A_1$, the following continuous knapsack problem:

$$
v_{ij} = \min \left\{ \sum_{k\in K} \tilde{c}_{ij}^k x_{ij}^k \mid \sum_{k\in K} x_{ij}^k \le u_{ij};\ 0 \le x_{ij}^k \le b_{ij}^k,\ k \in K \right\},
$$

where $\tilde{c}_{ij}^k = c_{ij} + \pi_i^k - \pi_j^k$, $k \in K$. Indeed, it is easy to show that the Lagrangian subproblem can be reformulated as follows:

$$
Z_{LR}^l = \sum_{k\in K}(\pi_{D(k)}^k - \pi_{O(k)}^k)d^k + \sum_{(i,j)\in A_1} \tilde{f}_{ij} + \sum_{(i,j)\in A_{01}} \min\{\tilde{f}_{ij}y_{ij} \mid y_{ij} \in \{0,1\}\},
$$

where $\tilde{f}_{ij} = v_{ij} + f_{ij}$, $(i, j) \in A_{01} \cup A_1$. An optimal solution to the subproblem for each arc $(i, j) \in A_{01}$ is given by $\tilde{y}_{ij} = 1$, if $\tilde{f}_{ij} < 0$, and $\tilde{y}_{ij} = 0$, otherwise.

Clearly, $Z^l_{LR}$ is a lower bound on the optimal value at the current node. Therefore, if $Z^l_{LR} \geq Z^*$, the current node can be fathomed. Furthermore, it is easy to derive variable fixing rules by using the quantity $\tilde{f}_{ij}$, which can be interpreted as a Lagrangian reduced cost associated with $y_{ij}$. Indeed, for each $(i, j) \in A_{01}$, it is immediate to see that $Z^l_{LR} + |\tilde{f}_{ij}|$ is a lower bound on the restricted problem obtained by adding the constraint $y_{ij} = 1 - \tilde{y}_{ij}$. Consequently, if $Z^l_{LR} + |\tilde{f}_{ij}| \geq Z^*$, then we can fix $y_{ij}$ to value $\tilde{y}_{ij}$.

The Lagrangian subproblem is solved after performing LP-based reduced cost fixing. The Lagrange multipliers are fixed to the values of the dual variables associated with the flow conservation equations that are obtained after performing the cutting-plane procedure. Note that the knapsack relaxation has been used to compute lower bounds in branch-and-bound algorithms for the MCND (Holmberg and Yuan 2000; Kliewer and Timajev 2005; Sellmann et al. 2002), where non-differentiable optimization, i.e., subgradient and bundle, methods were used to compute near-optimal Lagrange multipliers. The difference here is that we use the knapsack relaxation only to improve filtering at each node of the B&C tree and thus as a complement to the cutting-plane procedure, rather than as the main lower bounding method.

### 3.3 Flow upper bound reduction

We can use the LP-based reduced costs of the flow variables $x^k_{ij}, \overline{c}^k_{ij}$, to perform bound reduction on these variables. The basic idea is the following: assume we add the constraint $x^k_{ij} > a^k_{ij}$ to the LP relaxation and that the resulting lower bound exceeds $Z^*$. We can then conclude that the constraint $x^k_{ij} \leq a^k_{ij}$ is valid. In order to compute $a^k_{ij}$, we use the following result.

**Proposition 1** *Let $\overline{x}^k_{ij}$ be the value of variable $x^k_{ij}$ in the optimal solution to the LP relaxation. If $\overline{x}^k_{ij} = 0$, $\overline{c}^k_{ij} > 0$ and $Z^l + \overline{f}_{ij}(1 - \overline{y}_{ij}) + \overline{c}^k_{ij} b^k_{ij} > Z^*$, we have $x^k_{ij} \leq a^k_{ij} < b^k_{ij}$, where*

$$a^k_{ij} = \frac{Z^* - Z^l - \overline{f}_{ij}(1 - \overline{y}_{ij})}{\overline{c}^k_{ij}}.$$

*Proof* We consider two cases. First, let us assume that $0 < \overline{y}_{ij} \leq 1$, which implies $\overline{f}_{ij}(1 - \overline{y}_{ij}) = 0$. We note that, if $0 < \overline{x}^k_{ij} < b^k_{ij}$, then $\overline{c}^k_{ij} = 0$ and, in this case, the LP relaxation lower bound remains the same when we increase $x^k_{ij}$ further. Therefore, for the LP relaxation lower bound to increase and exceed $Z^*$ when we add the constraint $x^k_{ij} > a^k_{ij}$, we must have $\overline{x}^k_{ij} = 0$, $\overline{c}^k_{ij} > 0$ and $Z^l + \overline{c}^k_{ij} b^k_{ij} > Z^*$. Since any optimal solution must satisfy $Z^l + \overline{c}^k_{ij} x^k_{ij} \leq Z^*$, we conclude that $x^k_{ij} \leq a^k_{ij} < b^k_{ij}$, where

$$a^k_{ij} = \frac{Z^* - Z^l}{\overline{c}^k_{ij}}.$$

Next, we consider the case where $\overline{y}_{ij} = 0$. Then, we necessarily have $\overline{x}^k_{ij} = 0$ and $\overline{f}_{ij} \geq 0$. This means that, if we add the constraint $x^k_{ij} > a^k_{ij}$, the LP relaxation lower bound will exceed $Z^*$ only if $Z^l + \overline{f}_{ij} + \overline{c}^k_{ij} b^k_{ij} > Z^u$. Since any optimal solution must satisfy $Z^l + \overline{f}_{ij} + \overline{c}^k_{ij} x^k_{ij} \leq Z^*$ and assuming $\overline{c}^k_{ij} > 0$, we have $x^k_{ij} \leq a^k_{ij} < b^k_{ij}$, where

$$a^k_{ij} = \frac{Z^* - Z^l - \overline{f}_{ij}}{\overline{c}^k_{ij}}.$$

□

Similarly, we can use the solution to the knapsack relaxation to reduce the upper bounds on the flow variables, as shown in the following Proposition, the proof of which is omitted, as it is similar to that of Proposition 1.

**Proposition 2** *Let $\tilde{x}^k_{ij}$ be the value of variable $x^k_{ij}$ in the optimal solution to the Lagrangian subproblem. If $\tilde{x}^k_{ij} = 0$, $\tilde{c}^k_{ij} > 0$ and $Z^l_{LR} + \tilde{f}_{ij}(1 - \tilde{y}_{ij}) + \tilde{c}^k_{ij} b^k_{ij} > Z^*$, we have $x^k_{ij} \leq a^k_{ij} < b^k_{ij}$, where*

$$a^k_{ij} = \frac{Z^* - Z^l_{LR} - \tilde{f}_{ij}(1 - \tilde{y}_{ij})}{\tilde{c}^k_{ij}}.$$

We use these results as follows. After performing LP-based reduced cost fixing, we look for flow variables $x^k_{ij}$ that verify the condition for reducing their upper bound $b^k_{ij}$ to $a^k_{ij}$. We do the same after carrying out Lagrangian-based reduced cost fixing. New upper bounds are then used at the current node and all its descendants. The upper bounds are stored at each node, in order to initialize them for the child node that is activated when backtracking is performed. In this way, when looking for violated strong inequalities in the cutting-plane procedure, we use the local cuts $x^k_{ij} \leq a^k_{ij} y_{ij}$ instead of $x^k_{ij} \leq b^k_{ij} y_{ij}$.

In the next section, we derive another type of cuts, this time by investigating the structure of feasible solutions. Contrary to the cuts obtained by flow upper bound reduction, these cuts are global, i.e., they apply at every node of the B&C tree.

### 3.4 Combinatorial Benders cuts

At every node of the B&C tree, feasible solutions must satisfy the following *multi-commodity flow system*, noted $MF$:

$$\sum_{j \in V_i^+} x^k_{ij} - \sum_{j \in V_i^-} x^k_{ji} = \begin{cases} d^k, & \text{if } i = O(k), \\ -d^k, & \text{if } i = D(k), \\ 0, & \text{otherwise,} \end{cases} \quad i \in V, k \in K, \tag{10}$$

$$\sum_{k \in K} x^k_{ij} \leq u_{ij}, \quad (i, j) \in A_{01} \cup A_1, \tag{11}$$

$$\sum_{k \in K} x_{ij}^k = 0, \quad (i, j) \in A_0, \tag{12}$$

$$x_{ij}^k \geq 0, \quad (i, j) \in A, k \in K. \tag{13}$$

In particular, any feasible solution generated by the cutting-plane procedure at the current node satisfies this system. We exploit the structure of $MF$ in two ways.

First, we note that when $MF$ is infeasible, we can derive a cut that prevents this infeasible design configuration, i.e., subset $A_0$, to be generated again; this cut is generated whenever the cutting-plane procedure returns an infeasible LP relaxation (it is also generated in capacity-based filtering, see Sect. 3.6).

**Proposition 3** *If $MF$ is infeasible, then the following inequality is valid for the MCND:*

$$\sum_{(i,j) \in A_0} y_{ij} \geq 1. \tag{14}$$

Inequality (14) is a *combinatorial Benders cut* (Codato and Fischetti 2006), which has the general form $\sum_{(i,j) \in A_1} (1 - y_{ij}) + \sum_{(i,j) \in A_0} y_{ij} \geq 1$ that can be strengthened to (14), since only closed arcs can induce an infeasible subproblem. A different inequality can be derived from LP duality arguments, giving rise to classical Benders cuts, which have been studied for the MCND Costa et al. (2009, 2012). In our B&C algorithm, combinatorial Benders cuts are stored as global cuts. Their violation is verified after the cutting-plane procedure has been completed. In case violated combinatorial Benders cuts are found, the cutting-plane procedure is restarted.

At each node of the B&C tree, combinatorial Benders cuts are also used in simple operations that attempt to detect infeasibility just before calling the cutting-plane procedure. These node-based preprocessing operations work as follows. Assuming the current design configuration is given by $A'_0$, $A'_1$ and $A'_{01}$, we define the design vector $y'$ as $y'_{ij} = 0$, if $(i, j) \in A'_0$, and $y'_{ij} = 1$, otherwise. We scan the set of combinatorial Benders cuts generated so far and for each of them, associated with a set $A_0$, we verify: (1) if $\sum_{(i,j) \in A_0} y'_{ij} < 1$, which is equivalent to the condition $A_0 \subseteq A'_0$, in which case the node can be fathomed; (2) if $\sum_{(i,j) \in A_0} y'_{ij} = 1$, which is equivalent to the condition $|A_0 \cap A'_{01}| \leq 1$, in which case if all the arcs in $A_0$ are fixed to 0, except one, then that arc must be fixed to 1. Similar node-based preprocessing operations can be applied to lifted knapsack inequalities generated by the cutting-plane procedure and stored in the global cut pool (the details are obvious and therefore omitted).

A second approach to exploiting the structure of $MF$ consists in developing filtering methods aiming to detect as early as possible any infeasibility that might occur as a result of closing too many arcs. Two sources of infeasibility can be identified: first, for some commodity $k$, there is no longer any path connecting $O(k)$ to $D(k)$, which gives rise to connectivity-based filtering; second, the overall capacity is not sufficient to satisfy the demand for at least one commodity, which yields capacity-based filtering.

### 3.5 Connectivity-based filtering

As mentioned above, the current node can be fathomed when we can identify at least one commodity $k$ such that there is no path between $O(k)$ and $D(k)$. In addition to detecting this type of infeasibility prior to the call to the cutting-plane procedure, we also fix flow and design variables based on simple connectivity tests. Indeed, when, for some commodity $k$, an arc $(i, j)$ does not belong to any path between $O(k)$ and $D(k)$, the upper bound $b_{ij}^k$ associated with variable $x_{ij}^k$ can be fixed to 0. Similarly, when, for some commodity $k$, an arc $(i, j)$ belongs to all paths between $O(k)$ and $D(k)$, the lower bound associated with variable $x_{ij}^k$ can be fixed to $d^k$ (unless $b_{ij}^k < d^k$, in which case the current node can be fathomed; this case can happen as a result of flow upper bound reduction that can decrease the upper bound $b_{ij}^k$, see Sect. 3.3). In addition, an arc $(i, j)$ can be closed when it does not belong to any path between $O(k)$ and $D(k)$ for all commodities $k$. Conversely, an arc $(i, j)$ can be opened when it belongs to all paths between $O(k)$ and $D(k)$ for at least one commodity $k$. This last test prevents the occurrence of infeasible subproblems due to a lack of connectivity.

These tests can be easily performed using graph-traversal algorithms. Indeed, to every node $i \in V$, we associate the commodity subsets $K_i^+ = \{k \in K \mid i = O(k)\}$ and $K_i^- = \{k \in K \mid i = D(k)\}$. Starting from every node $i$, we perform complete forward and backward traversals of the graph. Each arc $a \in A_{01} \cup A_1$ has two sets of labels $p_a^k$ and $m_a^k$, for each commodity $k$; each label is initialized with value 0. Whenever we encounter an arc $a$ during the forward traversal from node $i$, we set the label $p_a^k$ of each commodity $k$ in $K_i^+$ to value 1. Likewise, when performing the backward traversal starting at node $i$, the label $m_a^k$ of each arc $a$ for commodity $k \in K_i^-$ is set to value 1. After completing forward and backward traversals (each being performed in linear time) for all nodes, a final pass through all arcs is performed. For each arc $a \in A_{01} \cup A_1$ and commodity $k \in K$, two cases can happen: (1) $p_a^k = m_a^k = 1$, in which case arc $a$ belongs to some path between $O(k)$ and $D(k)$; (2) $p_a^k = 0$ or $m_a^k = 0$, which implies that arc $a$ does not belong to any path between $O(k)$ and $D(k)$. This information suffices to perform the fathoming and filtering tests outlined above. In particular, to determine that an arc $a = (i, j)$ belongs to all paths between $O(k)$ and $D(k)$ for commodity $k$, $(i, j)$ must satisfy case (1), while any other outgoing arc from $i$ must verify case (2).

Connectivity-based filtering is called only when some arcs have been closed since the last time it was performed. It is also performed at the root node of the B&C tree in order to simplify the problem instance.

### 3.6 Capacity-based filtering

This filtering method solves the following linear program, denoted $MC$ and obtained from system $MF$:

$$Z_{MC}^l = \sum_{(i,j)\in A_1} f_{ij} + \min \sum_{k\in K} \left\{ \sum_{(i,j)\in A_{01}} (c_{ij} + f_{ij}/u_{ij}) x_{ij}^k + \sum_{(i,j)\in A_1} c_{ij} x_{ij}^k \right\} \quad (15)$$

subject to constraints (10) to (13). This linear multicommodity flow problem is a relaxation of any LP generated during the cutting-plane procedure. Indeed, it is equivalent to the LP relaxation of the MCND without any strong or knapsack inequality added, the so-called *weak relaxation* (Gendron and Crainic 1994). To see why, simply note that each design variable $y_{ij}$ appears in only one capacity constraint in the weak relaxation. As a consequence, since $f_{ij} \geq 0$, there must be an optimal solution such that $y_{ij} = \sum_{k \in K} x_{ij}^k / u_{ij}$ for each arc $(i, j) \in A_{01}$. By substituting $y_{ij}$ using this equation, we obtain the above linear multicommodity flow problem. Hence, $MC$ provides a lower bound $Z_{MC}^l$ on the optimal value at the current node, which is often significantly weaker than the cutting-plane lower bound $Z^l$, except when the current node is located deep in the B&C tree.

Capacity-based filtering starts by solving $MC$. If it is infeasible, the current node can be fathomed. Also, using Proposition 3, we can generate a combinatorial Benders cut, which is stored in the global cut pool. Otherwise, if $MC$ is feasible, we denote by $\widehat{x}$ an optimal solution. We first verify if $Z_{MC}^l \geq Z^*$, in which case the current node can be fathomed. Then, for each free arc $(i, j) \in A_{01}$ such that $\sum_{k \in K} \widehat{x}_{ij}^k > (1-\epsilon)u_{ij}$, where $\epsilon \in (0, 1)$ is a parameter, we solve $MC$ with the additional constraint $\sum_{k \in K} x_{ij}^k = 0$. If the resulting problem is infeasible, then we can conclude that arc $(i, j)$ must necessarily be opened in any optimal solution to the MCND. If the resulting problem is feasible, thus providing a lower bound $Z_{MC}^l(i, j)$, then we verify if $Z_{MC}^l(i, j) \geq Z^*$, in which case we can conclude again that arc $(i, j)$ must be opened in any optimal solution to the MCND.

To fully understand this filtering procedure, several remarks are in order. First, it is useless to test a free arc $(i, j) \in A_{01}$ such that $\sum_{k \in K} \widehat{x}_{ij}^k = 0$, since in that case, arc $(i, j)$ cannot be opened by the procedure. Second, this type of filtering achieves success mostly for free arcs that fully use their capacity in $\widehat{x}$. Hence, $\epsilon$ must be small (we use $\epsilon = 0.01$ in our tests). Third, it is possible to implement a similar filtering procedure that attempts to close free arcs with no flow circulating on them in solution $\widehat{x}$. Indeed, we tested this procedure, but given the weakness of the lower bound, its impact was very limited. Fourth, capacity-based filtering can succeed only when many arcs are fixed to 0. Hence, we perform it only when the number of closed arcs is large enough, i.e., if $|A_0| > \gamma |A|$, where $\gamma \in [0, 1]$ is a parameter (in Sect. 5, we show results for $\gamma = 0.85$, 0.9 and 0.95). Fifth, capacity-based filtering is called only when some arcs have been closed since the last time it was performed, because only arcs that are closed can incur infeasibility. Sixth, capacity-based filtering complements connectivity-based filtering and is therefore performed immediately after.

## 4 Overview of the branch-and-cut algorithm

This section summarizes the overall B&C algorithm. Before providing the details of the algorithm in Sect. 4.3, we first explain how upper bounds are computed during the course of the algorithm. This is the topic of Sect. 4.1, while in Sect. 4.2, we explain how we search the B&C tree.

### 4.1 Computation of upper bounds

As mentioned in Introduction, very good upper bounds are obtained by the heuristic methods proposed in the literature (Crainic et al. 2000, 2004; Crainic and Gendreau 2002; Ghamlouche et al. 2003, 2004; Hewitt et al. 2010; Katayama et al. 2009; Rodríguez-Martín and Salazar-González 2010). In our tests, reported in Sect. 5, we use as initial upper bound the value $(1+\xi) \times Z$, where $Z$ is the best known upper bound on the optimal value (which is the optimal value for most tested instances) and $\xi$ is a small number (we use $\xi = 0.00001$). Apart from the fact that it is realistic to assume that a very good initial upper bound is known, this setting allows to test the capacity of the B&C algorithm and the different filtering methods to focus only on lower bound improvement and optimality proof. Nevertheless, the B&C algorithm has the ability to compute upper bounds and to prove optimality, even if its initial upper bound is $+\infty$. We now show how these upper bounds are computed during the course of the algorithm.

The following (the proof of which is omitted, as it is trivial) states that we can derive an upper bound on the optimal value of the MCND from any feasible solution to the multicommodity flow system $MF$, presented in Sect. 3.4.

**Proposition 4** *For any feasible solution $\widehat{x}$ to $MF$,*

$$Z(\widehat{x}) = \sum_{k \in K} \sum_{(i,j) \in A} c_{ij} \widehat{x}_{ij}^k + \sum_{(i,j) \in A} f_{ij} \left\lceil \sum_{k \in K} \widehat{x}_{ij}^k / u_{ij} \right\rceil$$

*is an upper bound on the optimal value $Z$ of the MCND.*

**Corollary 5** *At any iteration of the cutting-plane procedure, if the LP relaxation is feasible, then*

$$Z(\bar{y}) = Z^l + \sum_{(i,j) \in A} f_{ij} (\lceil \bar{y}_{ij} \rceil - \bar{y}_{ij})$$

*is an upper bound on the optimal value $Z$ of the MCND.*

*Proof* First, we note that any solution $(\overline{x}, \overline{y})$ generated during the cutting-plane procedure satisfies the multicommodity flow system $MF$. In addition, we have $\bar{y}_{ij} \geq \sum_{k \in K} \overline{x}_{ij}^k / u_{ij}$, for each $(i, j) \in A$. By applying the previous proposition, we obtain an upper bound on the optimal value of the MCND:

$$\begin{aligned} Z(\overline{x}) &= \sum_{k \in K} \sum_{(i,j) \in A} c_{ij} \overline{x}_{ij}^k + \sum_{(i,j) \in A} f_{ij} \lceil \sum_{k \in K} \overline{x}_{ij}^k / u_{ij} \rceil \\ &\leq \sum_{k \in K} \sum_{(i,j) \in A} c_{ij} \overline{x}_{ij}^k + \sum_{(i,j) \in A} f_{ij} \lceil \overline{y}_{ij} \rceil \\ &= Z^l + \sum_{(i,j) \in A} f_{ij} (\lceil \overline{y}_{ij} \rceil - \overline{y}_{ij}). \end{aligned}$$

□

The last result is used to quickly compute an upper bound after performing the cutting-plane procedure. In particular, this bound has a nice interpretation when $\overline{y}$ is integral: in that case, $Z(\overline{y}) = Z^l$ and the lower bound test $Z^l \geq Z^*$ suffices to fathom the node. In addition, Proposition 4 is exploited when performing the capacity-based filtering method. Details are given below in the algorithm statement.

### 4.2 Tree search

We use a hybrid search strategy that combines the depth-first and best-first approaches. After branching, the next node to evaluate is the child that gives the smallest estimated lower bound increase among the two generated children, in order to mimic a best-first approach. When a strong branching evaluation has just been performed to select a branching arc $a^*$, this corresponds to the child that attains the value $\min(\Delta^0_{a^*}, \Delta^1_{a^*})$. When, instead, the branching arc $a^*$ is selected by a pseudo-cost estimate, the next child to evaluate is the one that achieves the value $\min(g^0_{a^*}\bar{\rho}^0_{a^*}, g^1_{a^*}\bar{\rho}^1_{a^*})$. The other child is stored in the *node pool* and will eventually be evaluated when backtracking is performed. When a newly generated node is stored in the node pool, we keep in memory its lower bound estimate, which is equal to the lower bound of its parent plus the estimated lower bound increase computed by the branching rule. When backtracking, we select the node that has the smallest lower bound estimate among all the nodes in the node pool.

### 4.3 Statement of the algorithm

We now outline the algorithm, the steps being commented below:

1. Initialize the upper bound $Z^*$, the node pool $\mathcal{L}$ and the current node as the root node ($\mathcal{L} \leftarrow \emptyset$ and $A_{01} \leftarrow A$).
2. *Evaluation:* Evaluate the current node:
   (a) Determine *LP-fix, LR-fix, Flow, Benders, Conn, Cap*.
   (b) If *Conn*, perform connectivity-based filtering; if the network at the current node is not connected, go to step 4 (*Backtrack*).
   (c) If *Cap*, perform capacity-based filtering:
      i. Solve $MC$.
      ii. If $MC$ is infeasible: if *Benders*, generate a combinatorial Benders cut; go to step 4.
      iii. Let $\widehat{x}$ be an optimal solution to $MC$; compute an upper bound $Z(\widehat{x})$; if $Z(\widehat{x}) < Z^*$, $Z^* \leftarrow Z(\widehat{x})$.
      iv. If $Z^l_{MC} \geq Z^*$, go to step 4.
      v. For each $(i, j) \in A_{01}$ such that $\sum_{k \in K} \widehat{x}^k_{ij} > (1 - \epsilon)u_{ij}$, solve $MC$ with the added constraint $\sum_{k \in K} \widehat{x}^k_{ij} = 0$; if $Z^l_{MC}(i, j) \geq Z^*$, open arc $(i, j)$.
   (d) Apply the cutting-plane procedure to solve the LP relaxation.
   (e) If the LP relaxation is infeasible: if *Benders*, generate a combinatorial Benders cut; go to step 4.

(f) Let $(\overline{x}, \overline{y})$ be an optimal solution to the LP relaxation; compute an upper bound $Z(\overline{y}) = Z^l + \sum_{(i,j)\in A} f_{ij}(\lceil \overline{y}_{ij} \rceil - \overline{y}_{ij})$; if $Z(\overline{y}) < Z^*$, $Z^* \leftarrow Z(\overline{y})$.
(g) If $Z^l \geq Z^*$, go to step 4.
(h) If *Benders*, try to add violated combinatorial Benders cuts; if cuts were generated, go to step 2d.
(i) If *LP-fix*, perform LP-based reduced cost fixing.
(j) If *Flow*, perform LP-based flow upper bound reduction.
(k) If *LR-fix* or *Flow*:
   i. Compute the Lagrangian relaxation bound $Z^l_{LR}$.
   ii. If $Z^l_{LR} \geq Z^*$, go to step 4.
   iii. If *LR-fix*, perform LR-based reduced cost fixing.
   iv. If *Flow*, perform LR-based flow upper bound reduction.

3. *Branching:* Perform branching to generate two child nodes; select one child as the next current node to evaluate; insert the other into $\mathcal{L}$; Go to step 2.
4. *Backtracking:* If $\mathcal{L} = \emptyset$, stop the algorithm; otherwise, select from $\mathcal{L}$ the next current node to evaluate and go to step 2.

In step 1, the upper bound is initialized as described in Sect. 4.1. The node pool $\mathcal{L}$ is also initialized, and the first current node is the root node. Step 2 is the main procedure to be performed at every node of the B&C tree. The details of that step are further commented below. Step 3 performs the reliability branching rule presented in Sect. 2.4. The next current node is selected among the two children according to the rule described in Sect. 4.2. Step 4 verifies the stopping condition $\mathcal{L} = \emptyset$ and, if it is not satisfied, it performs backtracking as discussed in Sect. 4.2. We also stop the algorithm when a time limit has been reached. Finally, the best global lower bound, $Z^l_+$, is stored and updated in an obvious way. This lower bound on the optimal value of the MCND is used to compute the final gap, $100 \times (Z^* - Z^l_+)/Z^*$, when the B&C algorithm is stopped by the time limit.

Step 2a determines the values of six parameters that are used to trigger the filtering methods at the current node. Each of these parameters is set to *False* if we do not want to activate the corresponding filtering method. Otherwise, a parameter value is set to *True* depending on the conditions that allow the execution of the filtering method, conditions that are described in the corresponding section. The parameters are *LP-fix, LR-fix, Flow, Benders, Conn, Cap*, which correspond to the following filtering methods, respectively: LP-based reduced cost fixing (Sect. 3.1 and step 2i), LR-based reduced cost fixing (Sect. 3.2 and step 2k), flow upper bound reduction (Sect. 3.3 and steps 2j and 2(k)iv), combinatorial Benders cuts (Sect. 3.4 and steps 2(c)ii, 2e and 2h), connectivity-based filtering (Sect. 3.5 and step 2b) and capacity-based filtering (Sect. 3.6 and step 2c). The cutting-plane procedure performed at step 2d follows the developments in Sect. 2.3. Finally, the computation and update of upper bounds, steps 2(c)iii and 2f, correspond to Sect. 4.1.

**Table 1** Classes and problem dimensions (number of instances within parentheses)

| Class I $\|V\|, \|A\|, \|K\|$ | (31) | Class II $\|V\|, \|A\|, \|K\|$ | (12) | Class III-A $\|V\|, \|A\|, \|K\|$ | (72) | Class III-B $\|V\|, \|A\|, \|K\|$ | (81) |
|---|---|---|---|---|---|---|---|
| 20, 230, 40 | (3) | 25, 100, 10 | (3) | 10, 35, 10 | (6) | 20, 120, 40 | (9) |
| 20, 230, 200 | (4) | 25, 100, 30 | (3) | 10, 35, 25 | (6) | 20, 120, 100 | (9) |
| 20, 300, 40 | (4) | 100, 400, 10 | (3) | 10, 35, 50 | (6) | 20, 120, 200 | (9) |
| 20, 300, 200 | (4) | 100, 400, 30 | (3) | 10, 60, 10 | (9) | 20, 220, 40 | (9) |
| 30, 520, 100 | (4) | | | 10, 60, 25 | (9) | 20, 220, 100 | (9) |
| 30, 520, 400 | (4) | | | 10, 60, 50 | (9) | 20, 220, 200 | (9) |
| 30, 700, 100 | (4) | | | 10, 85, 10 | (9) | 20, 320, 40 | (9) |
| 30, 700, 400 | (4) | | | 10, 85, 25 | (9) | 20, 320, 100 | (9) |
| | | | | 10, 85, 50 | (9) | 20, 320, 200 | (9) |

## 5 Computational results

This section presents computational results obtained by the B&C algorithm on a publicly available set of 196 instances (the so-called Canad instances, see Frangioni (2017)) used in several papers on the $MCND$, for instance (Ghamlouche et al. 2003; Hewitt et al. 2010; Kliewer and Timajev 2005), and described in detail in Crainic et al. (2001). These problem instances consist of general networks with one commodity per origin-destination pair and no parallel arcs. Associated with each arc are three positive quantities: the capacity, the routing cost, and the fixed cost. These instances are characterized by various degrees of capacity tightness, with regard to the total demand, and importance of the fixed cost, with respect to the routing cost.

The instances are divided into three classes. Class I [the "C" instances in Frangioni (2017)] consists of 31 problem instances with many commodities compared to the number of nodes, while Class II [the "C+" instances in Frangioni (2017)] contains 12 problem instances with few commodities compared to the number of nodes. Class III [the "R" instances in Frangioni (2017)] is divided into two categories, A and B, each containing nine sets of nine problem instances each. Each set is characterized by the numbers of nodes, arcs, and commodities, which are the same for the nine instances, and by instance-specific levels of capacity tightness and importance of the fixed cost. Class III-A (instances "R01" to "R09") contains 72 small size problem instances with 10 nodes (nine infeasible instances have been discarded), while Class III-B (instances "R10" to "R18") contains 81 medium to large size instances with 20 nodes. Table 1 gives the size of the instances in each class.

The B&C algorithm was implemented in C++ with the OOBB library (Crainic et al. 2009), using CPLEX version 12.6.1.0 as the LP solver. The code was compiled with g++ 4.8.1 and performed on an Intel Xeon ES-2609 v2 operating at 2,50 GHz, in single-threaded mode. All instances were solved with a time limit of 10 hours, which allows to divide the set of instances according to their difficulty, while at the same time to study trends in the evolution of the lower bounds for the most difficult instances, still unsolved after that time limit (we come back to this issue at the end of Sect. 5.1).

The following measures are used to evaluate the performance of the B&C algorithm: (1) CPU time in seconds; (2) number of generated B&C nodes; (3) relative gap in percentage computed as Gap = $100 \times (Z^* - Z^l_+)/Z^*$.

We first present the results obtained with different configurations of the filtering parameters in Sect. 5.1. Then, in Sect. 5.2, we compare the B&C variant including filtering with CPLEX and with other variants having limited filtering or no filtering at all. In both sections, we divide the instances into two classes: 148 instances solved by all parameter configurations within the time limit of 10 hours (for which Gap = 0), called *solved instances*, and 45 instances unsolved by any of the parameter configurations after the time limit of 10 hours (for which Gap $>$ 0), called *unsolved instances*. The remaining three instances are solved by some parameter configurations, but unsolved by others. To simplify the analysis, we did not include them in the tables of results presented in Sects. 5.1 and 5.2. Appendix presents detailed results of each of the 196 instances in the data set, including these three instances, for the "best" parameter configuration identified in Sect. 5.1.

### 5.1 Impact of the filtering methods

This section presents tables of results for different configurations of the filtering parameters, separating the analysis for each set of configurations into two tables, one for the solved instances and one for the unsolved instances. For the solved instances, we report only the CPU time in seconds and the number of nodes, Columns "CPU" and "Nodes," respectively, since Gap = 0 for all these instances. For the unsolved instances, we show only the number of nodes and the relative gap in percentage, Column "Gap," since the CPU time limit was attained for all these instances. Note that the value "Nodes" has different meanings, depending on the class of instances: for solved instances, a smaller number of nodes is to be preferred and is often correlated with a smaller CPU time, while for unsolved instances, a larger number of nodes is to be preferred and is often correlated with a smaller gap.

In each table, the first column gives the name of the class of instances, I, II, III-A or III-B, and the number of instances on which the average performance measures are computed. Both arithmetic means and shifted geometric means are reported for each performance measure. Shifted geometric means are now widely used for analyzing the performance of MIP solvers, since a geometric mean prevents hard instances close to the CPU time limit from having a huge impact on the measures, while the shift reduces the effect of very easy instances. For both "CPU" and "Nodes," we use a shift of 100, while for "Gap," the shift is set to 0.001%. The second column in each table identifies the filtering parameters activated in each configuration. The next columns give the mean values for the performance measures, first the arithmetic means, "Arithmetic," then the shifted geometric means, "Shifted geom."

Tables 2 and 3 show the results obtained on solved and unsolved instances, respectively, for four parameter configurations. We compare the configuration with no filtering with three duality-based filtering configurations: LP-based reduced cost fixing is activated in all three configurations, while LR-based reduced cost fixing is activated

**Table 2** Results with duality-based filtering on solved instances

| Class | Parameters | Arithmetic | | Shifted geom | |
|---|---|---|---|---|---|
| | | CPU | Nodes | CPU | Nodes |
| I (11) | – | 3840 | 5025 | 344 | 674 |
| | *LP-fix* | 2390 | 4802 | 284 | 660 |
| | *LP-fix, LR-fix* | 2100 | 4661 | 268 | 651 |
| | *LP-fix, LR-fix, Flow* | 1987 | 4612 | 263 | 631 |
| II (8) | – | 2212 | 18,600 | 150 | 1629 |
| | *LP-fix* | 1354 | 17,658 | 130 | 1590 |
| | *LP-fix, LR-fix* | 1372 | 17,658 | 132 | 1590 |
| | *LP-fix, LR-fix, Flow* | 1350 | 17,658 | 131 | 1590 |
| III-A (72) | – | 8 | 325 | 6 | 103 |
| | *LP-fix* | 6 | 282 | 5 | 94 |
| | *LP-fix, LR-fix* | 6 | 281 | 5 | 93 |
| | *LP-fix, LR-fix, Flow* | 6 | 280 | 5 | 93 |
| III-B (57) | – | 4459 | 3845 | 604 | 827 |
| | *LP-fix* | 3412 | 3762 | 521 | 818 |
| | *LP-fix, LR-fix* | 2917 | 3724 | 481 | 809 |
| | *LP-fix, LR-fix, Flow* | 2914 | 3735 | 480 | 808 |

**Table 3** Results with duality-based filtering on unsolved instances

| Class | Parameters | Arithmetic | | Shifted geom | |
|---|---|---|---|---|---|
| | | Nodes | Gap | Nodes | Gap |
| I (19) | – | 10,595 | 0.77 | 3465 | 0.61 |
| | *LP-fix* | 11,422 | 0.75 | 3727 | 0.60 |
| | *LP-fix, LR-fix* | 12,182 | 0.74 | 3963 | 0.57 |
| | *LP-fix, LR-fix, Flow* | 11,996 | 0.74 | 3984 | 0.57 |
| II (3) | – | 436,831 | 4.87 | 343,471 | 2.60 |
| | *LP-fix* | 465,655 | 4.88 | 340,208 | 2.61 |
| | *LP-fix, LR-fix* | 444,157 | 4.88 | 328,695 | 2.62 |
| | *LP-fix, LR-fix, Flow* | 455,112 | 4.88 | 334,995 | 2.62 |
| III-B (23) | – | 48,336 | 1.04 | 11,948 | 0.76 |
| | *LP-fix* | 62,201 | 1.01 | 13,544 | 0.70 |
| | *LP-fix, LR-fix* | 64,285 | 0.95 | 14,727 | 0.65 |
| | *LP-fix, LR-fix, Flow* | 67,527 | 0.91 | 16,578 | 0.60 |

in two of the configurations and flow upper bound reduction is activated in only one configuration.

Before analyzing the results reported in Tables 2 and 3, it is important to note that, when any of the three duality-based filtering configurations is used, two instances unsolved by the configuration with no filtering (one more in each of Classes I and III-

B) are solved within the time limit of 10 hours, so that the number of solved instances increases from 149 (with no filtering) to 151. This is already a clear indication of the positive effect of LP-based reduced cost fixing. The two additional solved instances are not reported in any of the two tables, so that the comparison for both classes of instances, solved and unsolved, relies on the same instances and the performance measures retain the same meaning.

The results in Table 2 show that, when more filtering is performed, the number of nodes is generally reduced. In general, the most significant reduction in the number of nodes is observed for LP-based reduced cost fixing. These reductions in the number of nodes always translate into reductions in the CPU time. In general, the results in Tables 2 and 3 indicate that LP-based reduced cost fixing has a clear positive impact on the overall performance. The impact of the other duality-based filtering techniques is less clear, but we note that both the LR-based reduced cost fixing and the flow upper bound reduction allow to reduce the final gap for some hard instances in Class III-B. Thus, for the remaining tested parameter configurations, we activate *LP-fix*, *LR-fix*, and *Flow*.

Tables 4 and 5 show the results obtained by performing feasibility-based filtering, in addition to duality-based filtering. More specifically, we display the results obtained with three parameter configurations, obtained by activating *Benders* and *Conn* in isolation and in conjunction. To facilitate the comparison, we report the results when these parameters are not activated, which are also displayed in Tables 2 and 3. The same instances are used, 148 solved ones and 45 unsolved ones. The remaining three instances in the set of 196 instances are solved by the two configurations that use Benders cuts, but one of these three instances (in Class II) is no more solved when using connectivity-based filtering. However, the final gap for this instance is 0.07%, which is negligible. We can thus consider that the four configurations reported in Tables 4 and 5 are performing equally well in terms of the total number of solved instances within the limit of 10 hours of CPU time.

The results in Table 4 show that the addition of Benders cuts has a negligible impact on all instances. In contrast, connectivity-based filtering generally has a positive impact, especially on Class III-B instances. Overall, the best configuration is obtained by activating only connectivity-based filtering, with notable reductions in the CPU time on Class III-B instances. The results in Table 5 show decreases in the gap for all configurations that include feasibility-based filtering, except for Class III-B for which lower gaps are obtained only when *Conn* alone is activated, while higher gaps are observed when *Benders* is activated. The results in Tables 4 and 5 point to the general conclusions that the addition of combinatorial Benders cuts might have a positive impact for some instances, but that better results are obtained when connectivity-based filtering is used alone, without activating *Benders*.

Tables 6 and 7 show the results obtained when the parameter *Cap* is activated, in addition to *LP-fix*, *LR-fix*, *Flow* and *Conn*. We report the results with three values of the parameter $\gamma$, which controls when capacity-based filtering is performed depending on the proportion of design variables fixed to 0: $\gamma = 0.85, 0.90, 0.95$ (values around $\gamma = 0.90$ generally give the best results, according to preliminary experiments). To ease the comparison, we also report the results when *Cap* is not activated, which are

**Table 4** Results with feasibility-based filtering on solved instances

| Class | Parameters | Arithmetic | | Shifted geom | |
|---|---|---|---|---|---|
| | | CPU | Nodes | CPU | Nodes |
| I (11) | *LP-fix, LR-fix, Flow* | 1987 | 4612 | 263 | 631 |
| | *LP-fix, LR-fix, Flow, Benders* | 2016 | 5869 | 261 | 631 |
| | *LP-fix, LR-fix, Flow, Conn* | 1923 | 4705 | 265 | 630 |
| | *LP-fix, LR-fix, Flow, Benders, Conn* | 1940 | 4704 | 268 | 630 |
| II (8) | *LP-fix, LR-fix, Flow* | 1350 | 17,658 | 131 | 1590 |
| | *LP-fix, LR-fix, Flow, Benders* | 1665 | 18,074 | 135 | 1528 |
| | *LP-fix, LR-fix, Flow, Conn* | 1072 | 15,307 | 127 | 1589 |
| | *LP-fix, LR-fix, Flow, Benders, Conn* | 1407 | 15,312 | 133 | 1544 |
| III-A (72) | *LP-fix, LR-fix, Flow* | 6 | 280 | 5 | 93 |
| | *LP-fix, LR-fix, Flow, Benders* | 6 | 278 | 5 | 92 |
| | *LP-fix, LR-fix, Flow, Conn* | 6 | 284 | 5 | 92 |
| | *LP-fix, LR-fix, Flow, Benders, Conn* | 6 | 282 | 5 | 92 |
| III-B (57) | *LP-fix, LR-fix, Flow* | 2914 | 3735 | 480 | 808 |
| | *LP-fix, LR-fix, Flow, Benders* | 2891 | 3733 | 475 | 808 |
| | *LP-fix, LR-fix, Flow, Conn* | 2557 | 3695 | 451 | 790 |
| | *LP-fix, LR-fix, Flow, Benders, Conn* | 2661 | 3683 | 452 | 768 |

**Table 5** Results with feasibility-based filtering on unsolved instances

| Class | Parameters | Arithmetic | | Shifted geom | |
|---|---|---|---|---|---|
| | | Nodes | Gap | Nodes | Gap |
| I (19) | *LP-fix, LR-fix, Flow* | 11,996 | 0.74 | 3984 | 0.57 |
| | *LP-fix, LR-fix, Flow, Benders* | 11,275 | 0.74 | 4006 | 0.56 |
| | *LP-fix, LR-fix, Flow, Conn* | 12,856 | 0.73 | 4424 | 0.55 |
| | *LP-fix, LR-fix, Flow, Benders, Conn* | 12,848 | 0.73 | 4401 | 0.55 |
| II (3) | *LP-fix, LR-fix, Flow* | 455,112 | 4.88 | 334,995 | 2.62 |
| | *LP-fix, LR-fix, Flow, Benders* | 440,386 | 4.87 | 327,037 | 2.61 |
| | *LP-fix, LR-fix, Flow, Conn* | 416,637 | 4.86 | 313,117 | 2.62 |
| | *LP-fix, LR-fix, Flow, Conn, Benders* | 380,457 | 4.83 | 380,557 | 2.61 |
| III-B (23) | *LP-fix, LR-fix, Flow* | 67,527 | 0.91 | 16,578 | 0.60 |
| | *LP-fix, LR-fix, Flow, Benders* | 63,725 | 0.95 | 14,701 | 0.65 |
| | *LP-fix, LR-fix, Flow, Conn* | 62,209 | 0.89 | 17,706 | 0.55 |
| | *LP-fix, LR-fix, Flow, Benders, Conn* | 59,398 | 0.92 | 15,766 | 0.61 |

already shown in Tables 4 and 5. The same instances are also used, 148 solved ones and 45 unsolved ones.

The results in Table 6 show that capacity-based filtering (with the tested values of $\gamma$) has a marginal impact on both the number of nodes and the CPU time. In

**Table 6** Results with capacity-based filtering on solved instances

| Class | Parameters | Arithmetic | | Shifted geom | |
|---|---|---|---|---|---|
| | | CPU | Nodes | CPU | Nodes |
| I (11) | *LP-fix, LR-fix, Flow, Conn* | 1923 | 4705 | 265 | 630 |
| | *LP-fix, LR-fix, Flow, Conn, Cap* ($\gamma = 0.85$) | 2023 | 4705 | 270 | 630 |
| | *LP-fix, LR-fix, Flow, Conn, Cap* ($\gamma = 0.90$) | 1929 | 4705 | 263 | 630 |
| | *LP-fix, LR-fix, Flow, Conn, Cap* ($\gamma = 0.95$) | 1912 | 4705 | 265 | 630 |
| II (8) | *LP-fix, LR-fix, Flow, Conn* | 1072 | 15,307 | 127 | 1589 |
| | *LP-fix, LR-fix, Flow, Conn, Cap* ($\gamma = 0.85$) | 1079 | 15,307 | 127 | 1589 |
| | *LP-fix, LR-fix, Flow, Conn, Cap* ($\gamma = 0.90$) | 1077 | 15,307 | 126 | 1589 |
| | *LP-fix, LR-fix, Flow, Conn, Cap* ($\gamma = 0.95$) | 1080 | 15,307 | 126 | 1589 |
| III-A (72) | *LP-fix, LR-fix, Flow, Conn* | 6 | 284 | 5 | 92 |
| | *LP-fix, LR-fix, Flow, Conn, Cap* ($\gamma = 0.85$) | 6 | 284 | 5 | 92 |
| | *LP-fix, LR-fix, Flow, Conn, Cap* ($\gamma = 0.90$) | 6 | 284 | 5 | 92 |
| | *LP-fix, LR-fix, Flow, Conn, Cap* ($\gamma = 0.95$) | 6 | 284 | 5 | 92 |
| III-B (57) | *LP-fix, LR-fix, Flow, Conn* | 2657 | 3695 | 451 | 790 |
| | *LP-fix, LR-fix, Flow, Conn, Cap* ($\gamma = 0.85$) | 2665 | 3695 | 454 | 790 |
| | *LP-fix, LR-fix, Flow, Conn, Cap* ($\gamma = 0.90$) | 2640 | 3695 | 451 | 790 |
| | *LP-fix, LR-fix, Flow, Conn, Cap* ($\gamma = 0.95$) | 2624 | 3695 | 449 | 790 |

**Table 7** Results with capacity-based filtering on unsolved instances

| Class | Parameters | Arithmetic | | Shifted geom | |
|---|---|---|---|---|---|
| | | Nodes | Gap | Nodes | Gap |
| I (19) | *LP-fix, LR-fix, Flow, Conn* | 12,856 | 0.73 | 4424 | 0.55 |
| | *LP-fix, LR-fix, Flow, Conn, Cap* ($\gamma = 0.85$) | 10,429 | 0.78 | 4402 | 0.55 |
| | *LP-fix, LR-fix, Flow, Conn, Cap* ($\gamma = 0.90$) | 13,082 | 0.73 | 4438 | 0.55 |
| | *LP-fix, LR-fix, Flow, Conn, Cap* ($\gamma = 0.95$) | 13,209 | 0.73 | 4453 | 0.55 |
| II (3) | *LP-fix, LR-fix, Flow, Conn* | 416,637 | 4.86 | 313,117 | 2.62 |
| | *LP-fix, LR-fix, Flow, Conn, Cap* ($\gamma = 0.85$) | 445,574 | 4.86 | 328,383 | 2.60 |
| | *LP-fix, LR-fix, Flow, Conn, Cap* ($\gamma = 0.90$) | 418,647 | 4.86 | 313,013 | 2.62 |
| | *Lp-fix, LR-fix, Flow, Conn, Cap* ($\gamma = 0.95$) | 416,490 | 4.86 | 313,148 | 2.62 |
| III-B (23) | *LP-fix, LR-fix, Flow, Conn* | 62,209 | 0.89 | 17,706 | 0.55 |
| | *LP-fix, LR-fix, Flow, Conn, Cap* ($\gamma = 0.85$) | 58,902 | 0.92 | 15,881 | 0.61 |
| | *LP-fix, LR-fix, Flow, Conn, Cap* ($\gamma = 0.90$) | 59,260 | 0.92 | 15,867 | 0.63 |
| | *LP-fix, LR-fix, Flow, Conn, Cap* ($\gamma = 0.95$) | 62,181 | 0.88 | 18,181 | 0.49 |

preliminary experiments, we have observed significant variations in the CPU time for smaller values of $\gamma$, but this behavior is highly instance-dependent and, thus, difficult to generalize to obtain consistent improvements. The results in Table 7 confirm that capacity-based filtering (with the tested values of $\gamma$) has a marginal impact on the

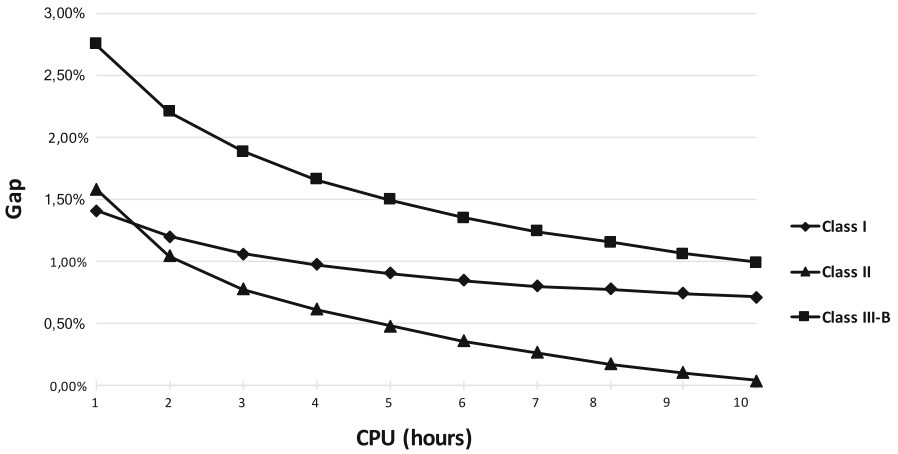


**Fig. 1** Gap vs CPU(h), one instance per class I: c58; II: c100_400_30_F_L; III-B: r17.6

performance of the algorithm. Overall, the results in Tables 6 and 7 point to the conclusion that capacity-based filtering, with appropriate values of the parameter $\gamma$, has a marginal effect on the overall performance of the algorithm. When $\gamma = 0.95$, we note, however, that slight improvements are obtained on most instances, with reductions in the CPU time for solved instances (in Classes I and III-B) and reductions in the gap for unsolved instances (in Class III-B). For the remaining tests, we thus activate *Cap* with $\gamma = 0.95$.

Note that we have performed additional tests where we modify the order in which the configurations are tested [a systematic approach could have been used, e.g., the fractional design-of-experiments described in Adenso-Diaz and Laguna (2006)]. The results are consistent with the following observations: (1) Duality-based filtering (including *Flow*) should be performed in all cases; (2) connectivity-based filtering must be performed in all cases and Benders cuts should be avoided; (3) capacity-based filtering generally has a marginally positive impact when it is performed in deep regions of the search tree (i.e., with values of $\gamma$ around 0.9). The sequential way in which we have presented the configuration testing has been adopted to facilitate the exposition.

Before comparing the different B&C variants, we look at the evolution of the lower bounds over the course of the algorithm. Figure 1 shows the evolution of gaps with respect to CPU times (in hours) on three difficult instances, each taken from a different class. The name of each instance comes from Frangioni (2017), which also specifies their dimensions $|V|$, $|A|$, $|K|$: c58 in Class I has size 30,700,100; c100_400_30_F_L in Class II has size 100,400,30; r17.6 in Class III-B has size 20,320,100. Note that these three instances are representative of the behavior over the whole set of unsolved instances. To obtain the gap evolution curves shown in Fig. 1, we ran the B&C algorithm by reporting the gap after each hour. We used the following configuration of filtering parameters, based on our observations: *LP-Fix*, *LR-Fix*, *Flow*, *Conn* and *Cap* ($\gamma = 0.95$). The gap evolution curve for each of the three instances exhibits a convex shape, indicating that the gap diminishes relatively quickly at the beginning and relatively slowly as we approach the time limit. On all unsolved instances, including

the three representative ones, the gap decreases by less than 2%, sometimes by less than 1%, as can be seen for Class I instance c58 on the graph. It is interesting to note that Class II instance c100_400_30_F_L is almost solved after 10 hours, with a final gap smaller than 0.1%. In general, however, even though 10 hours is a long time to be given to the B&C algorithm, most difficult instances are still far from being solved when the time limit is attained.

### 5.2 Comparison between branch-and-cut variants

In this section, we summarize the performances of three variants of the B&C algorithm: with no filtering, identified as "B&C"; with only the "classical" reduced cost fixing techniques *LP-Fix* and *LR-Fix*, "B&C&Fix"; and with the best identified configuration of filtering parameters, "B&C&Filter," which activates, in addition to *LP-Fix* and *LR-Fix*, *Flow*, *Conn*, and *Cap* ($\gamma = 0.95$). We use the same set of instances as before, 148 solved and 45 unsolved instances, so these results can be found already in the previous tables, but they are easier to read in Tables 8 and 9, which show the results on solved and unsolved instances, respectively. In addition, these two tables show the results obtained with CPLEX on the strong formulation, defined by (1)–(5), with the addition of the strong inequalities (6). Since for many instances, there are too many strong inequalities to add all of them *a priori*, they are declared as *user* cuts, which allows CPLEX to generate them dynamically, within its own branch-and-cut algorithm. In preliminary experiments, this approach was shown to be superior to the alternative that consists in solving the so-called weak formulation defined by (1)–(5) by CPLEX, i.e., the strong inequalities are not given to CPLEX. CPLEX is performed with default parameters, with two exceptions: we give as initial incumbent value the same upper bound provided to the B&C and we deactivate the heuristic features of CPLEX. Since CPLEX does not solve the same instances as the three B&C variants, we also count: (1) the number of instances that CPLEX does not solve among those that are solved by the B&C variants, reported with a "–" sign in column "Instances" of Table 8; (2) the number of instances that CPLEX solves among those that are unsolved by the B&C variants, reported with a "+" sign in column "Instances" of Table 9.

The results in Tables 8 and 9 show that the filtering methods have a notable impact on the performance of the B&C algorithm for all classes of instances. On the solved instances, reductions in the CPU time are observed: 50%, 51%, 25% and 41% reductions in the arithmetic means are obtained on Classes I, II, III-A, and III-B, respectively, with corresponding reductions of 23%, 16%, 17%, and 26% in the shifted geometric means. Among the filtering techniques, reduced cost fixing is responsible for a major part of these improvements, with reductions in the CPU time of 45%, 38%, 25% and 35% in the arithmetic means on Classes I, II, III-A and III-B, respectively, with corresponding reductions of 22%, 12%, 17% and 20% in the shifted geometric means. On unsolved instances, the arithmetic means of the gaps are reduced by 0.04% and 0.16% on Classes I and III-B, respectively, with corresponding reductions of 0.05% and 0.27% in the shifted geometric means. The reductions in the gap are generally correlated with increases in the number of nodes that can be explored within the time limit of 10 hours of CPU time: 25% and 29% increases in the arithmetic means of the

**Table 8** Comparison with CPLEX on solved instances

| Class | Method | Arithmetic | | | Shifted geom | | | Instances |
|---|---|---|---|---|---|---|---|---|
| | | CPU | Nodes | Gap | CPU | Nodes | Gap | |
| I (11) | *B&C&Filter* | 1912 | 4705 | 0.00 | 265 | 630 | 0.00 | 0 |
| | *B&C&Fix* | 2100 | 4661 | 0.00 | 268 | 651 | 0.00 | 0 |
| | *B&C* | 3840 | 5025 | 0.00 | 344 | 674 | 0.00 | 0 |
| | *CPLEX* | 3882 | 299,527 | 0.06 | 235 | 894 | 0.00 | −1 |
| II (8) | *B&C&Filter* | 1080 | 15,307 | 0.00 | 126 | 1589 | 0.00 | 0 |
| | *B&C&Fix* | 1372 | 17,658 | 0.00 | 132 | 1590 | 0.00 | 0 |
| | *B&C* | 2212 | 18,600 | 0.00 | 150 | 1629 | 0.00 | 0 |
| | *CPLEX* | 23 | 977 | 0.00 | 17 | 459 | 0.00 | 0 |
| III-A (72) | *B&C&Filter* | 6 | 284 | 0.00 | 5 | 92 | 0.00 | 0 |
| | *B&C&Fix* | 6 | 281 | 0.00 | 5 | 93 | 0.00 | 0 |
| | *B&C* | 8 | 325 | 0.00 | 6 | 103 | 0.00 | 0 |
| | *CPLEX* | 4 | 1306 | 0.00 | 3 | 103 | 0.00 | 0 |
| III-B (57) | *B&C&Filter* | 2624 | 3695 | 0.00 | 449 | 790 | 0.00 | 0 |
| | *B&C&Fix* | 2917 | 3724 | 0.00 | 481 | 809 | 0.00 | 0 |
| | *B&C* | 4459 | 3845 | 0.00 | 604 | 827 | 0.00 | 0 |
| | *CPLEX* | 11,628 | 1,965,599 | 0.60 | 930 | 12,364 | 0.01 | −16 |

number of nodes are obtained on Classes I and III-B, respectively, with corresponding increases of 29% and 52% in the shifted geometric means.

These results also show that the three B&C variants outperform CPLEX on Classes I and III-B, while the opposite is true for Class II. These observations are consistent with the results presented in Chouman et al. (2017), where it was already shown that the strong inequalities are particularly useful for instances with many commodities, such as those found in Classes I and III-B, while other types of cuts, namely flow cover/pack inequalities, are more effective for instances with few commodities, such as those in Class II. As our B&C algorithm relies mostly on strong inequalities, while CPLEX generates flow cover inequalities (among other types of cuts), these comparative results are consistent with those presented in Chouman et al. (2017). The appendix gives the detailed results, for each of the 196 instances in the data set, of the comparison between CPLEX and B&C&Filter. Overall, B&C&Filter solves 19 instances more than CPLEX on instances in Classes I and III-B (B&C&Filter solves two of the three instances not considered in Tables 8 and 9), while CPLEX solves one instance more than B&C&Filter on instances of Class II. On the unsolved instances in Classes I and III-B, the gap is also significantly smaller with B&C&Filter, compared with the gap obtained by CPLEX.

## 6 Conclusion

We have presented a B&C algorithm for the MCND that combines the cutting-plane method from Chouman et al. (2017), an adaptation of the reliability branching rule

**Table 9** Comparison with CPLEX on unsolved instances

| Class | Method | Arithmetic | | | Shifted geom | | | Instances |
|---|---|---|---|---|---|---|---|---|
| | | CPU | Nodes | Gap | CPU | Nodes | Gap | |
| I (19) | *B&C&Filter* | 36,000 | 13,209 | 0.73 | 36,000 | 4453 | 0.55 | 0 |
| | *B&C&Fix* | 36,000 | 12,182 | 0.74 | 36,000 | 3963 | 0.57 | 0 |
| | *B&C* | 36,000 | 10,595 | 0.77 | 36,000 | 3465 | 0.61 | 0 |
| | *CPLEX* | 36,000 | 1,286,115 | 1.84 | 36,000 | 877,536 | 1.55 | 0 |
| II (3) | *B&C&Filter* | 36,000 | 416,490 | 4.86 | 36,000 | 313,148 | 2.62 | 0 |
| | *B&C&Fix* | 36,000 | 444,157 | 4.88 | 36,000 | 328,695 | 2.62 | 0 |
| | *B&C* | 36,000 | 469,831 | 4.87 | 36,000 | 343,471 | 2.60 | 0 |
| | *CPLEX* | 24,010 | 2,504,962 | 3.42 | 5433 | 270,793 | 0.30 | +1 |
| III-B (23) | *B&C&Filter* | 36,000 | 62,181 | 0.88 | 36,000 | 18,181 | 0.49 | 0 |
| | *B&C&Fix* | 36,000 | 64,285 | 0.95 | 36,000 | 14,727 | 0.65 | 0 |
| | *B&C* | 36,000 | 48,336 | 1.04 | 36,000 | 11,948 | 0.76 | 0 |
| | *CPLEX* | 36,000 | 2,420,627 | 2.74 | 36,000 | 1,567,072 | 2.27 | 0 |

Achterberg et al. (2005), and a series of filtering methods taking advantage of the structure of the MCND. Our experiments on a large set of randomly generated instances have demonstrated the efficiency and the effectiveness of both the B&C algorithm and the filtering methods. In particular, these experiments have shown that an appropriate selection of filtering techniques allows the B&C algorithm to perform better than the variant of the algorithm without filtering. These experiments have confirmed that the B&C algorithm, with or without filtering, is competitive with a state-of-the-art MIP solver, especially for instances with many commodities (typically more than 100).

The filtering methods exploit the particular structure of the MCND. It would be interesting to adapt them to other exact algorithms for the MCND (see the references in Introduction). In particular, the feasibility-based filtering techniques (combinatorial Benders cuts, connectivity-based and capacity-based filtering) do not depend on the cutting-plane method and can be used in any enumerative algorithm. In contrast, the duality-based filtering techniques (reduced cost bound reduction, LP-based and Lagrangian-based reduced cost fixing) depend on the cutting-plane procedure, but could be adapted in the context of column generation and Lagrangian relaxation methods. The implementation of these different bounding methods under the same interface for enumerative algorithms, including adaptations of the filtering and branching procedures presented here, would allow a fair comparison of the exact approaches proposed so far for the MCND. Finally, another avenue of research would be to adapt the filtering methods to other difficult network design problems.

**Acknowledgements** The authors express their gratitude to Geneviève Hernu and Serge Bisaillon, analysts at CIRRELT. Their efforts and dedication in implementing and testing the innumerable variants of our methods were instrumental in achieving our results. While working on this project, the second author was the NSERC Industrial Research Chair on Logistics Management, ESG UQAM, and Adjunct Professor, Department of Computer Science and Operations Research, Université de Montréal, and Department of Economics and Business Administration, Molde University College, Norway. Funding for this project has been provided by NSERC, through its Industrial Research Chair and Discovery Grants programs, by the partners of the Chair, CN, Rona, Alimentation Couche-Tard and the Ministry of Transportation of Québec. We also gratefully acknowledge the support of FRQNT through their infrastructure grants and of Calcul Québec and Compute Canada through access to their computing infrastructure.

## Appendix: Detailed results for CPLEX and B&C&Filter

Tables 10, 11, 12, 13, 14, and 15 give the detailed results for the 196 instances in the data set. The name of each instance, "Instance," is taken from Frangioni (2017), which also specifies the dimensions of each instance. The performance measures are defined in Sect. 5. The lines "Arithm" and "ShiftG" at the end of a table provide the arithmetic and the shifted geometric means over all instances in the corresponding Class. Each instance name in bold corresponds to an instance solved by B&C&Filter, but not by CPLEX. Each instance name in italics shows an instance solved by CPLEX, but not by B&C&Filter.

**Table 10** CPLEX and B&C&Filter on Class I instances

| Instance | $Z^*$ | CPLEX | | | | B&C&Filter | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | $Z^l_+$ | CPU | Nodes | Gap (%) | $Z^l_+$ | CPU | Nodes | Gap (%) |
| c33 | 423,848 | 423,848 | 0 | 26 | 0.00 | 423,848 | 0 | 18 | 0.00 |
| c35 | 371,475 | 371,475 | 0 | 42 | 0.00 | 371,475 | 4 | 178 | 0.00 |
| c36 | 643,036 | 643,036 | 3 | 530 | 0.00 | 643,035 | 329 | 12,592 | 0.00 |
| **c37** | 94,213 | 92,347 | 36,000 | 2,327,873 | 1.98 | 94,213 | 14,136 | 4229 | 0.00 |
| c38 | 137,643 | 133,741 | 36,000 | 1,679,442 | 2.84 | 137,113 | 36,003 | 5300 | 0.39 |
| c39 | 97,914 | 97,914 | 3565 | 5638 | 0.00 | 97,914 | 6336 | 2621 | 0.00 |
| c40 | 136,186 | 132,484 | 36,000 | 1,674,809 | 2.72 | 135,240 | 36,001 | 5975 | 0.69 |
| c41 | 429,398 | 429,398 | 0 | 4 | 0.00 | 429,398 | 0 | 4 | 0.00 |
| c42 | 586,077 | 586,077 | 3 | 439 | 0.00 | 586,077 | 10 | 216 | 0.00 |
| c43 | 464,509 | 464,509 | 1 | 59 | 0.00 | 464,509 | 13 | 455 | 0.00 |
| c44 | 604,198 | 604,198 | 1 | 65 | 0.00 | 604,198 | 3 | 95 | 0.00 |
| c45 | 74,811 | 73,375 | 36,000 | 1,647,997 | 1.92 | 74,544 | 36,001 | 12,422 | 0.36 |
| c46 | 115,525 | 111,534 | 36,000 | 1,271,601 | 3.46 | 113,726 | 36,006 | 4609 | 1.56 |
| c47 | 74,991 | 74,991 | 3101 | 6201 | 0.00 | 74,991 | 2457 | 1234 | 0.00 |
| c48 | 107,102 | 104,473 | 36,000 | 1,233,930 | 2.46 | 106,417 | 36,001 | 7741 | 0.64 |

**Table 10** continued

| Instance | $Z^*$ | CPLEX | | | | B&C&Filter | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | $Z^l_+$ | CPU | Nodes | Gap (%) | $Z^l_+$ | CPU | Nodes | Gap (%) |
| **c49** | 53,958 | 53,623 | 36,000 | 3,281,413 | 0.62 | 53,958 | 11,838 | 34,277 | 0.00 |
| c50 | 94,043 | 90,872 | 36,000 | 2,220,076 | 3.37 | 93,270 | 36,002 | 9422 | 0.82 |
| c51 | 52,046 | 51,736 | 36,000 | 2,985,786 | 0.60 | 51,966 | 36,000 | 101,008 | 0.15 |
| c52 | 97,338 | 94,741 | 36,000 | 2,985,786 | 2.67 | 95,987 | 36,000 | 11,192 | 1.39 |
| c53 | 112,775 | 112,030 | 36,000 | 406,570 | 0.66 | 112,607 | 36,007 | 1471 | 0.15 |
| c54 | 149,094 | 147,141 | 36,000 | 1,320,948 | 1.31 | 147,877 | 36,025 | 778 | 0.82 |
| c55 | 114,640 | 114,302 | 36,000 | 433,985 | 0.30 | 114,593 | 36,003 | 2161 | 0.04 |
| c56 | 152,414 | 150,091 | 36,000 | 279,812 | 1.53 | 150,801 | 36,023 | 819 | 1.06 |
| c57 | 47,603 | 47,603 | 31 | 376 | 0.00 | 47,603 | 37 | 69 | 0.00 |
| c58 | 60,023 | 58,498 | 36,000 | 2,325,696 | 2.54 | 59,576 | 36,000 | 15,179 | 0.74 |
| c59 | 45,872 | 45,457 | 36,000 | 2,374,757 | 0.91 | 45,719 | 36,000 | 45,490 | 0.33 |
| c60 | 54,938 | 54,170 | 36,000 | 2,171,334 | 1.40 | 54,603 | 36,001 | 24,567 | 0.61 |
| c61 | 97,862 | 96,804 | 36,000 | 444,895 | 1.08 | 97,287 | 36,013 | 719 | 0.59 |
| c62 | 134,690 | 13,137 | 36,000 | 202,232 | 2.64 | 132,102 | 36,023 | 490 | 1.92 |
| c63 | 95,308 | 94,238 | 36,000 | 265,454 | 1.12 | 94,655 | 36,008 | 921 | 0.69 |
| c64 | 129,869 | 127,888 | 36,000 | 163,467 | 1.53 | 128,672 | 36,021 | 703 | 0.92 |
| Arithm | | | 24,603 | 969,641 | 1.21 | | 23,203 | 9902 | 0.45 |
| ShiftG | | | 6761 | 81,493 | 0.14 | | 6766 | 2274 | 0.05 |

**Table 11** CPLEX and B&C&Filter on Class II instances

| Instance | $Z^*$ | CPLEX | | | | B&C&Filter | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | $Z^l_+$ | CPU | Nodes | Gap (%) | $Z^l_+$ | CPU | Nodes | Gap (%) |
| c100_400_10_F_L | 23,949 | 23,949 | 141 | 2859 | 0.00 | 23,949 | 8341 | 10,9171 | 0.00 |
| c100__400_10_F_T | 63,753 | 60,545 | 36,000 | 5,318,740 | 5.03 | 59,206 | 36,000 | 661,706 | 7.13 |
| c100_400_10_V_L | 28,423 | 28,423 | 0 | 27 | 0.00 | 28,423 | 70 | 5052 | 0.00 |
| c100_400_30_F_L | 49,018 | 47,885 | 36,000 | 4,390,559 | 2.31 | 48,975 | 36,001 | 32,546 | 0.09 |
| c100_400_30_F_T | 136,780 | 129,636 | 36,000 | 2,194,543 | 5.22 | 12,7076 | 36,000 | 93,930 | 7.09 |
| *c100_400_30_V_T* | 384,802 | 384,802 | 30 | 1603 | 0.00 | 383,440 | 36,000 | 493,833 | 0.35 |
| c25_100_10_F_L | 14,941 | 14,941 | 5 | 1223 | 0.00 | 14,941 | 7 | 477 | 0.00 |
| c25_100_10_F_T | 49,899 | 49,899 | 1 | 405 | 0.00 | 49,899 | 10 | 1607 | 0.00 |
| c25_100_10_V_L | 14,712 | 14,712 | 0 | 0 | 0.00 | 14,712 | 0 | 4 | 0.00 |
| c25_100_30_F_L | 37,055 | 37,055 | 31 | 2201 | 0.00 | 37,055 | 153 | 3836 | 0.00 |
| c25_100_30_F_T | 85,530 | 85,530 | 6 | 1053 | 0.00 | 85,530 | 60 | 2290 | 0.00 |
| c25_100_30_V_T | 365,272 | 365,272 | 0 | 44 | 0.00 | 365,272 | 0 | 22 | 0.00 |
| Arithm | | | 9018 | 992,761 | 1.05 | | 12,720 | 117,040 | 1.22 |
| ShiftG | | | 395 | 5441 | 0.01 | | 1127 | 7877 | 0.01 |

**Table 12** CPLEX and B&C&Filter on Class III-A instances (1)

| Instance | $Z^*$ | CPLEX | | | | B&C&Filter | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | $Z^l_+$ | CPU | Nodes | Gap (%) | $Z^l_+$ | CPU | Nodes | Gap (%) |
| r01.1 | 74,079 | 74,079 | 0 | 0 | 0.00 | 74,079 | 0 | 1 | 0.00 |
| r01.2 | 92,403 | 92,403 | 0 | 0 | 0.00 | 92,403 | 0 | 1 | 0.00 |
| r01.3 | 115,304 | 115,304 | 0 | 2 | 0.00 | 115,304 | 0 | 4 | 0.00 |
| r01.4 | 84,908 | 84,908 | 0 | 8 | 0.00 | 84,908 | 0 | 13 | 0.00 |
| r01.5 | 113,036 | 113,036 | 0 | 33 | 0.00 | 113,036 | 0 | 23 | 0.00 |
| r01.6 | 147,599 | 147,599 | 0 | 50 | 0.00 | 147,599 | 0 | 49 | 0.00 |
| r02.1 | 232,239 | 232,239 | 0 | 3 | 0.00 | 232,239 | 0 | 6 | 0.00 |
| r02.2 | 322,453 | 322,453 | 0 | 13 | 0.00 | 322,453 | 0 | 15 | 0.00 |
| r02.3 | 419,503 | 419,503 | 0 | 18 | 0.00 | 419,503 | 0 | 13 | 0.00 |
| r02.4 | 316,437 | 316,437 | 0 | 0 | 0.00 | 316,437 | 0 | 5 | 0.00 |
| r02.5 | 431,250 | 431,250 | 0 | 5 | 0.00 | 431,250 | 0 | 11 | 0.00 |
| r02.6 | 559,578 | 559,578 | 0 | 4 | 0.00 | 559,578 | 0 | 11 | 0.00 |
| r03.1 | 484,830 | 484,830 | 0 | 3 | 0.00 | 484,830 | 0 | 3 | 0.00 |
| r03.2 | 703,362 | 703,362 | 0 | 5 | 0.00 | 703,362 | 0 | 7 | 0.00 |
| r03.3 | 944,990 | 944,990 | 0 | 14 | 0.00 | 944,990 | 0 | 13 | 0.00 |
| r03.4 | 704,247 | 704,247 | 0 | 0 | 0.00 | 704,247 | 0 | 3 | 0.00 |
| r03.5 | 932,897 | 932,897 | 0 | 12 | 0.00 | 932,897 | 0 | 11 | 0.00 |
| r03.6 | 1,188,638 | 1,188,638 | 0 | 5 | 0.00 | 1,188,640 | 0 | 4 | 0.00 |

**Table 12** continued

| Instance | $Z^*$ | CPLEX | | | | B&C&Filter | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | $Z^l_+$ | CPU | Nodes | Gap (%) | $Z^l_+$ | CPU | Nodes | Gap (%) |
| r04.1 | 31,730 | 31,730 | 0 | 0 | 0.00 | 31,730 | 0 | 1 | 0.00 |
| r04.2 | 48,920 | 48,920 | 0 | 3 | 0.00 | 48,920 | 0 | 3 | 0.00 |
| r04.3 | 63,767 | 63,767 | 0 | 2 | 0.00 | 63,767 | 0 | 3 | 0.00 |
| r04.4 | 33,740 | 33,740 | 0 | 6 | 0.00 | 33,740 | 0 | 33 | 0.00 |
| r04.5 | 53,790 | 53,790 | 0 | 20 | 0.00 | 53,790 | 0 | 44 | 0.00 |
| r04.6 | 74,030 | 74,030 | 0 | 80 | 0.00 | 74,030 | 0 | 31 | 0.00 |
| r04.7 | 68,292 | 68,292 | 0 | 133 | 0.00 | 68,292 | 0 | 87 | 0.00 |
| r04.8 | 113,004 | 113,004 | 0 | 147 | 0.00 | 113,004 | 0 | 129 | 0.00 |
| r04.9 | 163,208 | 163,208 | 0 | 105 | 0.00 | 163,208 | 0 | 117 | 0.00 |
| r05.1 | 123,003 | 123,003 | 0 | 0 | 0.00 | 123,003 | 0 | 2 | 0.00 |
| r05.2 | 170,060 | 170,060 | 0 | 7 | 0.00 | 170,060 | 0 | 7 | 0.00 |
| r05.3 | 221,486 | 221,486 | 0 | 149 | 0.00 | 221,486 | 1 | 42 | 0.00 |
| r05.4 | 131,608 | 131,608 | 0 | 7 | 0.00 | 131,608 | 0 | 10 | 0.00 |
| r05.5 | 204,157 | 204,157 | 0 | 104 | 0.00 | 204,157 | 1 | 44 | 0.00 |
| r05.6 | 286,524 | 286,524 | 1 | 449 | 0.00 | 286,524 | 6 | 348 | 0.00 |
| r05.7 | 278,372 | 278,372 | 0 | 10 | 0.00 | 278,372 | 0 | 10 | 0.00 |
| r05.8 | 445,810 | 445,810 | 0 | 30 | 0.00 | 445,810 | 0 | 22 | 0.00 |
| r05.9 | 625,879 | 625,879 | 0 | 16 | 0.00 | 625,879 | 0 | 8 | 0.00 |

**Table 13** CPLEX and B&C&Filter on Class III-A instances (2)

| Instance | $Z^*$ | CPLEX | | | | B&C&Filter | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | $Z^l_+$ | CPU | Nodes | Gap (%) | $Z^l_+$ | CPU | Nodes | Gap (%) |
| r06.1 | 245,936 | 245,936 | 0 | 28 | 0.00 | 245,936 | 0 | 12 | 0.00 |
| r06.2 | 401,685 | 401,685 | 3 | 437 | 0.00 | 401,685 | 3 | 66 | 0.00 |
| r06.3 | 559,477 | 559,477 | 7 | 483 | 0.00 | 559,477 | 17 | 244 | 0.00 |
| r06.4 | 286,682 | 286,682 | 0 | 201 | 0.00 | 286,682 | 2 | 117 | 0.00 |
| r06.5 | 498,266 | 498,266 | 12 | 1432 | 0.00 | 498,266 | 43 | 1415 | 0.00 |
| r06.6 | 734,414 | 734,414 | 159 | 79,339 | 0.00 | 734,414 | 130 | 3103 | 0.00 |
| r06.7 | 682,921 | 682,921 | 0 | 0 | 0.00 | 682,921 | 0 | 2 | 0.00 |
| r06.8 | 1,030,479 | 1,030,479 | 0 | 0 | 0.00 | 1,030,480 | 0 | 3 | 0.00 |
| r06.9 | 423,316 | 423,316 | 2 | 245 | 0.00 | 423,316 | 2 | 54 | 0.00 |
| r07.1 | 32,807 | 32,807 | 0 | 0 | 0.00 | 32,807 | 0 | 1 | 0.00 |
| r07.2 | 47,252 | 47,252 | 0 | 0 | 0.00 | 47,252 | 0 | 2 | 0.00 |
| r07.3 | 62,962 | 62,962 | 0 | 4 | 0.00 | 62,962 | 0 | 9 | 0.00 |
| r07.4 | 37,432 | 37,432 | 0 | 13 | 0.00 | 37,432 | 0 | 61 | 0.00 |
| r07.5 | 56,475 | 56,475 | 0 | 37 | 0.00 | 56,475 | 1 | 342 | 0.00 |
| r07.6 | 77,249 | 77,249 | 0 | 79 | 0.00 | 77,249 | 4 | 1071 | 0.00 |
| r07.7 | 59,947 | 59,947 | 0 | 181 | 0.00 | 59,947 | 2 | 388 | 0.00 |
| r07.8 | 99,194 | 99,194 | 0 | 143 | 0.00 | 99,194 | 3 | 630 | 0.00 |
| r07.9 | 141,692 | 141,692 | 1 | 520 | 0.00 | 141,692 | 8 | 1738 | 0.00 |

**Table 13** continued

| Instance | $Z^*$ | CPLEX | | | | B&C&Filter | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | $Z^l_+$ | CPU | Nodes | Gap (%) | $Z^l_+$ | CPU | Nodes | Gap (%) |
| r08.1 | 102,531 | 102,531 | 0 | 13 | 0.00 | 102,531 | 0 | 19 | 0.00 |
| r08.2 | 143,894 | 143,894 | 0 | 4 | 0.00 | 143,894 | 0 | 3 | 0.00 |
| r08.3 | 182,793 | 182,793 | 0 | 7 | 0.00 | 182,793 | 0 | 10 | 0.00 |
| r08.4 | 109,325 | 109,325 | 0 | 12 | 0.00 | 109,325 | 0 | 16 | 0.00 |
| r08.5 | 157,047 | 157,047 | 0 | 101 | 0.00 | 157,047 | 0 | 20 | 0.00 |
| r08.6 | 207,540 | 207,540 | 1 | 240 | 0.00 | 207,540 | 1 | 64 | 0.00 |
| r08.7 | 154,160 | 154,160 | 1 | 266 | 0.00 | 154,160 | 9 | 781 | 0.00 |
| r08.8 | 274,867 | 274,867 | 6 | 1053 | 0.00 | 274,866 | 58 | 3696 | 0.00 |
| r08.9 | 415,793 | 415,793 | 9 | 1649 | 0.00 | 415,793 | 51 | 2965 | 0.00 |
| r09.1 | 171,512 | 171,512 | 0 | 9 | 0.00 | 171,512 | 0 | 6 | 0.00 |
| r09.2 | 296,712 | 296,712 | 1 | 104 | 0.00 | 296,712 | 1 | 29 | 0.00 |
| r09.3 | 424,266 | 424,266 | 17 | 662 | 0.00 | 424,266 | 9 | 116 | 0.00 |
| r09.4 | 192,736 | 192,736 | 0 | 81 | 0.00 | 192,736 | 1 | 34 | 0.00 |
| r09.5 | 357,318 | 357,318 | 5 | 579 | 0.00 | 357,318 | 9 | 193 | 0.00 |
| r09.6 | 522,187 | 522,187 | 42 | 3259 | 0.00 | 522,187 | 50 | 936 | 0.00 |
| r09.7 | 345,057 | 345,057 | 0 | 97 | 0.00 | 345,057 | 3 | 113 | 0.00 |
| r09.8 | 646,579 | 646,579 | 1 | 121 | 0.00 | 646,579 | 4 | 126 | 0.00 |
| r09.9 | 951,136 | 951,136 | 8 | 1191 | 0.00 | 951,136 | 21 | 923 | 0.00 |
| Arithm | | | 4 | 1306 | 0.00 | | 6 | 284 | 0.00 |
| ShiftG | | | 3 | 103 | 0.00 | | 5 | 92 | 0.00 |

**Table 14** CPLEX and B&C&Filter on Class III-B instances (1)

| Instance | $Z^*$ | CPLEX | | | | B&C&Filter | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | $Z^l_+$ | CPU | Nodes | Gap (%) | $Z^l_+$ | CPU | Nodes | Gap (%) |
| r10.1 | 200,087 | 200,087 | 0 | 44 | 0.00 | 200,087 | 0 | 15 | 0.00 |
| r10.2 | 346,814 | 346,814 | 32 | 2312 | 0.00 | 346,814 | 12 | 145 | 0.00 |
| r10.3 | 488,015 | 488,015 | 33 | 1991 | 0.00 | 488,015 | 14 | 92 | 0.00 |
| r10.4 | 229,196 | 229,196 | 3 | 601 | 0.00 | 229,196 | 41 | 1804 | 0.00 |
| r10.5 | 411,664 | 411,664 | 15,618 | 7,614,891 | 0.00 | 411,664 | 477 | 7528 | 0.00 |
| **r10.6** | 609,103 | 597,295 | 36,000 | 13,386,972 | 1.27 | 609,104 | 2186 | 19,515 | 0.00 |
| r10.7 | 486,895 | 486,895 | 5 | 533 | 0.00 | 486,895 | 93 | 1559 | 0.00 |
| r10.8 | 951,056 | 951,056 | 11 | 985 | 0.00 | 951,056 | 86 | 1837 | 0.00 |
| r10.9 | 1,421,746 | 1,421,746 | 13 | 1386 | 0.00 | 1,421,750 | 105 | 2473 | 0.00 |
| r11.1 | 714,431 | 714,431 | 8 | 327 | 0.00 | 714,431 | 8 | 55 | 0.00 |
| **r11.2** | 1,263,713 | 1,246,267 | 36,000 | 5,510,745 | 1.38 | 1,263,710 | 1342 | 3208 | 0.00 |
| **r11.3** | 1,843,610 | 1,800,570 | 36,000 | 3,919,318 | 2.34 | 1,843,610 | 5026 | 6606 | 0.00 |
| r11.4 | 870,451 | 870,451 | 67 | 1984 | 0.00 | 870,450 | 296 | 1764 | 0.00 |
| r11.5 | 1,623,640 | 1,623,640 | 14,002 | 1,090,954 | 0.00 | 1,623,640 | 1034 | 3426 | 0.00 |
| r11.6 | 2,414,060 | 2,414,060 | 12,543 | 914,791 | 0.00 | 2,414,060 | 2522 | 5626 | 0.00 |
| r11.7 | 2,294,912 | 2,294,912 | 1 | 18 | 0.00 | 2,294,910 | 2 | 11 | 0.00 |
| r11.8 | 3,507,100 | 3,507,100 | 2 | 29 | 0.00 | 3,507,100 | 3 | 33 | 0.00 |
| r11.9 | 4,579,353 | 4,579,353 | 1 | 22 | 0.00 | 4,579,350 | 2 | 21 | 0.00 |
| r12.1 | 1,639,443 | 1,639,443 | 94 | 820 | 0.00 | 1,639,440 | 384 | 548 | 0.00 |
| **r12.2** | 3,396,050 | 3,335,491 | 36,000 | 848,792 | 1.78 | 3,396,050 | 19,362 | 10,149 | 0.00 |
| **r12.3** | 5,228,710 | 5,123,923 | 36,000 | 708,876 | 2.02 | 5,228,710 | 22,634 | 8779 | 0.00 |
| r12.4 | 2,303,557 | 2,303,557 | 27 | 189 | 0.00 | 2,303,560 | 100 | 85 | 0.00 |
| r12.5 | 4,669,799 | 4,669,799 | 47 | 313 | 0.00 | 4,669,800 | 74 | 68 | 0.00 |

**Table 14** continued

| Instance | $Z^*$ | CPLEX | | | | B&C&Filter | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | $Z^I_+$ | CPU | Nodes | Gap (%) | $Z^I_+$ | CPU | Nodes | Gap (%) |
| r12.6 | 7,100,019 | 7,100,019 | 28 | 326 | 0.00 | 7,100,020 | 29 | 31 | 0.00 |
| r12.7 | 7,635,270 | 7,635,270 | 2 | 7 | 0.00 | 7,635,270 | 1 | 4 | 0.00 |
| r12.8 | 10,067,742 | 10,067,742 | 2 | 4 | 0.00 | 10,067,700 | 1 | 5 | 0.00 |
| r12.9 | 11,967,768 | 11,967,768 | 2 | 0 | 0.00 | 11,967,800 | 0 | 1 | 0.00 |
| r13.1 | 142,947 | 142,947 | 0 | 28 | 0.00 | 142,947 | 0 | 10 | 0.00 |
| r13.2 | 263,800 | 263,800 | 56 | 693 | 0.00 | 263,800 | 55 | 190 | 0.00 |
| r13.3 | 365,836 | 365,836 | 128 | 1170 | 0.00 | 365,836 | 112 | 228 | 0.00 |
| r13.4 | 150,977 | 150,977 | 2 | 152 | 0.00 | 150,977 | 13 | 346 | 0.00 |
| **r13.5** | 282,682 | 278,591 | 36,000 | 18,513,092 | 1.45 | 282,682 | 384 | 3076 | 0.00 |
| **r13.6** | 406,789 | 393,351 | 36,000 | 13,001,246 | 3.30 | 406,790 | 1163 | 4752 | 0.00 |
| r13.7 | 208,088 | 208,088 | 19,575 | 4,660,581 | 0.00 | 208,088 | 3141 | 56,891 | 0.00 |
| **r13.8** | 444,826 | 436,994 | 36,000 | 6,152,916 | 1.76 | 444,826 | 18,787 | 140,596 | 0.00 |
| r13.9 | 697,966 | 682,249 | 36,000 | 4,530,796 | 2.25 | 695,248 | 36,000 | 141,712 | 0.39 |
| r14.1 | 403,414 | 403,414 | 10 | 592 | 0.00 | 403,414 | 13 | 95 | 0.00 |
| **r14.2** | 749,503 | 727,753 | 36,000 | 5,832,500 | 2.90 | 749,503 | 3193 | 4013 | 0.00 |
| **r14.3** | 1,063,097 | 1,024,169 | 36,000 | 5,060,355 | 3.66 | 1,063,100 | 11,921 | 5686 | 0.00 |
| r14.4 | 437,607 | 437,607 | 15 | 616 | 0.00 | 437,607 | 47 | 259 | 0.00 |
| **r14.5** | 849,163 | 826,985 | 36,000 | 4,495,951 | 2.61 | 849,162 | 11,695 | 16,927 | 0.00 |
| **r14.6** | 1,214,608 | 1,181,871 | 36,000 | 3,018,985 | 2.70 | 1,214,610 | 10,611 | 8821 | 0.00 |
| r14.7 | 668,217 | 664,472 | 36,000 | 1,682,889 | 0.56 | 667,661 | 36,000 | 121,353 | 0.08 |
| r14.8 | 1,613,429 | 1,581,485 | 36,000 | 945,701 | 1.98 | 1601,650 | 36,000 | 48,993 | 0.73 |
| r14.9 | 2,602,689 | 2,589,482 | 36,000 | 1,265,274 | 0.51 | 2,600,930 | 36,000 | 40,987 | 0.07 |

**Table 15** CPLEX and B&C&Filter on Class III-B instances (2)

| Instance | $Z^*$ | CPLEX | | | | B&C&Filter | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | $Z^l_+$ | CPU | Nodes | Gap (%) | $Z^l_+$ | CPU | Nodes | Gap (%) |
| r15.1 | 1,000,787 | 1,000,787 | 1153 | 1248 | 0.00 | 1,000,790 | 165 | 157 | 0.00 |
| r15.2 | 1,966,206 | 1,907,836 | 36,000 | 2,018,943 | 2.97 | 1,955,300 | 36,002 | 5400 | 0.55 |
| r15.3 | 2,883,855 | 2,754,195 | 36,000 | 1408,171 | 4.50 | 2,825,910 | 36,007 | 2455 | 2.01 |
| **r15.4** | 1,148,604 | 1,143,615 | 36,000 | 2,641,851 | 0.44 | 1,148,600 | 9862 | 9825 | 0.00 |
| r15.5 | 2,477,502 | 2,408,512 | 36,000 | 1,084,452 | 2.79 | 2,450,110 | 36,002 | 6055 | 1.11 |
| r15.6 | 3,829,646 | 3,705,682 | 36,000 | 666,920 | 3.24 | 3,768,370 | 36,006 | 3815 | 1.60 |
| r15.7 | 2,297,919 | 2,297,919 | 19,245 | 2,297,919 | 0.00 | 2,297,920 | 15,207 | 9260 | 0.00 |
| r15.8 | 5,573,413 | 5,573,413 | 70 | 623 | 0.00 | 5,573,410 | 651 | 429 | 0.00 |
| r15.9 | 8,696,932 | 8,696,932 | 10 | 45 | 0.00 | 8,696,930 | 21 | 21 | 0.00 |
| r16.1 | 136,161 | 136,161 | 1 | 3 | 0.00 | 136,161 | 0 | 5 | 0.00 |
| r16.2 | 239,500 | 239,500 | 237 | 2555 | 0.00 | 239,500 | 240 | 443 | 0.00 |
| r16.3 | 325,671 | 325,671 | 324 | 1838 | 0.00 | 325,671 | 578 | 868 | 0.00 |
| r16.4 | 138,532 | 138,532 | 1 | 37 | 0.00 | 138,532 | 1 | 20 | 0.00 |
| r16.5 | 241,801 | 241,801 | 34 | 396 | 0.00 | 241,801 | 32 | 52 | 0.00 |
| **r16.6** | 337,762 | 330,113 | 36,000 | 9,737,300 | 2.27 | 337,762 | 254 | 436 | 0.00 |
| r16.7 | 169,233 | 166,674 | 36,000 | 10,894,249 | 1.51 | 168,978 | 36,000 | 658,405 | 0.15 |
| r16.8 | 348,167 | 336,926 | 36,000 | 6,530,533 | 3.23 | 343,829 | 36,000 | 114,295 | 1.25 |
| r16.9 | 529,988 | 505,290 | 36,000 | 5,343,647 | 4.66 | 519,088 | 36,001 | 62,046 | 2.06 |
| r17.1 | 354,138 | 354,138 | 9 | 148 | 0.00 | 354,138 | 6 | 16 | 0.00 |

**Table 15** continued

| Instance | $Z^*$ | CPLEX | | | | B&C&Filter | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | $Z^l_+$ | CPU | Nodes | Gap (%) | $Z^l_+$ | CPU | Nodes | Gap (%) |
| **r17.2** | 645,488 | 630,727 | 36,000 | 4,480,835 | 2.29 | 645,488 | 5636 | 2008 | 0.00 |
| r17.3 | 910,518 | 859,704 | 36,000 | 3,734,717 | 5.58 | 910,503 | 36,002 | 8369 | 0.00 |
| r17.4 | 370,590 | 370,590 | 51 | 1237 | 0.00 | 370,590 | 142 | 366 | 0.00 |
| **r17.5** | 706,747 | 687,002 | 36,000 | 3,778,888 | 2.79 | 706746 | 7506 | 5655 | 0.00 |
| r17.6 | 1,019,758 | 96,2576 | 36,000 | 3,176,208 | 5.61 | 1,009,520 | 36,001 | 12,539 | 1.00 |
| r17.7 | 501,635 | 497,354 | 36,000 | 3,194,904 | 0.85 | 500,341 | 36,000 | 101,477 | 0.26 |
| r17.8 | 1,106,201 | 1,079,657 | 36,000 | 1,464,132 | 2.40 | 1,094,330 | 36,000 | 25,633 | 1.07 |
| r17.9 | 1,777,763 | 1,721,337 | 36,000 | 868,308 | 3.17 | 1,748,900 | 36,002 | 12723 | 1.62 |
| **r18.1** | 828,117 | 820,275 | 36,000 | 2,613,589 | 0.95 | 828,117 | 5217 | 3648 | 0.00 |
| r18.2 | 1,533,675 | 1,533,675 | 4374 | 1695 | 0.00 | 1,533,680 | 5848 | 696 | 0.00 |
| r18.3 | 2,174,276 | 2,092,060 | 36,000 | 1,481,947 | 3.78 | 2,153,910 | 36,004 | 2219 | 0.94 |
| r18.4 | 919,983 | 907,438 | 36,000 | 2,349,757 | 1.36 | 917,774 | 36,000 | 21,571 | 0.24 |
| r18.5 | 1,823,766 | 1,767,059 | 36,000 | 1,204,891 | 3.11 | 1,803,070 | 36,005 | 3671 | 1.13 |
| r18.6 | 2,714,335 | 2,592,175 | 36,000 | 876,199 | 4.50 | 2,649,110 | 36,011 | 2281 | 2.40 |
| r18.7 | 1,477,395 | 1,460,132 | 36,000 | 519,212 | 1.17 | 1,465,930 | 36,001 | 18,194 | 0.78 |
| r18.8 | 3,887,637 | 3,805,414 | 36,000 | 221,422 | 2.12 | 3,866,920 | 36,002 | 7535 | 0.53 |
| r18.9 | 6,361,906 | 6,286,310 | 36,000 | 211,150 | 1.19 | 6,347,330 | 36,002 | 8424 | 0.23 |
| Arithm | | | 18,849 | 2,146,781 | 1.22 | | 12,301 | 21,992 | 0.25 |
| ShiftG | | | 2855 | 52995 | 0.04 | | 1784 | 2135 | 0.00 |

## References


Aardal K (1998) Capacitated facility location: separation algorithms and computational experience. Math Program 81:149–175

Aardal K, Pochet Y, Wolsey LA (1995) Capacitated facility location: valid inequalities and facets. Math Oper Res 20:562–582

Achterberg T, Koch T, Martin A (2005) Branching rules revisited. Oper Res Lett 33:42–54

Adenso-Diaz B, Laguna M (2006) Fine-tuning of algorithms using fractional experimental designs and local search. Oper Res 54:99–114

Applegate D, Bixby RE, Chvatal V, Cook W (1995) Finding cuts in the TSP. Technical report 95-05, DIMACS technical report

Atamtürk A (2001) Flow pack facets of the single node fixed-charge flow polytope. Oper Res Lett 29:107–114

Atamtürk A (2002) On capacitated network design cut-set polyhedra. Math Program 92:425–437

Atamtürk A, Rajan D (2002) On splittable and unsplittable capacitated network design arc-set polyhedra. Math Program 92:315–333

Atamtürk A, Savelsbergh MWP (2005) Integer-programming software systems. Ann Oper Res 140:67–124

Balas E (1975) Facets of the knapsack polytope. Math Program 8:146–164

Barahona F (1996) Network design using cut inequalities. SIAM J Optim 6:823–837

Benichou M, Gauthier JM, Girodet P, Hentges G, Ribiere G, Vincent O (1971) Experiments in mixed-integer programming. Math Program 1:76–94

Bienstock D, Chopra S, Günlük O, Tsai CY (1998) Minimum cost capacity installation for multicommodity network flows. Math Program 81:177–199

Bienstock D, Günlük O (1996) Capacitated network design-polyhedral structure and computation. INFORMS J Comput 8:243–259

Chouman M, Crainic TG, Gendron B (2017) Commodity representations and cut-set-based inequalities for multicommodity capacitated fixed charge network design problem. Transp Sci 51:650–667

Codato G, Fischetti M (2006) Combinatorial Benders cuts for mixed-integer linear programming. Oper Res 54:756–766

Costa AM, Cordeau JF, Gendron B (2009) Benders, metric and cutset inequalities for multicommodity capacitated network design. Comput Optim Appl 42:371–392

Costa AM, Cordeau JF, Gendron B, Laporte G (2012) Accelerating Benders decomposition with heuristic master problem solutions. Pesqui Oper 32:3–20

Crainic TG, Gendreau M, Farvolden JM (2000) A simplex-based tabu search method for capacitated network design. INFORMS J Comput 12:223–236

Crainic TG, Frangioni A, Gendron B (1999) Telecommunications network planning. In: Soriano P, Sanso B (eds) Multicommodity capacitated network design. Kluwer Academics Publisher, Dordrecht, pp 1–19

Crainic TG, Frangioni A, Gendron B (2001) Bundle-based relaxation methods for multicommodity capacitated fixed charge network design. Discrete Appl Math 112:73–99

Crainic TG, Frangioni A, Gendron B, Guertin F (2009) OOBB: an object-oriented library for parallel branch-and-bound. In: Presented at the CORS/INFORMS international conference, Toronto, Canada, June 14–17 2009

Crainic TG, Gendreau M (2002) Cooperative parallel tabu search for capacitated network design. J Heuristics 8:601–627

Crainic TG, Gendron B, Hernu G (2004) A slope scaling/Lagrangean perturbation heuristic with long-term memory for multicommodity capacitated fixed-charge network design. J Heuristics 10:525–545

Frangioni A (2017) http://www.di.unipi.it/~frangio

Gabrel V, Knippel A, Minoux M (1999) Exact solution of multicommodity network optimization problems with general step cost functions. Oper Res Lett 25:15–23

Gendron B, Crainic TG (1994) Relaxations for multicommodity capacitated network design problems. Technical report, Publication CRT-945, Centre de recherche sur les transports, Université de Montréal

Gendron B, Larose M (2014) Branch-and-price-and-cut for large-scale multicommodity capacitated fixed-charge network design. EURO J Comput Optim 2:55–75

Ghamlouche I, Crainic TG, Gendreau M (2003) Cycle-based neighbourhoods for fixed charge capacitated multicommodity network design. Oper Res 51:655–667

Ghamlouche I, Crainic TG, Gendreau M (2004) Path relinking, cycle-based neighbourhoods and capacitated multicommodity network design. Ann Oper Res 131:109–133

Gu Z, Nemhauser GL, Savelsbergh MWP (1998) Lifted cover inequalities for 0–1 integer programs: computation. INFORMS J Comput 10:427–437
Gu Z, Nemhauser GL, Savelsbergh MWP (1999) Lifted cover inequalities for 0–1 integer programs: complexity. INFORMS J Comput 11:117–123
Gu Z, Nemhauser GL, Savelsbergh MWP (1999) Lifted flow cover inequalities for mixed 0–1 integer programs. Math Program 85:439–467
Günlük O (1999) A branch-and-cut algorithm for capacitated network design problems. Math Program 86:17–39
Hammer PL, Johnson EL, Peled UN (1975) Facets of regular 0–1 polytopes. Math Program 8:179–206
Hewitt M, Nemhauser GL, Savelsbergh MWP (2010) Combining exact and heuristic approches for the capacitated fixed-charge network flow problem. INFORMS J Comput 22:314–325
Holmberg K, Yuan D (2000) A Lagrangian heuristic based branch-and-bound approach for the capacitated network design problem. Oper Res 48:461–481
Hooker JN (2002) Logic, optimization, and constraint programming. INFORMS J Comput 14:295–321
Katayama N, Chen M, Kubo M (2009) A capacity scaling heuristic for the multicommodity capacitated network design problem. J Comput Appl Math 232:90–101
Kliewer G, Timajev L (2005) Relax-and-cut for capacitated network design. In Proceedings of algorithms-ESA 2005: 13th annual european symposium on algorithms, pp 47–58. Lecture notes in computer science 3369
Leung JMY, Magnanti TL (1989) Valid inequalities and facets of the capacitated plant location problems. Math Program 44:271–291
Louveaux Q, Wolsey LA (2007) Lifting, superadditivity, mixed integer rounding and single node flow sets revisited. Ann Oper Res 153:47–77
Magnanti TL, Mirchandani PB, Vachani R (1993) The convex hull of two core capacitated network design problems. Math Program 60:233–250
Magnanti TL, Mirchandani PB, Vachani R (1995) Modeling and solving the two-facility capacitated network loading problem. Oper Res 43:142–157
Martello S, Toth P (1997) Upper bounds and algorithms for hard 0–1 knapsack problems. Oper Res 45:768–778
Ortega F, Wolsey LA (2003) A branch-and-cut algorithm for the single commodity uncapacitated fixed charge network flow problem. Networks 41:143–158
Padberg MW, Van Roy TJ, Wolsey LA (1985) Valid linear inequalities for fixed charge problems. Oper Res 33:842–861
Raack C, Koster AMCA, Orlowski S, Wessäly R (2011) On cut-based inequalities for capacitated network design polyhedra. Networks 57:141–156
Rodríguez-Martín I, Salazar-González JJ (2010) A local branching heuristic for the capacitated fixed-charge network design problem. Comput Oper Res 37:575–581
Savelsbergh MWP (1994) Preprocessing and probing techniques for mixed integer programming problems. ORSA J Comput 6:445–445
Sellmann M, Kliewer G, Koberstein A (2002) Lagrangian cardinality cuts and variable fixing for capacitated network design. In: Proceedings of algorithms-ESA 2002: 10th annual european symposium on algorithms, pp 845–858. Lecture notes in computer science 2461
Van Roy TJ, Wolsey LA (1987) Solving mixed integer programming problems using automatic reformulation. Oper Res 35:45–57
Wolsey LA (1975) Faces of linear inequalities in 0–1 variables. Math Program 8:165–178