CIRRELT

Centre interuniversitaire de recherche
sur les réseaux d'entreprise, la logistique et le transport

Interuniversity Research Centre
on Enterprise Networks, Logistics and Transportation

The Impact of Filtering in a Branch-and-Cut Algorithm for Multicommodity Capacitated Fixed Charge Network Design

Mervat Chouman
Teodor Gabriel Crainic
Bernard Gendron

July 2014

CIRRELT-2014-35

Bureaux de Montréal :
Université de Montréal
Pavillon André-Aisenstadt
C.P. 6128, succursale Centre-ville
Montréal (Québec)
Canada H3C 3J7
Téléphone : 514 343-7575
Télécopie : 514 343-7121

Bureaux de Québec :
Université Laval
Pavillon Palasis-Prince
2325, de la Terrasse, bureau 2642
Québec (Québec)
Canada G1V 0A6
Téléphone : 418 656-2073
Télécopie : 418 656-2624

www.cirrelt.ca

The Impact of Filtering in a Branch-and-Cut Algorithm for Multicommodity Capacitated Fixed Charge Network Design

Mervat Chouman[1], Teodor Gabriel Crainic[2,3], Bernard Gendron[2,4,*]

[1] Department of Operation and Information Management, Effat University, Jeddah, Kingdom of Saudi Arabia

[2] Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT)

[3] Department of Management and Technology, Université du Québec à Montréal, P.O. Box 8888, Station Centre-Ville, Montréal, Canada H3C 3P8

[4] Department of Computer Science and Operations Research, Université de Montréal, P.O. Box 6128, Station Centre-Ville, Montréal, Canada H3C 3J7

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 significantly 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.

Keywords. Mixed-integer programming (MIP), multicommodity capacitated fixed charge network design (MCND), branch-and-cut (B&C), filtering.

Acknowledgements. The authors express their gratitude to Geneviève Hernu and Serge Bisaillon, analysts at CIRRELT. Their efforts, high professionalism and dedication in implementing and testing the innumerable variants of the methods and algorithms presented in this paper were instrumental in achieving our results. While working on this project, T.G. Crainic was the NSERC Industrial Research Chair on Logistics Management, ESG UQAM, and Adjunct Professor with the Department of Computer Science and Operations Research, Université de Montréal, and the Department of Economics and Business Administration, Molde University College, Norway. Funding for this project has been provided by the Natural Sciences and Engineering Council of Canada (NSERC), through its Industrial Research Chair and Discovery Grants programs, by the partners of the Chair, CN, Rona, Alimentation Couche-Tard and the ministère des Transports du Québec. We also gratefully acknowledge the support of Fonds de recherche du Québec through their infrastructure grants and of Calcul Québec and Compute Canada through access to their high-performance computing infrastructure.

---

* Corresponding author: Bernard.Gendron@cirrelt.ca

Dépôt légal – Bibliothèque et Archives nationales du Québec
Bibliothèque et Archives Canada, 2014

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 [14]. In this last paper, several valid inequalities, separation routines and modelling 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 [14], 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 [37], while in MIP, they arise within *preprocessing* routines [49] and domain reduction tests based on reduced-cost information. As such, filtering methods are an integral part of state-of-the-art MIP solvers [8].

To the best of our knowledge, along with the cutting-plane approach [14] 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 [16, 17], column generation [27] and Lagrangian relaxation approaches [19, 20, 26, 36, 39, 50]. Heuristic methods have also been proposed for computing feasible solutions [18, 22, 23, 28, 29, 35, 38, 48]. Typically, instances with few commodities, i.e., origin-destination pairs with positive demand, (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 [14], 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 (optimal solutions could be obtained by running the MIP solver in parallel for a long period of time). 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 [1, 2, 6, 7, 10, 12, 13, 25, 33, 40, 42, 43, 45, 47], there are no systematic studies regarding the behaviour 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 [14] and by the latest developments in MIP software tools [3, 8]. 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 [14] with an adaptation of the reliability branching rule introduced in [3].

- 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 significant 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 Section 4 summarizes the overall B&C algorithm. In Section 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 Section 2.1. In Section 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 Section 2.3, while the branching rule used in the B&C algorithm is described in Section 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 to 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 set (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 th 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 [14]. 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. [14] also use flow cover/pack inequalities [5, 32, 41, 46, 51]. 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 [14, 19, 26]. 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 [9, 34, 52] are based on the following definition: $C \subseteq (S, \bar{S})$ is a *cover* if the total capacity of the arcs in $(S, \bar{S}) \setminus C$ does not cover the demand, i.e., $\sum_{(i,j)\in(S,\bar{S})\setminus C} u_{ij} < d_{(S,\bar{S})}$. 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 [44]. 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 [14], 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 [14]). 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 [14, 30, 31]. 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

The cutting-plane procedure is a simpler variant of the method described in [14]. 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 δ 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 [14]):

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. The two types of cuts are kept in memory in two different cut pools. 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 from the two cut pools, by considering the LP solution from its parent and by adding global and local cuts violated by this solution (the local cuts are obtained by tracing them back in the local cut pool for all the ancestors of the node).

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 Section 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* [3], a rule that has often proved successful in solving general MIP models. Reliability branching 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 Δ_a^0 and Δ_a^1, respectively. We also define the corresponding *per unit* increase in the LP bounds from the parent to its children as follows: $\rho_a^h = \frac{\Delta_a^h}{g_a^h}$, $h = 0, 1$, where $g_a^0 = \bar{y}_a$ and $g_a^1 = 1 - \bar{y}_a$. 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_a^0 and n_a^1 times; we can then define the *average per unit* increase in the LP bounds from the parent node to its children as $\bar{\rho}_a^h$, $h = 0, 1$ (i.e., the average value of ρ_a^h over the n_a^h 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 [11] is based on computing and storing the values $\bar{\rho}_a^h$, $h = 0, 1$, for each arc a. This branching rule selects the free arc a^* that achieves the maximum over $\bar{A}_{01}$ of the quantities $\min(g_a^0\bar{\rho}_a^0, g_a^1\bar{\rho}_a^1)$, i.e.,

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

In this formula, $g_a^h\bar{\rho}_a^h$, $h = 0, 1$, 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., Δ_a^h, $h = 0, 1$, 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 [4], which is based on computing estimates of the LP bound increases Δ_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 Δ_a^0 and Δ_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^* that achieves the maximum over $\bar{A}_{01}$ of the quantities $\min(\Delta_a^0, \Delta_a^1)$, i.e.,

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

Strong branching is usually very effective in reducing the size of the B&C tree, but the branching procedure itself is often too time-consuming for this reduction in tree size to pay off. Pseudo-cost branching is an interesting alternative, since it shows a good tradeoff between the computing efficiency of the branching rule and the size of the B&C tree. However, as mentioned above, no values of the LP bound increases are available at the beginning of the exploration; hence, the initial estimates are not reliable. The idea behind reliability branching [3] is therefore 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 to 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.

We adopt reliability branching in our implementation, which 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., $\min(n^0_{a^*}, n^1_{a^*}) < \eta$, then the arcs $a \in \bar{A}_{01}$ with unreliable pseudo-costs are sorted in non-increasing order of $\min(g^0_a\bar{\rho}^0_a, g^1_a\bar{\rho}^1_a)$. Using that particular order, we keep as a candidate for branching the arc a^* that achieves so far the maximum of $\min(\Delta^0_a, \Delta^1_a)$, where Δ^h_a, $h = 0, 1$, are computed with the dual simplex method (limited to 100 iterations). If that candidate is not updated for λ successive attempts, we select a^* for branching. The branching procedure thus performs the following steps (we use $\eta = 8$ and $\lambda = 4$ as in [3]):

1. $a^* \leftarrow \arg\max_{a \in \bar{A}_{01}} \{\min(g^0_a\bar{\rho}^0_a, g^1_a\bar{\rho}^1_a)\}$.
2. If $\min(n^0_{a^*}, n^1_{a^*}) \geq \eta$, then stop.
3. Let $m \leftarrow 0$, $s^* \leftarrow 0$ and sort the arcs of $(\bar{A}_{01} \cap \{a \in A | \min(n^0_a, n^1_a) < \eta\})$ in non-increasing order of $\min(g^0_a\bar{\rho}^0_a, g^1_a\bar{\rho}^1_a)$.
4. For all $a \in (\bar{A}_{01} \cap \{a \in A | \min(n^0_a, n^1_a) < \eta\})$ (sorted):
   (a) $m \leftarrow m + 1$.
   (b) Compute Δ^0_a and update n^0_a and $\bar{\rho}^0_a$ (unless the LP is infeasible); if $Z^l + \Delta^0_a \geq Z^*$, 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 Δ^1_a and update n^1_a and $\bar{\rho}^1_a$; if $Z^l + \Delta^1_a \geq Z^*$, 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 $\min(\Delta^0_a, \Delta^1_a) > s^*$, then $a^* \leftarrow a$, $s^* \leftarrow \min(\Delta^0_a, \Delta^1_a)$ 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 $Z^l + \Delta^h_a$ 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 Sections 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 Sections 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* [26]. 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:

$$Z^l_{LR} = \sum_{k\in K}(\pi^k_{D(k)} - \pi^k_{O(k)})d^k + \min \sum_{(i,j)\in A_{01}\cup A_1} \left\{ \sum_{k\in K}(c_{ij} + \pi^k_i - \pi^k_j)x^k_{ij} + f_{ij}y_{ij} \right\}$$

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

$$0 \le x^k_{ij} \le b^k_{ij}y_{ij}, \quad (i,j) \in A_{01}\cup A_1, k \in K,$$

$$y_{ij} = 1, \quad (i,j) \in A_1,$$

$$y_{ij} \in \{0,1\}, \quad (i,j) \in A_{01}.$$

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

$$v_{ij} = \min\{\sum_{k\in K} \tilde{c}^k_{ij}x^k_{ij} \mid \sum_{k\in K} x^k_{ij} \le u_{ij};\ 0 \le x^k_{ij} \le b^k_{ij},\ k \in K\},$$

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

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

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 $Z^l_{LR} \ge Z^*$, 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 to 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 $Z^l_{LR} + |\tilde{f}_{ij}| \ge Z^*$, 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 to 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 [36, 39, 50], 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}, $\bar{c}^k_{ij}$, to perform bound reduction on these variables. The basic idea is the following: assume we add the constraint

$x_{ij}^k > a_{ij}^k$ to the LP relaxation and that the resulting lower bound exceeds Z^*. We can then conclude that the constraint $x_{ij}^k \leq a_{ij}^k$ is valid. In order to compute a_{ij}^k, we use the following result.

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

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

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 $0 < \overline{x}_{ij}^k < b_{ij}^k$, then $\overline{c}_{ij}^k = 0$ and, in this case, the LP relaxation lower bound remains the same when we increase x_{ij}^k further. Therefore, for the LP relaxation lower bound to increase and exceed Z^* when we add the constraint $x_{ij}^k > a_{ij}^k$, we must have $\overline{x}_{ij}^k = 0$, $\overline{c}_{ij}^k > 0$ and $Z^l + \overline{c}_{ij}^k b_{ij}^k > Z^*$. Since any optimal solution must satisfy $Z^l + \overline{c}_{ij}^k x_{ij}^k \leq Z^*$, we conclude that $x_{ij}^k \leq a_{ij}^k < b_{ij}^k$, where

$$a_{ij}^k = \frac{Z^* - Z^l}{\overline{c}_{ij}^k}.$$

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

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

■

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.

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

$$a_{ij}^k = \frac{Z^* - Z_{LR}^l - \tilde{f}_{ij}(1 - \tilde{y}_{ij})}{\tilde{c}_{ij}^k}.$$

Proof. We consider two cases. First, let us assume that $\tilde{y}_{ij} = 1$. We note that if $\tilde{x}_{ij}^k > 0$, then $\tilde{c}_{ij}^k \leq 0$, and in this case the Lagrangian lower bound can only decrease when we increase x_{ij}^k further. Therefore, for the Lagrangian lower bound to increase and exceed Z^* when we add the constraint $x_{ij}^k > a_{ij}^k$, we must have $\tilde{x}_{ij}^k = 0$, $\tilde{c}_{ij}^k > 0$ and $Z_{LR}^l + \tilde{c}_{ij}^k b_{ij}^k > Z^*$. Since any optimal solution must satisfy $Z_{LR}^l + \tilde{c}_{ij}^k x_{ij}^k \leq Z^*$, we conclude that $x_{ij}^k \leq a_{ij}^k < b_{ij}^k$, where

$$a_{ij}^k = \frac{Z^* - Z_{LR}^l}{\tilde{c}_{ij}^k}.$$

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

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

■

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. In particular, when looking for violated strong inequalities in the cutting-plane procedure, we use the inequalities $x^k_{ij} \leq a^k_{ij} y_{ij}$ instead of $x^k_{ij} \leq b^k_{ij} y_{ij}$. When violated inequalities of this type are generated, they are stored in the local cut pool, so that they can be activated when backtracking is performed.

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 *multicommodity flow system*, noted MF:

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

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

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

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

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 Section 3.6).

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

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

Proof. Only the arcs in A_0 can incur infeasibility. Therefore, to obtain a feasible solution, at least one arc in A_0 must be opened, i.e., $\sum_{(i,j)\in A_0} y_{ij} \geq 1$. ■

Inequality (14) is a *combinatorial Benders cut* [15], which has the general form $\sum_{(i,j)\in A_1}(1-y_{ij}) + \sum_{(i,j)\in A_0} y_{ij} \geq 1$, which 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 [16, 17]. 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 preprocessing operations that attempt to detect infeasibility just before calling the cutting-plane procedure. These 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 there is an arc in $A_0 \cap A_{01}^{'}$, it must be opened in any feasible solution, i.e., we add the constraint $y_{ij} = 1$, $(i,j) \in A_0 \cap A_{01}^{'}$. Similar 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 preprocessing-based filtering methods aiming to detect as early as possible any infeasibility that might occur as a result of closing too many arcs. In particular, we note that two sources of infeasibility can be identified: first, for some commodity k, there is no longer any path connecting $O(k)$ to $D(k)$; second, the overall capacity is not sufficient to satisfy the demand for at least one commodity. Next, we present filtering methods based on these two sources of infeasibility, the connectivity-based and capacity-based filtering methods, which are also applied before calling the cutting-plane procedure.

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 to 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 to 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 Section 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.

We now explain how these tests can be performed in time $O(|V||A|)$ 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, therefore identifying all the arcs that can be accessed from i. 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 traversal being performed in time $O(|A|)$) 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 $D(k)$. These informations suffice to perform the fathoming and filtering tests outlined above.

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:

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

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* [26]. 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 $Z^l_{MC}(i,j)$, then we verify if $Z^l_{MC}(i,j) \geq Z^*$, in which case we can conclude again that arc (i,j) is must be opened in any optimal solution to the MCND.

Tu 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 $\sum_{k \in K} \widehat{x}^k_{ij} = 0$, 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, ϵ 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 Section 5, we show results for $\gamma = 0.5$, 0.7 and 0.9). 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.

The strict conditions that trigger the execution of capacity-based filtering limit the number of multicommodity flow problems to be solved, but each such problem can be computationally cumbersome, especially for instances with many commodities. Several improvements are performed to quickly compute the multicommodity flow problems and to reoptimize them efficiently when trying to open some free arcs. Using the fact that costs and capacities do not depend on the commodities, we aggregate the commodities by origin: all commodities that share the same origin are aggregated into a single commodity. Any new, aggregated, commodity has one origin and multiple destinations. We solve the LP dual of the resulting multicommodity flow problem. An initial dual solution is obtained by computing the shortest path trees from the origin to the destinations for each aggregated commodity, with arc lengths equal to c_{ij} for $(i,j) \in A_1$ and to $c_{ij} + f_{ij}/u_{ij}$ for $(i,j) \in A_{01}$. The shortest path lengths provide the initial values of the dual variables associated to the flow conservation equations, while the dual variables of the capacity constraints are initialized to 0. The dual simplex method is then used to solve the LP dual. As soon as the lower bound exceeds Z^*, we can stop the dual simplex iterations. To try setting a free arc (i,j) to open, we must first set it to closed, which is done by changing the length of the arc from $c_{ij} + f_{ij}/u_{ij}$ to a very large value (we use $1000 \times (c_{ij} + f_{ij})$). This amounts to modify the right-hand side of a single constraint in the LP dual model. Hence, by using the dual simplex method on the LP dual, reoptimization can be performed efficiently.

4 Overview of the Branch-and-Cut Algorithm

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

4.1 Computation of Upper Bounds

As mentioned in the Introduction, very good upper bounds are obtained by the heuristic methods proposed in the literature [18, 22, 23, 28, 29, 35, 38, 48]. In our tests, reported in Section 5, we use as initial upper bound the value $(1 + \xi) \times Z$, where Z is the optimal value and ξ 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 property 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 Section 3.4.

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

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

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

Proof. If $\widehat{x}$ is feasible to MF, then $(\widehat{x}, \widehat{y})$ with $\widehat{y}_{ij} = \lceil \sum_{k \in K} \widehat{x}^k_{ij} / u_{ij} \rceil$, $(i, j) \in A$, is also feasible to the MCND, since $\widehat{y}_{ij} = \lceil \sum_{k \in K} \widehat{x}^k_{ij} / u_{ij} \rceil = 1$, if $0 < \sum_{k \in K} \widehat{x}^k_{ij} \leq u_{ij}$, and 0, otherwise. Therefore, $Z(\widehat{x}) = \sum_{k \in K} \sum_{(i,j) \in A} c_{ij} \widehat{x}^k_{ij} + \sum_{(i,j) \in A} f_{ij} \widehat{y}_{ij}$, the objective function value of the feasible solution $(\widehat{x}, \widehat{y})$, 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 $\bar{y}_{ij} \geq \sum_{k \in K} \overline{x}^k_{ij} / u_{ij}$, for each $(i, j) \in A$. By applying the previous proposition, we obtain an upper bound on the optimal value of the MCND:

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

■

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.

The motivation for this search strategy is to combine the advantages of depth-first and best-first. As in depth-first, we select the next node to evaluate among the two children that have just been generated by the branching operation. This allows for fast reoptimization of the LPs generated by the cutting-plane procedure at the child node, using the dual simplex method. As in best-first, we tend to select the node with the smallest lower bound, which ensures that the most promising branches are selected first and that the global lower bound found by the B&C algorithm gradually improves during the course of the algorithm.

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 Section 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 Section 2.4. The next current node is selected among the two children according to the rule described in Section 4.2. Step 4 verifies the stopping condition $\mathcal{L} = \emptyset$ and, if it is not satisfied, it performs backtracking as discussed in Section 4.2. We also stop the algorithm when a time limit has been reached. In addition to the node pool, we store and update the pools of global cuts and local cuts, as presented in Section 2.3. 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 (Section 3.1 and step 2i), LR-based reduced cost fixing (Section 3.2 and step 2k), flow upper bound reduction (Section 3.3 and steps 2j and 2(k)iv), combinatorial Benders cuts (Section 3.4 and steps 2(c)ii, 2e and 2h), connectivity-based filtering (Section 3.5 and step 2b) and capacity-based filtering (Section 3.6 and step 2c). The cutting-plane procedure performed at step 2d follows the developments in Section 2.3. Finally, the computation and update of upper bounds, steps 2(c)iii and 2f, correspond to Section 4.1.

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 [24]) used in several papers on the $MCND$ (for instance [28, 35, 39]) and described in detail in [20]. 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 [24]) consists of 31 problem instances with many commodities compared to the number of nodes, while Class II (the "C+" instances in [24]) contains 12 problem instances with few commodities compared to the number of nodes. Class III (the "R" instances in [24]) 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 [21], using CPLEX version 12.5.1.0 as the LP solver. The code was compiled with g++ 4.4.7 and performed on an Intel Xeon X5675, operating at 3,07 GHz, in single-threaded mode. All instances were solved with a time limit of 10 hours. 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 $\text{Gap} = 100 \times (Z^* - Z^l_+)/Z^*$.

We first present the results obtained with different configurations of the filtering parameters. We divide the instances into two classes: the instances solved by all parameter configurations within the time limit of 10 hours (for which Gap = 0) and those unsolved by any of the parameter configurations after the time limit of 10 hours (for which Gap > 0). Tables 2 and 3 show the results obtained on these two classes of instances, respectively, for four parameter configurations. 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

Class I (31)	**Class II** (12)	**Class III-A** (72)	**Class III-B** (81)
$\|V\|, \|A\|, \|K\|$	$\|V\|, \|A\|, \|K\|$	$\|V\|, \|A\|, \|K\|$	$\|V\|, \|A\|, \|K\|$
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)

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

measures are computed (arithmetic averages are computed for each performance measure). The second column identifies the filtering parameters activated in each configuration. 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 in two of the configurations and flow upper bound reduction is activated in only one configuration. The next columns give the values for the performance measures. For the solved instances, we show the CPU time in seconds, "CPU", and the number of nodes, "Nodes". For the unsolved instances, we show the number of nodes and the relative gap in percentage, "Gap".

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, three instances unsolved by the configuration with no filtering (one more in each of Classes I, II and III-B) are solved within the time limit of 10 hours, so that the number of solved instances increases from 148 (with no filtering) to 151. This is already a clear indication of the positive effect of LP-based reduced cost fixing. The three additional solved instances are not reported in any of the two tables, so that the comparison for both classes of instances, solved and unsolved, rely 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. The only exception to this observation is when flow upper bound reduction is performed on Class III-B instances, where we can observe a slight increase in the number of nodes. In general, the most significant reduction in the number of nodes is observed for LP-based reduced cost fixing and a notable reduction in the number of nodes is also observed when LR-based reduced cost fixing is performed on Class I instances. These reductions in the number of nodes do not always translate into reductions in the CPU time. This is true in particular when flow upper bound reduction is performed, where the CPU time increases, irrespective of the class of instances. The results in Table 3 confirm that flow upper bound reduction consumes a significant amount of time, at least for the instances in Classes I and III-B, for which, with the attained limit of 10 hours of CPU time, the number of nodes is significantly reduced when flow upper bound reduction is performed. This observation is not verified, however, for the three instances in Class II. In general, the

Class	Parameters	CPU	Nodes
I (11)	–	4923	5025
	LP-fix	2839	4802
	LP-fix, LR-fix	2713	4661
	LP-fix, LR-fix, Flow	3204	4457
II (8)	–	2230	18600
	LP-fix	1270	17658
	LP-fix, LR-fix	1324	17658
	LP-fix, LR-fix, Flow	1436	17237
III-A (72)	–	8	325
	LP-fix	7	282
	LP-fix, LR-fix	7	281
	LP-fix, LR-fix, Flow	8	278
III-B (57)	–	3845	5262
	LP-fix	3779	3762
	LP-fix, LR-fix	3870	3724
	LP-fix, LR-fix, Flow	4046	3843

Table 2: Results with duality-based filtering on instances solved by all configurations

Class	Parameters	Nodes	Gap
I (19)	–	9124	0.82
	LP-fix	9216	0.81
	LP-fix, LR-fix	8805	0.84
	LP-fix, LR-fix, Flow	7858	0.84
II (3)	–	436214	4.89
	LP-fix	421922	4.89
	LP-fix, LR-fix	390425	4.91
	LP-fix, LR-fix, Flow	394126	4.88
III-B (23)	–	45951	1.10
	LP-fix	54725	1.10
	LP-fix, LR-fix	56752	1.06
	LP-fix, LR-fix, Flow	44962	1.10

Table 3: Results with duality-based filtering on instances unsolved by any of the configurations

Class	Parameters	CPU	Nodes
I (11)	*LP-fix, LR-fix*	2713	4661
	LP-fix, LR-fix, Cap ($\gamma = 0.5$)	4406	4659
	LP-fix, LR-fix, Cap ($\gamma = 0.7$)	2245	4661
	LP-fix, LR-fix, Cap ($\gamma = 0.9$)	2516	4661
II (8)	*LP-fix, LR-fix*	1324	17658
	LP-fix, LR-fix, Cap ($\gamma = 0.5$)	1216	17641
	LP-fix, LR-fix, Cap ($\gamma = 0.7$)	1361	17658
	LP-fix, LR-fix, Cap ($\gamma = 0.9$)	1313	17658
III-A (72)	*LP-fix, LR-fix*	7	281
	LP-fix, LR-fix, Cap ($\gamma = 0.5$)	6	281
	LP-fix, LR-fix, Cap ($\gamma = 0.7$)	6	281
	LP-fix, LR-fix, Cap ($\gamma = 0.9$)	6	281
III-B (57)	*LP-fix, LR-fix*	3870	3724
	LP-fix, LR-fix, Cap ($\gamma = 0.5$)	3878	3723
	LP-fix, LR-fix, Cap ($\gamma = 0.7$)	3373	3724
	LP-fix, LR-fix, Cap ($\gamma = 0.9$)	3586	3724

Table 4: Results with capacity-based filtering on instances solved by all configurations

results in Tables 2 and 3 indicate that LP-based reduced cost fixing has a clear positive impact on the overall performance, while flow upper bound reduction is too time-consuming and has a negative impact on the overall performance. The impact of LR-based reduced cost fixing is less clear, but we note that it allows to reduce the final gap for some hard instances in Class III-B and when its impact is negative, it is only slightly so. Thus, for the remaining tested parameter configurations, we activate both *LP-fix* and *LR-fix*, but not *Flow*.

Tables 4 and 5 show the results obtained when the parameter *Cap* is activated, in addition to *LP-fix* and *LR-fix*. We report the results with three values of the parameter γ, which controls when capacity-based filtering is performed depending on the proportion of design variables fixed to 0: $\gamma = 0.5, 0.7, 0.9$. To ease the comparison, we also report the results when *Cap* is not activated, which were already shown in Tables 2 and 3. The same instances are also used, 148 solved ones and 45 unsolved ones. The remaining three instances in the set of 196 instances are solved by all configurations, but the results are not reported to facilitate the comparison.

The results in Table 4 show that capacity-based filtering has a marginal impact on the number of nodes, but can incur significant variations in the CPU time. In particular, a significant increase is observed with $\gamma = 0.5$ on Class I instances and significant reductions are observed with $\gamma = 0.7$ on Class I and III-B instances. Reductions in the CPU time are also observed with $\gamma = 0.9$ for all classes of instances. The results in Table 5 show that reductions in the gap are obtained with capacity-based filtering, irrespective of the value of γ. The most significant reductions are obtained with $\gamma = 0.5$ for instances in Classes I and II, and with $\gamma = 0.7$ and 0.9 for instances in Class III-B. In this last case, $\gamma = 0.9$ gives a better overall performance, since the number of nodes is increased, given the fixed amount of 10 hours of CPU time. Overall, the results in Tables 4 and 5 point to the conclusion that

Class	Parameters	Nodes	Gap
I (19)	*LP-fix, LR-fix*	9124	0.82
	LP-fix, LR-fix, Cap ($\gamma = 0.5$)	10429	0.78
	LP-fix, LR-fix, Cap ($\gamma = 0.7$)	9334	0.81
	LP-fix, LR-fix, Cap ($\gamma = 0.9$)	8979	0.81
II (3)	*LP-fix, LR-fix*	436214	4.91
	LP-fix, LR-fix, Cap ($\gamma = 0.5$)	432013	4.88
	LP-fix, LR-fix, Cap ($\gamma = 0.7$)	408335	4.91
	Lp-fix, LR-fix, Cap ($\gamma = 0.9$)	421665	4.90
III-B (23)	*LP-fix, LR-fix*	45951	1.10
	LP-fix, LR-fix, Cap ($\gamma = 0.5$)	38457	1.05
	LP-fix, LR-fix, Cap ($\gamma = 0.7$)	53344	1.03
	LP-fix, LR-fix, Cap ($\gamma = 0.9$)	57609	1.03

Table 5: Results with capacity-based filtering on instances unsolved by any of the configurations

capacity-based filtering, with appropriate values of the parameter γ, has a positive effect on the overall performance of the algorithm. For the remaining tested parameter configurations, we thus activate *Cap*. Furthermore, we select the value $\gamma = 0.9$, since it is the only one of the three tested values that shows performance improvements on all classes of instances, with reductions in the CPU time for solved instances and reductions in the gap for unsolved instances.

Tables 6 and 7 show the results obtained by performing feasibility-based filtering, in addition to *LP-fix*, *LR-fix* and *Cap*. 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 were also displayed in Tables 4 and 5. 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 alone. However, the final gap for this instance is 0.03%, which is negligible. We can thus consider that the four configurations reported in Tables 6 and 7 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 6 show that the addition of Benders cuts has a negligible impact on Class I instances, a negative impact on Class II instances and a positive impact on Class III-B instances. In contrast, connectivity-based filtering has a positive impact on all classes of instances, whether *Benders* is activated or not. Overall, the best configuration is obtained by activating only connectivity-based filtering, with significant reductions in the CPU time, even though the number of nodes can be slightly increased. The results in Table 7 show decreases in the gap for all configurations that include feasibility-based filtering. The most significant reductions in the gap are obtained when only *Conn* is activated. In this case, the reductions in the gap are accompanied by significant increases in the number of nodes. The results in Tables 6 and 7 point to the general conclusions that the addition of combinatorial

Class	Parameters	CPU	Nodes
I (11)	*LP-fix, LR-fix, Cap* ($\gamma = 0.9$)	2516	4661
	LP-fix, LR-fix, Cap ($\gamma = 0.9$)*, Benders*	2551	4655
	LP-fix, LR-fix, Cap ($\gamma = 0.9$)*, Conn*	2239	4789
	LP-fix, LR-fix, Cap ($\gamma = 0.9$)*, Benders, Conn*	2575	4785
II (8)	*LP-fix, LR-fix, Cap* ($\gamma = 0.9$)	1313	17658
	LP-fix, LR-fix, Cap ($\gamma = 0.9$)*, Benders*	1831	18074
	LP-fix, LR-fix, Cap ($\gamma = 0.9$)*, Conn*	1113	15307
	LP-fix, LR-fix, Cap ($\gamma = 0.9$)*, Benders, Conn*	1539	15320
III-A (72)	*LP-fix, LR-fix, Cap* ($\gamma = 0.9$)	6	281
	LP-fix, LR-fix, Cap ($\gamma = 0.9$)*, Benders*	6	279
	LP-fix, LR-fix, Cap ($\gamma = 0.9$)*, Conn*	6	287
	LP-fix, LR-fix, Cap ($\gamma = 0.9$)*, Benders, Conn*	7	284
III-B (57)	*LP-fix, LR-fix, Cap* ($\gamma = 0.9$)	3586	3724
	LP-fix, LR-fix, Cap ($\gamma = 0.9$)*, Benders*	3339	3720
	LP-fix, LR-fix, Cap ($\gamma = 0.9$)*, Conn*	2665	3758
	LP-fix, LR-fix, Cap ($\gamma = 0.9$)*, Benders, Conn*	3182	3748

Table 6: Results with feasibility-based filtering on instances solved by all configurations

Class	Parameters	Nodes	Gap
I (19)	*LP-fix, LR-fix, Cap* ($\gamma = 0.9$)	9124	0.82
	LP-fix, LR-fix, Cap ($\gamma = 0.9$)*, Benders*	10634	0.78
	LP-fix, LR-fix, Cap ($\gamma = 0.9$)*, Conn*	13482	0.73
	LP-fix, LR-fix, Cap ($\gamma = 0.9$)*, Benders, Conn*	12063	0.76
II (3)	*LP-fix, LR-fix, Cap* ($\gamma = 0.9$)	436214	4.91
	LP-fix, LR-fix, Cap ($\gamma = 0.9$)*, Benders*	397293	4.88
	LP-fix, LR-fix, Cap ($\gamma = 0.9$)*, Conn*	451612	4.84
	LP-fix, LR-fix, Cap ($\gamma = 0.9$)*, Conn, Benders*	403726	4.82
III-B (23)	*LP-fix, LR-fix, Cap* ($\gamma = 0.9$)	57609	1.03
	LP-fix, LR-fix, Cap ($\gamma = 0.9$)*, Benders*	63703	0.95
	LP-fix, LR-fix, Cap ($\gamma = 0.9$)*, Conn*	64376	0.89
	LP-fix, LR-fix, Cap ($\gamma = 0.9$)*, Benders, Conn*	60244	0.94

Table 7: Results with feasibility-based filtering on instances unsolved by any of the configurations

Class	Method	CPU	Nodes	Gap	Instances
I (11)	*B&C&Filter*	2239	4789	0.00	0
	B&C	4923	5025	0.00	0
	CPLEX-Weak	9830	412986	0.24	-3
	CPLEX-Strong	7054	224615	0.17	-2
II (8)	*B&C&Filter*	1113	15307	0.00	0
	B&C	2230	18600	0.00	0
	CPLEX-Weak	58	1225	0.00	0
	CPLEX-Strong	52	1151	0.00	0
III-A (72)	*B&C&Filter*	6	287	0.00	0
	B&C	8	325	0.00	0
	CPLEX-Weak	16	9178	0.00	0
	CPLEX-Strong	4	1421	0.00	0
III-B (57)	*B&C&Filter*	2665	3748	0.00	0
	B&C	3845	5262	0.00	0
	CPLEX-Weak	12917	1710457	0.69	-18
	CPLEX-Strong	12687	1290286	0.71	-18

Table 8: Comparison with CPLEX on instances solved by the B&C method

Benders cuts generally has a positive impact on the overall performance of the algorithm, but that significantly better results are obtained when connectivity-based filtering is used alone, without activating *Benders*.

The next two tables of results summarize the performances of two variants of the B&C: the one with no filtering, "B&C", and the one with the best identified configuration of filtering parameters, "B&C&Filter", which activates *LP-Fix*, *LR-Fix*, *Cap* ($\gamma = 0.9$) and *Conn*. 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 two formulations: the weak one, defined by (1)-(5), and the strong one, obtained from the weak one by adding 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 *lazy* constraints, which allows CPLEX to generate them dynamically, within its own cutting-plane algorithm. The results for the two formulations are shown on lines "CPLEX-Weak" and "CPLEX-Strong" in Tables 8 and 9. For both formulations, 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 the two CPLEX variants do not solve the same instances as the two B&C variants, we also count: 1) the number of instances that each CPLEX variant 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 each CPLEX variant 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 significant impact on

Class	Method	CPU	Nodes	Gap	Instances
I (19)	*B&C&Filter*	36000	13482	0.73	0
	B&C	36000	9124	0.82	0
	CPLEX-Weak	36000	742932	1.89	0
	CPLEX-Strong	36000	611284	1.87	0
II (3)	*B&C&Filter*	36000	451612	4.84	0
	B&C	36000	436214	4.89	0
	CPLEX-Weak	24015	2087072	3.69	+1
	CPLEX-Strong	24017	2348956	3.72	+1
III-B (23)	*B&C&Filter*	36000	64376	0.89	0
	B&C	36000	45951	1.10	0
	CPLEX-Weak	36000	1944022	2.89	0
	CPLEX-Strong	36000	1455278	2.85	0

Table 9: Comparison with CPLEX on instances unsolved by the B&C&Filter method

the performance of the B&C algorithm for all classes of instances. On the solved instances, significant reductions in the CPU time are observed: 55%, 50%, 25% and 31% reductions on average are obtained on Classes I, II, III-A and III-B, respectively. On unsolved instances, the average gaps are reduced by 0.09%, 0.05% and 0.21% on Classes I, II and III-B, respectively. The reductions in the gap are correlated with significant increases in the number of nodes that can be explored within the time limit of 10 hours of CPU time: 48%, 4% and 40% increases in the number of nodes are obtained on Classes I, II and III-B, respectively. These results also show that the two B&C variants outperform the two CPLEX variants on Classes I and III-B, while the opposite is true for Class II. These observations are consistent with the results presented in [14], 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 [14]. Of the two CPLEX variants, CPLEX-Strong is generally slightly better than CPLEX-Weak. Overall, B&C&Filter solves 20 instances more than CPLEX-Strong on instances in Classes I and III-B, while CPLEX-Strong 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-Strong.

6 Conclusion

We have presented a state-of-the-art B&C algorithm for the MCND that combines the cutting-plane method from [14], an adaptation of the reliability branching rule introduced in [3], and a series of filtering methods taking particular 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 significantly 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 the Introduction). In particular, the feasibility-based filtering techniques (combinatorial Benders cuts, connectivity-based filtering 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 (LP-based reduced cost fixing, Lagrangian-based reduced cost fixing and reduced cost bound reduction) depend on the cutting-plane procedure, but could be adapted in the context of column generation and Lagrangian relaxation methods. Ultimately, 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, large-scale, network design problems.

Acknowledgments

The authors express their gratitude to Geneviève Hernu and Serge Bisaillon, analysts at CIRRELT. Their efforts, high professionalism and dedication in implementing and testing the innumerable variants of the methods and algorithms presented in this paper 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 with the Department of Computer Science and Operations Research, Université de Montréal, and the Department of Economics and Business Administration, Molde University College, Norway. Funding for this project has been provided by the Natural Sciences and Engineering Council of Canada (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 Fonds de recherche du Québec through their infrastructure grants and of Calcul Québec and Compute Canada through access to their high-performance computing infrastructure.

References

[1] K. Aardal. Capacitated facility location: separation algorithms and computational experience. *Mathematical Programming*, 81:149–175, 1998.

[2] K. Aardal, Y. Pochet, and L.A. Wolsey. Capacitated facility location: valid inequalities and facets. *Mathematics of Operations Research*, 20:562–582, 1995.

[3] T. Achterberg, T. Koch, and A. Martin. Branching rules revisited. *Operations Research Letters*, 33:42–54, 2005.

[4] D. Applegate, R.E. Bixby, V. Chvatal, and W. Cook. Finding cuts in the TSP. Technical Report 95-05, DIMACS Technical Report, 1995.

[5] A. Atamtürk. Flow pack facets of the single node fixed-charge flow polytope. *Operations Research Letters*, 29:107–114, 2001.

[6] A. Atamtürk. On capacitated network design cut-set polyhedra. *Mathematical Programming*, 92:425–437, 2002.

[7] A. Atamtürk and D. Rajan. On splittable and unsplittable capacitated network design arc-set polyhedra. *Mathematical Programming*, 92:315–333, 2002.

[8] A. Atamtürk and M.W.P. Savelsbergh. Integer-programming software systems. *Annals of Operations Research*, 140:67–124, 2005.

[9] E. Balas. Facets of the knapsack polytope. *Mathematical Programming*, 8:146–164, 1975.

[10] F. Barahona. Network design using cut inequalities. *SIAM Journal of Optimization*, 6:823–837, 1996.

[11] M. Benichou, J.M. Gauthier, P. Girodet, G. Hentges, G. Ribiere, and O. Vincent. Experiments in mxed-integer programming. *Mathematical Programming*, 1:76–94, 1971.

[12] D. Bienstock, S. Chopra, O. Günlük, and C.Y. Tsai. Minimum cost capacity installation for multicommodity network flows. *Mathematical Programming*, 81:177–199, 1998.

[13] D. Bienstock and O. Günlük. Capacitated network design-polyhedral structure and computation. *INFORMS Journal on Computing*, 8:243–259, 1996.

[14] M. Chouman, T.G. Crainic, and B. Gendron. Commodity representations and cut-set inequalities for multicommodity capacitated fixed charge network design problem. Technical Report CIRRELT-2011-56, CIRRELT (Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation), 2011.

[15] G. Codato and M. Fischetti. Combinatorial Benders Cuts for Mixed-Integer Linear Programming. *Operations Research*, 54:756–766, 2006.

[16] A.M. Costa, J.F. Cordeau, and B. Gendron. Benders, metric and cutset inequalities for multicommodity capacitated network design. *Computational Optimization and Applications*, 42:371–392, 2009.

[17] A.M. Costa, J.F. Cordeau, B. Gendron, and G. Laporte. Accelerating Benders decomposition with heuristic master problem solutions. *Pesquisa Operacional*, 32:3–20, 2012.

[18] T. G. Crainic, M. Gendreau, and J.M. Farvolden. A simplex-based tabu search method for capacitated network design. *INFORMS Journal on Computing*, 12:223–236, 2000.

[19] T.G. Crainic, A. Frangioni, and B. Gendron. Multicommodity capacitated network design. In P. Soriano and B. Sanso, editors, *Telecommunications Network Planning*, pages 1–19. Kluwer Academics Publisher, 1999.

[20] T.G. Crainic, A. Frangioni, and B. Gendron. Bundle-based relaxation methods for multicommodity capacitated fixed charge network design. *Discrete Applied Mathematics*, 112:73–99, 2001.

[21] T.G. Crainic, A. Frangioni, B. Gendron, and F. Guertin. Oobb: An object-oriented library for parallel branch-and-bound. Presented at the CORS/INFORMS International Conference, Toronto, Canada, June 14-17 2009, 2009.

[22] T.G. Crainic and M. Gendreau. Cooperative parallel tabu search for capacitated network design. *Journal of Heuristics*, 8:601–627, 2002.

[23] T.G. Crainic, B. Gendron, and G. Hernu. A slope scaling/Lagrangean perturbation heuristic with long-term memory for multicommodity capacitated fixed-charge network design. *Journal of Heuristics*, 10:525–545, 2004.

[24] A. Frangioni. `http://www.di.unipi.it/~frangio`, May 2014.

[25] V. Gabrel, A. Knippel, and M. Minoux. Exact solution of multicommodity network optimization problems with general step cost functions. *Operations Research Letters*, 25:15–23, 1999.

[26] B. Gendron and T.G. Crainic. Relaxations for multicommodity capacitated network design problems. Technical report, Publication CRT-945, Centre de recherche sur les transports, Université de Montréal, 1994.

[27] B. Gendron and M. Larose. Branch-and-price-and-cut for large-scale multicommodity capacitated fixed-charge network design. *EURO Journal on Computational Optimization*, 2014.

[28] I. Ghamlouche, T.G. Crainic, and M. Gendreau. Cycle-based neighbourhoods for fixed charge capacitated multicommodity network design. *Operations Research*, 51:655–667, 2003.

[29] I. Ghamlouche, T.G. Crainic, and M. Gendreau. Path relinking, cycle-based neighbourhoods and capacitated multicommodity network design. *Annals of Operations Research*, 131:109–133, 2004.

[30] Z. Gu, G.L. Nemhauser, and M.W.P. Savelsbergh. Lifted cover inequalities for 0-1 integer programs: computation. *INFORMS Journal on Computing*, 10:427–437, 1998.

[31] Z. Gu, G.L. Nemhauser, and M.W.P. Savelsbergh. Lifted cover inequalities for 0-1 integer programs: complexity. *INFORMS Journal on Computing*, 11:117–123, 1999.

[32] Z. Gu, G.L. Nemhauser, and M.W.P. Savelsbergh. Lifted flow cover inequalities for mixed 0-1 integer programs. *Mathematical Programming*, 85:439–467, 1999.

[33] O. Günlük. A branch-and-cut algorithm for capacitated network design problems. *Mathematical Programming*, 86:17–39, 1999.

[34] P.L. Hammer, E.L. Johnson, and U.N. Peled. Facets of regular 0-1 polytopes. *Mathematical Programming*, 8:179–206, 1975.

[35] M. Hewitt, G.L. Nemhauser, and M.W.P. Savelsbergh. Combining exact and heuristic approches for the capacitated fixed-charge network flow problem. *INFORMS Journal on Computing*, 22:314–325, 2010.

[36] K. Holmberg and D. Yuan. A Lagrangian heuristic based branch-and-bound approach for the capacitated network design problem. *Operations Research*, 48:461–481, 2000.

[37] J.N. Hooker. Logic, optimization, and constraint programming. *INFORMS Journal on Computing*, 14:295–321, 2002.

[38] N. Katayama, M. Chen, and M. Kubo. A capacity scaling heuristic for the multicommodity capacitated network design problem. *Journal of Computational and Applied Mathematics*, 232:90–101, 2009.

[39] G. Kliewer and L. Timajev. Relax-and-cut for capacitated network design. In *Proceedings of Algorithms-ESA 2005: 13th Annual European Symposium on Algorithms*, pages 47–58. Lecture Notes in Computer Science 3369, 2005.

[40] J.M.Y. Leung and T.L. Magnanti. Valid inequalities and facets of the capacitated plant location problems. *Mathematical Programming*, 44:271–291, 1989.

[41] Q. Louveaux and L.A. Wolsey. Lifting, superadditivity, mixed integer rounding and single node flow sets revisited. *Annals of Operations Research*, 153:47–77, 2007.

[42] T.L. Magnanti, P.B. Mirchandani, and R. Vachani. The convex hull of two core capacitated network design problems. *Mathematical Programming*, 60:233–250, 1993.

[43] T.L. Magnanti, P.B. Mirchandani, and R. Vachani. Modeling and solving the two-facility capacitated network loading problem. *Operations Research*, 43:142–157, 1995.

[44] S. Martello and P. Toth. Upper bounds and algorithms for hard 0-1 knapsack problems. *Operations Research*, 45:768–778, 1997.

[45] F. Ortega and L.A. Wolsey. A branch-and-cut algorithm for the single commodity uncapacitated fixed charge network flow problem. *Networks*, 41:143–158, 2003.

[46] M.W. Padberg, T.J. Van Roy, and L.A. Wolsey. Valid linear inequalities for fixed charge problems. *Operations Research*, 33:842–861, 1985.

[47] C. Raack, A.M.C.A. Koster, S. Orlowski, and R. Wessäly. On cut-based inequalities for capacitated network design polyhedra. *Networks*, 57:141–156, 2011.

[48] I. Rodríguez-Martín and J. J. Salazar-González. A local branching heuristic for the capacitated fixed-charge network design problem. *Computers & Operations Research*, 37:575–581, 2010.

[49] M.W.P. Savelsbergh. Preprocessing and probing techniques for mixed integer programming problems. *ORSA Journal on Computing*, 6:445–445, 1994.

[50] M. Sellmann, G. Kliewer, and A. Koberstein. Lagrangian cardinality cuts and variable fixing for capacitated network design. In *Proceedings of Algorithms-ESA 2002: 10th Annual European Symposium on Algorithms*, pages 845–858. Lecture Notes in Computer Science 2461, 2002.

[51] T.J. Van Roy and L.A. Wolsey. Solving mixed integer programming problems using automatic reformulation. *Operations Research*, 35:45–57, 1987.

[52] L.A. Wolsey. Faces of linear inequalities in 0-1 variables. *Mathematical Programming*, 8:165–178, 1975.