Benders Decomposition, Branch-and-Cut, and Hybrid Algorithms for the Minimum Connected Dominating Set Problem

Bernard Gendron
Département d'informatique et de recherche opérationnelle, Université de Montréal, Interuniversity Research Center on Enterprise Networks, Logistics and Transportation (CIRRELT), Montreal, Quebec, Canada H3T 1J4, bernard.gendron@cirrelt.ca

Abilio Lucena
Departamento de Administração e Programa de Engenharia de Sistemas e Computação, Universidade Federal do Rio de Janeiro, Rio de Janeiro, RJ, 21941-901, Brasil, abiliolucena@globo.com

Alexandre Salles da Cunha
Departamento de Ciência da Computação, Universidade Federal de Minas Gerais, Belo Horizonte, 31270-901, Brasil, acunha@dcc.ufmg.br

Luidi Simonetti
Instituto de Computação, Universidade Federal Fluminense, Niterói CEP 24210-240, Brasil, luidi@ic.uff.br

We present exact algorithms for solving the minimum connected dominating set problem in an undirected graph. The algorithms are based on two approaches: a Benders decomposition algorithm and a branch-and-cut method. We also develop a hybrid algorithm that combines these two approaches. Two variants of each of the three resulting algorithms are considered: a stand-alone version and an iterative probing variant. The latter variant is based on a simple property of the problem, which states that if no connected dominating set of a given cardinality exists, then there are no connected dominating sets of lower cardinality. We present computational results on a large set of instances from the literature.

Keywords: connected dominating sets; valid inequalities; Benders decomposition; branch and cut

History: Accepted by Michela Milano, Area Editor for Constraint Programming and Hybrid Optimization; received November 2012; revised June 2013; accepted October 2013. Published online in Articles in Advance.

1. Introduction

A dominating set in an undirected connected graph $G = (V, E)$ is a set $D \subseteq V$ such that $\Gamma(D) = V$, where $\Gamma(D) = D \cup \{j \in V \mid \{i, j\} \in E, i \in D\}$. The minimum dominating set problem consists in finding a dominating set of minimum cardinality. A connected dominating set is a dominating set $D$ such that the subgraph $G(D) = (D, E(D))$ is connected, where $E(D) = \{\{i, j\} \in E \mid i \in D, j \in D\}$. The minimum connected dominating set problem (MCDSP) consists in identifying a connected dominating set of minimum cardinality.

The MCDSP is closely related to the maximum leaf spanning tree problem (MLSTP), which consists in finding a spanning tree of $G$ with as many leaves as possible (see Lucena et al. 2010 for a review of the literature on the MLSTP). Indeed, given a connected dominating set $D$, a spanning tree of $G(D)$ can be easily identified. Such a tree can be enlarged into a spanning tree of $G$, where all vertices in $V \setminus D$ are leaves. Thus, for every connected dominating set $D$ of $G$, a spanning tree of $G$ with at least $|V| - |D|$ leaves can be efficiently found. In particular, if $D$ is a minimum connected dominating set, a spanning tree of $G$ with the maximum possible number of leaves results from the procedure outlined previously.

Domination in graphs is a concept behind a growing number of applications found in the literature. Early applications could be traced back to the location of radar stations (Berge 1973) and a particular network communication problem described in Liu (1968). Now, applications can be found in areas as diverse as the spread of technological innovations (Rogers 2003, Valente 1995), the marketing of new products (Domingos and Richardson 2001, Goldenberg et al. 2001), power systems (Asavathiratham et al. 2001, Fan and Watson 2012), Web graph problems (Cooper et al. 2005), the spread of communicable diseases (Eubank et al. 2004, Stanley 2006), and helping to alleviate social problems through social networks (Wang et al. 2011).

Applications that specifically involve MCDSP/MLSTP arise in the design of ad-hoc wireless sensor networks, where network topologies may change...
dynamically (Balasundaram and Butenko 2006). They can also be found in the design of defense strategies against the attack of worms in peer-to-peer networks (Liang and Sencun 2007). Another recent application addresses the design of fiber optics networks where regenerators of information may be required at some network vertices (Chen et al. 2010). A regenerator is an expensive equipment necessary to boost information quality that is degraded after traveling long distances in cable. Finally, minimum connected dominating sets are also suggested as models to investigate protein-protein interactions (Milenković et al. 2011).

For the MLSTP, polyhedral investigations are carried out in Fujie (2004), and exact algorithms are developed in Fujie (2003) and Lucena et al. (2010). A branch-and-bound algorithm is presented in Fujie (2003), where clever insights lead to the efficient computation of the (relatively weak) corresponding linear programming (LP) relaxation bounds. Two integer programming (IP) formulations are proposed in Lucena et al. (2010). The first one is based on a Steiner reformulation of the problem. The second one considers the problem in a directed graph, seeking for a spanning arborescence with as many leaves as possible. Although the lower bounds implied by the former are stronger, better computational results are obtained with a branch-and-cut method based on the latter. The latter algorithm is also shown to computationally dominate the algorithm in Fujie (2003). A heuristic method for the MLSTP is also suggested in Lucena et al. (2010). This heuristic method and the Steiner reformulation proposed in Lucena et al. (2010) are similar to those introduced in Chen et al. (2010) for the regenerator location problem (RLP), these contributions being independently developed. As indicated in Chen et al. (2010), the RLP can be reformulated as a MLSTP. Given that the investigation focus of that reference is the RLP, the MLSTP heuristics and the exact solution algorithm proposed there are only tested on MLSTP instances that originate from the RLP.

For the MCDSP, approximation algorithms are developed in Guha and Khuller (1998) and Marathe et al. (1995). Additionally, an IP formulation of the problem and a preliminary version of a branch-and-cut algorithm, to be further investigated here, are described in Simonetti et al. (2011). Compact formulations of the problem and also of the closely related connected power dominating set problem are investigated in Fan and Watson (2012). These formulations result after imposing solution connectivity through various means: Miller-Tucker-Zemlin (MTZ) inequalities, single commodity flows, multicommodity flows, and big-M based subtour breaking constraints introduced in Martin (1991). Corresponding exact solution algorithms are obtained by simply providing these formulations as an input to the IP solver CPLEX. Among them, as far as CPU times are concerned, the best performing algorithm is the one based on the MTZ formulation. The algorithms are tested on only six instances. Another exact solution algorithm, capable of solving the MCDSP in $O(1.8966^n)$ time, is introduced in Fernau et al. (2011). No computational results, indicating how well that algorithm works in practice, are available in that reference.

In this paper, we present two exact algorithms for the MCDSP, which rely on a general formulation of the problem based on the following binary variables: $y_i = 1$, if $i \in V$ belongs to a dominating set, 0 otherwise. The model can be stated as follows:

\[ z = \min \sum_{i \in V} y_i \quad (1) \]
\[ \sum_{j \in \Gamma(i)} y_j \geq 1, \quad i \in V, \quad (2) \]
\[ \text{connected}(y) \quad (3) \]
\[ y_i \in [0, 1], \quad i \in V. \quad (4) \]

The objective function $(1)$ minimizes the number of vertices in any connected dominating set. The cover inequalities $(2)$ define a dominating set. The generic constraint $(3)$ imposes the connectivity of the subgraph induced by $y$. The two exact algorithms, a Benders decomposition approach and a branch-and-cut method, differ in the way they handle the generic constraint $(3)$. The Benders decomposition algorithm iterates between (1) the solution of a master problem defined by $(1), (2), (3)$, and $(4)$, plus a number of additional inequalities, thus providing a dominating set $D$ induced by $y$; and (2) the solution of a subproblem, represented by the generic constraint $(3)$, that verifies whether $D$ is a connected subgraph and adds cuts to the master problem accordingly. To represent constraint $(3)$, the branch-and-cut algorithm uses a classical spanning tree formulation that introduces additional edge-based variables representing whether an edge belongs to a spanning tree for the subgraph induced by $y$. In addition to these two methods, we also investigate a hybrid algorithm that applies the Benders decomposition strategy, but builds an initial master problem at each iteration by adding valid inequalities derived from the spanning tree formulation, as in the branch-and-cut method.

We further exploit a basic property of connected dominating sets to devise an iterative probing strategy that can be used in combination with any of the three algorithms. This property simply states that if no connected dominating set of a given cardinality $d > 0$ exists, then there are no connected dominating sets of cardinality $d - 1$. We use this property within a simple iterative approach that starts with an initial connected dominating set of cardinality $d$ and attempts to find a connected dominating set of cardinality $d - 1$; if no such set is found, the iterations stop with the current best solution of cardinality $d$. At every iteration, we use any of our three algorithms to find a connected dominating
set of cardinality $d - 1$, thus giving rise to the iterative probing variants of each of the three algorithms.

In summary, recalling the earlier discussion, we present and compare in this paper six exact algorithms for the MCDSP. Namely, stand-alone and iterative probing variants of Benders decomposition, two branch-and-cut schemes, and two hybrid methods. These algorithms are also going to be compared with the ones investigated in Lucena et al. (2010) and the MTZ-based algorithm found in Fan and Watson (2012). Given the very close similarities that exist between one of the exact algorithms in Lucena et al. (2010) and the one investigated in Chen et al. (2010), the algorithm in the latter reference is not going to be explicitly involved in our comparisons.

The remainder of the paper is organized as follows: In §2 we give the details of the iterative probing strategy. In §3, we present the Benders decomposition method, and the branch-and-cut algorithm is the topic of §4. Section 5 describes the hybrid algorithm and §6 presents the heuristic method used to initialize all algorithms. Computational results are reported in §7. We conclude the paper in §8.

2. Iterative Probing Strategy

Before stating the basic property that allowed us to devise the iterative probing strategy for the MCDS, let us introduce the notation used throughout the paper: let $G = (V, E)$ be a connected undirected graph and $\mathcal{P}(V) = \{S \subset V \mid S \neq \emptyset\}$ be the collection of all proper subsets of $V$; for any $S \in \mathcal{P}(V)$, we denote by $\overline{S}$ its complement $V \setminus S$ and by $\Gamma(S) = S \cup \{j \in V \mid \{i, j\} \in E, i \in S\}$ its closed neighborhood (when $S = \{i\}$, we write $\Gamma(\{i\}) = \Gamma(i)$); $\mathcal{D} = \{S \in \mathcal{P}(V) \mid \Gamma(S) = V\}$ and $\mathcal{D} = \mathcal{P}(V) \setminus \mathcal{D}$ are the collections of all dominating sets and all nondominating sets of $G$, respectively; for any $S \in \mathcal{P}(V)$, we denote by $G(S) = (S, E(S))$ the subgraph induced by $S$, where $E(S) = \{\{i, j\} \in E \mid i \in S, j \in S\}$; $\mathcal{E} \subseteq \mathcal{P}(V)$ and $\mathcal{E} = \mathcal{P}(V) \setminus \mathcal{D}$ are the collections of all proper subsets of $V$ that induce a connected subgraph and a disconnected subgraph of $G$, respectively.

The iterative probing strategy is based on the following property:

**Proposition 1.** If there exists a connected dominating set of cardinality $d < |V|$, then there exists a connected dominating set of cardinality $d + 1$.

This property is trivial to show. Assume there exists a connected dominating set $D$ of cardinality $d < |V|$; by adding any vertex in $V \setminus D$ to $D$, we then obtain a connected dominating set of cardinality $d + 1$. As a direct consequence of this property, we have the following:

**Corollary 2.** If there are no connected dominating sets of cardinality $d + 1 > 1$, then there are no connected dominating sets of cardinality $d$.

Given a connected dominating set of cardinality $d + 1 > 1$, the iterative probing strategy simply looks for a connected dominating set of cardinality $d$. If there is no such set, the algorithm stops. Otherwise, a connected dominating set of cardinality $d$ is obtained; at the next iteration, the algorithm looks for a connected dominating set of cardinality $d - 1$. More formally, the iterative probing strategy can be stated as follows:

1. Find a connected dominating set $D$; let $d + 1 = |D|$. If $d = 0$, then stop: $D$ is the optimal solution of value $d + 1$.
2. **Probing:** Try to find a connected dominating set of cardinality $d$.
3. If no connected dominating set has been found, then stop: $D$ is the optimal solution of value $d + 1$.
4. Let $D$ be the connected dominating set just found; let $d + 1 = |D|$ and return to step 2.

Under the assumption that the decision problem in step 2 is solved exactly, this strategy provides an optimal solution to the MCDS, by virtue of Corollary 2. At the first step of this strategy, the case $d = 0$ is easy to verify, prior to the solution of any problem instance, by checking the condition $|\Gamma(i)| = |V|$ for some $i \in V$; whenever this is the case, the minimum connected dominating set contains only one element and the problem is trivially solved.

To solve the decision problem in step 2, we add to the general formulation (1)–(4) the following $d$-cut equation:

$$\sum_{i \in V} y_i = d.$$  \hspace{1cm} (5)

The corresponding decision problem, called the $d$-CDSP, can be solved by any of the three methods described in the next sections: the Benders decomposition algorithm (see §3), the branch-and-cut algorithm (see §4), or the hybrid algorithm (see §5).

3. Benders Decomposition Algorithm

The general strategy in Benders decomposition is to alternate between solving a master problem and a so-called Benders subproblem. In the master problem, some constraints of the original problem are relaxed, which induces a lower bound on the optimal objective function value. In our case, the generic constraint connected($y$) is relaxed and the master problem is defined by (1), (2), (4), and additional constraints, called Benders cuts. Given a solution to the master problem, the Benders subproblem attempts to identify a solution that satisfies all the constraints. Here, the solution $y$ to the master problem induces a dominating set $D$ and a subgraph $G(D) = (D, E(D))$ for which we verify the connectivity, thus attempting to enforce the constraint connected($y$). Determining the connectivity of $G(D)$ can be performed in $O(|E(D)|)$ by a graph traversal algorithm. If $G(D)$ is connected, we obtain
an upper bound on the optimal objective function value and the algorithm stops since the lower and upper bounds are equal. Otherwise, the solution $y$ is not feasible and so-called Benders feasibility cuts are generated and added to the master problem to be solved at the next iteration.

As mentioned earlier, the existence of a dominating set of cardinality 1 can be easily verified by checking the condition $|\Gamma(i)| = |V|$ for some $i \in V$. Provided such a set does not exist, the cover inequalities (2) can be strengthened as follows:

$$\sum_{j \in \Gamma(i)} y_j \geq 1, \quad i \in V. \quad (6)$$

This simple modification allows avoiding trivial master problems at the initial stages of the algorithm.

3.1. Benders Cuts

In Benders decomposition, the classical way of cutting a feasible solution of value $z^u$ is simply to impose the constraint that the objective function value should be strictly less than $z^u$. In our case, since the objective function value is integer, this constraint can be written as

$$\sum_{i \in V} y_i \leq z^u - 1. \quad (7)$$

This optimality cut is added to the master problem when solving the MCDSP. The value $z^u$ is determined by the heuristic method described in §6.

Let $D$ be the dominating set found when solving the Benders master problem; when the subgraph induced by $D$ is not connected, at least one vertex in $D = V \setminus D$ must be included in any minimum connected dominating set, yielding the feasibility cut

$$\sum_{i \in D} y_i \geq 1. \quad (8)$$

Since there is a finite number of such cuts and, eventually, all disconnected dominating sets would be discarded by adding these cuts, the algorithm converges to an optimal solution to the MCDSP. This type of feasibility cut arises naturally in the so-called logic-based and combinatorial Benders decomposition frameworks (Codato and Fischetti 2006, Hooker and Ottosson 2003).

Because the resulting algorithm is convergent, it suggests a mathematical programming formulation for the MCDSP, obtained by adding to (1), (4), and (6) the following constraints, called cut inequalities:

$$\sum_{i \in S} y_i \geq 1, \quad S \in \mathcal{C}, \quad (9)$$

where $\mathcal{C}$, as previously indicated, is the collection of all proper subsets of $V$ that induce disconnected subgraphs. This formulation is similar to one of the models proposed by Fujie (2004) for the MLSTP.

The cut inequalities are generally weak since many disconnected sets require much more than one extra node to become connected. To characterize stronger versions of the cut inequalities, we introduce the notion of a minimally disconnected set, which is a set of vertices $T \subset V$ that induces a disconnected subgraph, but such that there exists one vertex $j \in T$ for which $G(T \cup \{j\})$ is connected. Any cut inequality associated to a disconnected, but not minimally disconnected, set $S$ is dominated by at least one cut inequality associated to a minimally disconnected set $T$.

From any set $S \in \mathcal{C}$, to derive a minimally disconnected set $T$ that generates a tighter cut inequality, one has to solve a Steiner problem in a graph obtained by shrinking all connected components of $S$. The shrunk nodes (connected components) are the terminals, and all the other nodes (those in $V \setminus S$) are the nonterminals. Additionally, the cost of any edge in the graph is set to one. Therefore, solving the Steiner problem in the resulting graph provides the minimum number of Steiner vertices, say $m_s$, that must be added to $S$ to obtain a connected subgraph of $G$. By removing any of the added vertices, we define a minimally disconnected set $T$. Since $\sum_{i \in S} y_i \geq \sum_{i \in T} y_i \geq 1$, the cut inequality associated to the minimally disconnected set $T$ dominates that associated to the disconnected set $S$, i.e., $\sum_{i \in S} y_i \geq 1$ is a lifting of the cut inequality corresponding to $S$. This implies a second formulation for the MCDSP, where the cut inequalities are restricted to minimally disconnected sets. Let $\mathcal{C}_1 = \mathcal{C} \cap \{S \in \mathcal{P}(V) \mid m_s = 1\}$, the collection of minimally disconnected sets; the model is then defined by adding to (1), (4), and (6), the following lifted cut inequalities:

$$\sum_{i \in S} y_i \geq 1, \quad S \in \mathcal{C}_1. \quad (10)$$

In the framework of combinatorial Benders decomposition (Codato and Fischetti 2006), our notion of a minimally disconnected set corresponds to the concept of minimal infeasible subsystem. For the MCDSP, we can further improve the cut, by means of the following result:

**Proposition 3.** Let $S \in \mathcal{C}$ and $m_s$ be the minimum number of vertices that must be added to $S$ to obtain a connected subgraph of $G$. The following inequality is valid for the MCDSP:

$$\sum_{i \in S} y_i \geq m_s. \quad (11)$$

**Proof.** There are two possibilities for $S$:

1. $S$ is a dominating set, in which case $m_s$ is the smallest number of vertices required to enlarge $S$ to obtain a connected, and therefore a dominating, set; thus, the result follows.
(2) \( S \) is not a dominating set, in which case \( m_S \) is the smallest number of vertices required to enlarge \( S \) to obtain a connected set; \( m_S \) is therefore a lower bound on the number of vertices required to enlarge \( S \) to obtain a connected and dominating set for \( G \). The result thus follows. \( \square \)

This proposition implies a third formulation for the MCDSP, obtained by adding to (1), (4), and (6), the following strengthened cut inequalities:

\[
\sum_{i \in \bar{S}} y_i \geq m_{\bar{S}}, \quad S \in \mathcal{E}.
\] (12)

Given a disconnected dominating set \( D \) obtained when solving the Benders master problem and \( m_D \), one can generate a corresponding strengthened cut inequality. Clearly, this inequality dominates the associated lifted cut inequality. Indeed, let \( U \subseteq \bar{D} \) be the set of vertices that is added to \( D \) to obtain a minimally disconnected set \( D \cup U \) to generate the lifted cut inequality. By the definition of \( U \) and \( m_D \), we have \( m_D = 1 + |U| \). Thus, \( \sum_{i \in \bar{D}} y_i \geq m_D \geq 1 + \sum_{i \in \bar{U}} y_i \), which implies \( \sum_{i \in \bar{D}} y_i \geq 1 \), the lifted cut inequality.

Thus, for the MCDSP, we are able to generate stronger feasibility cuts than the ones derived in the combinatorial Benders framework. To be practical, we do not compute the exact value for the right-hand side \( m_{\bar{S}} \), since that would imply solving a Steiner problem for each \( S \in \mathcal{E} \). Instead, we compute a lower bound \( m_{\bar{S}} \) on \( m_{\bar{S}} \) as follows. The shrunk graph described earlier is directed by replacing each edge with two arcs, one in each direction. If an arc ends at a nonterminal node, its cost is 1, otherwise, its cost is 0. Then, we compute the shortest paths between all pairs of terminals. The minimum distance between any two terminals gives the minimum number of nonterminals that are needed to connect them. As such, it provides a valid lower bound on \( m_{\bar{S}} \). The lower bound \( m_{\bar{S}} \) we use is thus the maximum of the minimum distance between any two terminals (computed by performing Dijkstra’s algorithm once for each terminal used as the source node).

The feasibility cut we actually use in our Benders decomposition algorithm is thus

\[
\sum_{i \in \bar{S}} y_i \geq m_{\bar{S}}, \quad S \in \mathcal{E},
\] (13)

where \( m_{\bar{S}} \) is computed as described earlier. Since cuts (13) can be derived independently of the particular Benders decomposition algorithm that we developed, they can be used in other methods; we will use them in our branch-and-cut algorithm presented in §4.

### 3.2. Outline of the Algorithm

In this section, we give an outline of the stand-alone variant of the Benders decomposition algorithm; the iterative probing variant of the same algorithm is described in §3.3. The algorithm solves the model defined by (1), (4), (6), and (13). The optimality cut (7) is also added.

To initialize the algorithm, we apply the heuristic method described in §6, followed by the application of the strengthening procedure described in §3.3. We thus obtain a connected dominating set \( D \) of value \( z^u = |D| \), which is used in the optimality cut (7).

The Benders decomposition algorithm can be outlined as follows:

1. Find a connected dominating set \( D \) of value \( z^u \); if \( z^u = 1 \), then stop: \( D \) is the optimal solution.

2. Solve the Benders master problem.

3. If no feasible solution has been found, then stop: \( D \) is the optimal solution of value \( z^u \).

4. Otherwise, let \( D \) be the dominating set just found; if \( D \) is a connected dominating set, then stop: \( D \) is the optimal solution.

5. If \( D \) is not connected, generate a feasibility cut (13) and return to step 2.

### 3.3. Iterative Probing Variant

At every step of the iterative probing strategy, we can perform the Benders decomposition algorithm previously outlined by simply adding to the master problem formulation the \( d \)-cut equation (5). We can, however, further improve the performance of the iterative probing variant of Benders decomposition by generating, instead of the cut on the objective function (7), which is implied by the \( d \)-cut equation (5), another form of optimality cut that mirrors the feasibility cut. If the subgraph induced by \( D \) is connected, then, to find an improving connected dominating set, we must exclude \( D \) from further consideration, which can be done by requiring at least one vertex in \( D \) to be excluded from an optimal solution, yielding the following cut:

\[
\sum_{i \in D} y_i \leq |D| - 1.
\] (14)

This constraint is implied by the \( d \)-cut equation (5), since \( d = |D| - 1 \). Hence, this weaker version of the optimality cut is never added to the master problem. However, it is possible to efficiently derive from it a strengthened cut, as follows. If, for each vertex \( i \in D \), removing \( i \) from \( D \) yields a nondominating or a disconnected subgraph, the right-hand side can also be replaced by \( |D| - 2 \), because there cannot be an optimal solution obtained by removing any single vertex in \( D \).

The strengthening procedure thus scans each vertex \( i \in D \) to verify if \( D \setminus \{i\} \) is a connected dominating set and thus to generate an improving solution; therefore, it can be performed in time \( O(|D| |E|) \). After scanning the vertices in \( D \), if we determine that removing \( i \) from \( D \) yields a nondominating or a disconnected subgraph
for each vertex \( i \in D \), we then generate the strengthened optimality cut:

\[
\sum_{i \in D} y_i \leq |D| - 2. \tag{15}
\]

If, for some vertex \( i \in D \), \( D \setminus \{i\} \) is a connected dominating set, we have identified a feasible solution of value \( d = |D| - 1 \). We immediately stop checking the condition for the other vertices in \( D \); instead, we restart the strengthening procedure with \( D \setminus \{i\} \) in place of \( D \). As a result, the strengthening procedure will always terminate with a strengthened optimality cut (associated to \( D \) or a subset of \( D \) defining a connected dominating set), potentially generating a series of successively improving feasible solutions along the way.

It is worth noting that it suffices to maintain at most one strengthened optimality cut over the whole course of the algorithm. Indeed, let us assume that the Benders master problem at the current iteration includes the \( d \)-cut equation (5), with right-hand side \( d = |D| - 1 \), as well as the strengthened optimality cut (15). Assuming the problem is feasible, we then obtain a dominating set \( D' \) of cardinality \( d \). If \( D' \) is disconnected, a feasibility cut is generated, but no optimality cut. If \( D' \) is connected, we update the \( d \)-cut equation \( \sum_{i \in V} y_i = d - 1 = |D'| - 1 = |D| - 2 \), which dominates the strengthened optimality cut \( \sum_{i \in D} y_i \leq |D| - 2 \). The latter can therefore be removed and possibly replaced by another one of the form \( \sum_{i \in D'} y_i \leq |D'| - 2 \), where \( D' \subseteq D \).

Note that the strengthened optimality cut (15) dominates the Benders feasibility cut associated to each disconnected set \( D \setminus \{i\} \). Each such set is minimally disconnected, since \( D \) is connected; the Benders feasibility cut has the form \( \sum_{i \in E \setminus \{j\}} y_j \geq 1 \). This inequality is implied by the \( d \)-cut equation \( \sum_{i \in V} y_i = |D| - 1 \) and the strengthened optimality cut, since \( \sum_{i \in E \setminus \{j\}} y_j = \sum_{i \in E \setminus \{j\}} y_i - \sum_{i \in D \setminus \{j\}} y_i = \sum_{i \in E \setminus \{j\}} y_i - \sum_{i \in D} y_i + y_j \geq (|D| - 1) - (|D| - 2) + y_j \geq 1 \).

The iterative probing variant of the Benders decomposition algorithm follows the stand-alone variant, with a few exceptions. First, the optimality cut (7) is replaced by the \( d \)-cut equation (5) for which we gradually decrease the right-hand side; a strengthened optimality cut of the form (15) is also added to the Benders master problem. The heuristic method described in §6 is performed to provide a connected dominating set \( D \) of value \( |D| = d + 1 \), which is used to initialize the \( d \)-cut equation (5). The initial Benders master problem thus contains the \( d \)-cut equation (5) and the strengthened optimality cut (15) associated to the best connected dominating set \( D \) found so far.

The iterative probing variant of the Benders decomposition algorithm can be outlined as follows:

1. Find a connected dominating set \( D \); if \( |D| = 1 \), then stop: \( D \) is the optimal solution.
2. Apply the strengthening procedure; add the resulting strengthened optimality cut to the Benders master problem; if a new connected dominating set \( D \) has been found, let \( d = |D| - 1 \) and update the right-hand side of the \( d \)-cut equation.
3. Solve the Benders master problem.
4. If no feasible solution has been found, then stop: \( D \) is the optimal solution.
5. Let \( D \) be the dominating set just found; if \( D \) is a connected dominating set:
   (a) Let \( d = |D| - 1 \) and update the right-hand side of the \( d \)-cut equation.
   (b) Apply the strengthening procedure; add the new strengthened optimality cut to the Benders master problem; if a new connected dominating set \( D \) has been found, let \( d = |D| - 1 \) and update the right-hand side of the \( d \)-cut equation.
6. If \( D \) is not connected, generate a feasibility cut (13) and return to step 3.

4. Branch-and-Cut Algorithm

The branch-and-cut algorithm is based on a particular representation of the generic constraint \( connected(y) \) as linear inequalities, namely, a representation that defines a spanning tree of the subgraph of \( G \) induced by \( y \). Denote by \( G(D) \) such a subgraph, \( D \subseteq V \) being its vertex set. Additionally, define \( x_{i,j} = 1 \), if edge \( e \in E \) belongs to a spanning tree of \( G(D) \), 0 otherwise.

Finally, consider a polyhedral region \( \mathcal{P}_0 \) defined by inequalities (2) and the following constraints, which characterize a spanning tree of \( G(D) \):

\[
\sum_{e \in S} x_e = \sum_{i \in V} y_i - 1, \quad S \subseteq \mathcal{P}(V), \quad j \in S, \quad (16)
\]

\[
\sum_{e \in S} x_e \leq \sum_{i \in S} y_i, \quad S \subseteq \mathcal{P}(V), \quad j \in S, \quad 0 \leq x_e, \quad e \in E, \quad (17)
\]

\[
0 \leq y_i, \quad i \in V. \quad (18)
\]

A formulation for the MCDSP (Simonetti et al. 2011) is thus given by

\[
z = \min \left\{ \sum_{i \in V} y_i \mid (x, y) \in \mathcal{P}_0 \cap ([0, 1]^{|V|}) \right\}. \quad (20)
\]

One classical approach for strengthening the LP relaxation of an IP model consists in identifying structures in the formulation and then appending valid inequalities for each structure into the original model. Our formulation embeds two basic structures.

Denoting by \( \mathcal{F} \) the polyhedral region defined by (16)–(19), the first one, i.e., the tree polytope, is defined as the convex hull of \( \{(x, y) \in \mathcal{F} \cap ([0, 1]^{|V|}) \} \). Likewise, denoting by \( \mathcal{Y} \) the polyhedral region defined by (2) and (19), the second structure, i.e., the set
A particular case of inequalities (23) arises when \( \Gamma(i) \cap \Gamma(j) = \emptyset \) for a given pair of vertices \( i, j \in V \). Under these conditions, it is immediate to verify that any cut set \( E(S, \bar{S}) \) separating \( \Gamma(i) \) and \( \Gamma(j) \) implies a valid inequality of the type (23).

Separating (23) for the particular case highlighted is not difficult. To illustrate it, assume that a LP relaxation of formulation (20) is available together with its corresponding support graph. After glueing together the vertices, respectively found in \( \Gamma(i) \) and \( \Gamma(j) \), super vertices \( i \) and \( j \) would result in the support graph. In doing so, when applicable, multiple edges between two given end nodes are replaced by a single edge. Additionally, the weight of that edge is set equal to the sum of the weights of the edges that imply it. It thus follows that determining the maximum flow between super vertices \( i \) and \( j \) identifies a minimum capacity cut between \( \Gamma(i) \) and \( \Gamma(j) \) and therefore solves the separation problem.

To evaluate the benefits of such separation schemes, we keep a list with all pairs of vertices \( i \) and \( j \) such that \( \Gamma(i) \cap \Gamma(j) = \emptyset \). During each LP relaxation at the root node of the enumeration tree, we identify, as described earlier, the minimum cut separating \( \Gamma(i) \) and \( \Gamma(j) \), for each pair \( i, j \) in the list. The impact of adding cuts identified this way into the LP relaxation at the root node is, however, quite small. Therefore, we actually do not use such a separation procedure in the final version of our branch-and-cut algorithm.

Finally, inequalities (23) can be generalized whenever \( V \) is partitioned into subsets \( (S_1, \ldots, S_k) \), for \( k \geq 2 \), where \( S_n \), for any \( l \in \{1, \ldots, k\} \), satisfies the following conditions: (1) \( S_l \not\subseteq \emptyset \) and (2) \( S_l \cap S \neq \emptyset \), for any \( S \subset \emptyset \). Under these conditions, the following inequality is valid for \( P_0 \):

\[
\sum_{e \in E(S_1, \ldots, S_k)} x_e \geq k - 1, \tag{24}
\]

where \( E(S_1, \ldots, S_k) \) is the set of edges with endpoints in different partition sets. These inequalities relate to the Steiner partition facets introduced in Chopra and Rao (1994a, b) and are not easy to separate exactly. However, as described next, we have devised a simple separation heuristic for them.

Preliminary computational results for the formulations described in this section appeared in Simonetti et al. (2011). In Simonetti, the LP relaxation bounds are obtained by optimizing the objective function over polytope \( P_1 = P_0^+ \cap \{(23)\} \), where \( P_0^+ \) corresponds to \( P_0 \) with the stronger cover inequalities (21) being used instead of (2). In §7, these results are compared with stronger bounds obtained by optimizing over polytope \( P_1^+ = P_1 \cap \{(13), (22) \text{ and } (24)\} \).

4.2. Outline of the Algorithm

The branch-and-cut algorithm is initialized by generating an upper bound with the heuristic method...
described in §6. The strengthening procedure of §3.3 is then applied in an attempt to improve the feasible solution thus obtained and to generate a strengthened optimality cut (15). In addition, the strengthening procedure is called every time the branch-and-cut algorithm generates a feasible solution. A new strengthened optimality cut (15) is then added to the model. Furthermore, this optimality cut is also stored in a cut pool to ensure that the cut is available for use at the nodes to be explored after backtracking.

Valid dual bounds are obtained by first solving the LP relaxation of
\[
\min \left\{ \sum_{i \in V} y_i \middle| (x, y) \in \mathcal{P} \cap (\mathbb{R}^{|\bar{E}|}, [0, 1]|V|) \right\},
\]
\[\mathcal{P}\] being the polyhedral region defined by (15), (16), (21), and
\[
x_i \leq y_i, \quad x_j \leq y_j, \quad e \in \{i, j\} \in E,
\]
where (26) naturally follow from the GSECs corresponding to \(S = \{i, j\}, i, j \in V, i \neq j\). Let \((\bar{x}, \bar{y})\) be an optimal solution to this model and \(G = (\bar{V}, \bar{E})\) be the support subgraph it implies for \(G\), i.e., \(\bar{V} = \{i \in V \mid \bar{y}_i > 0\}\) and \(\bar{E} = \{e \in E \mid \bar{x}_e > 0\}\). If \((\bar{x}, \bar{y})\) is integer and there is no GSEC (17) violated by it, the solution is optimal for the MCDSP. Otherwise, one should attempt to reinforce the relaxation by appending to it valid inequalities that are violated by \((\bar{x}, \bar{y})\).

The exact separation of GSECs can be efficiently carried out in \(O(|\bar{V}|)\) time complexity (Padberg and Wolsey 1983) via a maximum flow-minimum cut algorithm. However, for the solution algorithms investigated in this paper, GSECs are not separated exactly, because in terms of overall branch-and-cut running times, it proves more advantageous to only separate GSECs heuristically, through a procedure described later. In spite of that, for comparison purposes, corresponding values of \(\min\{\sum_{i \in V} y_i \mid (x, y) \in \mathcal{P}_i^*\}, \min\{\sum_{i \in V} y_i \mid (x, y) \in \mathcal{P}_j^*\}\) and \(\min\{\sum_{i \in V} y_i \mid (x, y) \in \mathcal{P}_k^*\}\) obtained under the exact separation of GSECs, are reported in §7.1.

Our heuristic separation of GSECs is carried out as follows. First, the edges in \(\bar{E}\) are sorted in nonincreasing order of their \(\bar{x}_e\) values. Then, a maximum cardinality forest of \(\bar{G}\) is computed through Kruskal’s algorithm (Kruskal 1956). Preference for entering the solution is given to edges with higher \(\bar{x}_e\) values. In accordance with Kruskal’s algorithm, each edge entering the solution merges two connected components into a larger one. In this process, for every new connected component being formed, their vertices are checked for GSEC violation. The procedure stops after a maximum cardinality forest is obtained.

Although driven for separating GSECs, the heuristic outlined above is also used to separate additional families of valid inequalities. This is carried out right after an edge inclusion operation is performed by the heuristic. Accordingly, let \(S\) be the vertex set for the connected component thus obtained, where \(\bar{S}\) is its complement in \(\bar{V}\). Sets \(S\) and \(\bar{S}\) are then checked for violation of strengthened GSECs (22), feasibility cuts (13), and cut sets (23). More precisely, if \(\bar{S} \notin \mathcal{G}\), at least one vertex in \(S\) must be part of any connected dominating set and therefore \(S\) implies a valid lifted GSEC inequality (22) that should be checked for violation. Moreover, if \(\bar{S} \in \mathcal{G}\), the value of \(m_{\bar{S}}\) should be computed and the inequality (13) implied by \(\bar{S}\) should be checked for violation. Finally, if \(S\) and \(\bar{S}\) belong to \(\mathcal{G}\), violation of (23) should be checked. Furthermore, if \(\bar{S} \in \mathcal{G}\) also applies, the cut inequality is lifted into a \(k\)-partition inequality (24). If no violated inequality is found by the heuristic, branching on variables is implemented.

### 4.3. Iterative Probing Variant

At each step of the iterative probing strategy, the branch-and-cut algorithm is called to solve the decision problem obtained by appending the \(d\)-cut equation (5) to the model. When a feasible solution of value \(d\) is obtained, the branch-and-cut algorithm is stopped. A new step of the iterative probing strategy is then performed. First, the strengthening procedure is applied in an attempt to improve the current feasible solution of value \(d\). Then, the model is initialized with the updated \(d\)-cut equation and all the cuts generated so far, except GSECs (17). Finally, the branch-and-cut algorithm for solving the new decision problem is performed.

Valid inequalities used to strengthen the formulation are the same as those used in the stand-alone version of the branch-and-cut algorithm. Accordingly, at every iterative probing solution round, inequalities (13), (17), and (22) to (24) are separated as described earlier.

### 5. Hybrid Algorithm

Our computational results, to be presented in §7, show that the branch-and-cut algorithm outperforms the Benders decomposition method on sparse instances, whereas the Benders decomposition algorithm is much faster on dense instances. This observation motivated the development of a hybrid algorithm that attempts to combine the best features of the two approaches. Indeed, the relaxations built by the branch-and-cut algorithm for sparse instances are extremely good, and the Benders decomposition algorithm improves the feasible solutions quickly for dense instances.

The hybrid algorithm therefore implements a Benders decomposition algorithm, but builds stronger restricted master problems by performing the separation of valid inequalities (13), (17), and (22) to (24), when solving the root node of every Benders master problem. More
At that point, all algorithms introduced in this paper, as well as those introduced in Lucena et al. (2010). That heuristic, in spite of being part of a MLSTP paper, is actually geared into solving the MCDSP. It works with two sets: \( \mathcal{D} \), to represent vertices in a connected dominating set and \( \mathcal{L} \), to represent those vertices that have at least one neighbor in \( \mathcal{D} \). The procedure is initialized by setting \( \mathcal{D} = \{v\} \) and \( \mathcal{L} = \Gamma(v) \setminus \{v\} \) for any \( v \in V \). Then, the basic operation performed at each iteration is to try to push vertices from \( \mathcal{L} \) into \( \mathcal{D} \), until a connected dominating set is found. Assuming that \( i \) is moved from \( \mathcal{L} \) to \( \mathcal{D} \), in the next iteration we have \( \mathcal{D} \leftarrow \mathcal{D} \setminus \{i\} \cup (\Gamma(i) \setminus (\mathcal{D} \cup \mathcal{L})) \) and \( \mathcal{L} \leftarrow \mathcal{D} \cup \{i\} \). Preference is given to include in \( \mathcal{L} \) vertices with as many neighbors as possible, not already included in \( \mathcal{D} \cup \mathcal{L} \). The heuristic stops when \( V = \mathcal{D} \cup \mathcal{L} \). At that point, \( \mathcal{D} \) defines a connected dominating set.

In our implementation, the heuristic is executed \( n \) times. In each one, set \( \mathcal{D} \) is initialized with a different vertex \( v \in V \). Therefore the procedure we actually implemented could be cast as a multistart version of the greedy heuristic in Lucena et al. (2010).

### 7. Computational Experiments

In this section, we empirically evaluate the six exact solution algorithms presented in this paper: the stand-alone (SABC) and iterative probing (IPBC) branch-and-cut algorithms, the stand-alone (SABE) and iterative probing (IPBE) Benders decomposition algorithms, and the stand-alone (SAHY) and iterative probing (IPHY) hybrid algorithms. All algorithms were implemented in C and computational experiments were carried out on a 2.0 GHz Intel XEON E5405 machine with 8 GB RAM. Search tree management for SABC and IPBC was enforced via the callback routines of the mixed-IP (MIP) solver XPRESS, release 19.00. The enumeration strategy implemented for these two algorithms was best-first search. For the Benders decomposition algorithms, the MIP module of XPRESS was used, under default settings, to solve the Benders master problems.

Two sets of MLSTP/MCDSP instances were used in our experiments. The first one was introduced in Lucena et al. (2010). They are associated with graphs \( G = (V, E) \) with densities ranging from 5% to 70% and number of vertices \( n \in \{30, 50, 70, 100, 120, 150, 200\} \). At most one test instance exists for every possible combination of \( n \) and \( d \) and therefore any given instance is clearly identified in our tables as \( n \_ d \). The second set comprises IEEE reliability test instances, used in the computational experiments in Fan and Watson (2012). They are associated with graphs with densities ranging from less than 1% to nearly 15% and \( n \in \{14, 30, 57, 73, 118, 300\} \).

For each instance in these two sets, every solution algorithm was allowed to run for at most 3,600 CPU seconds. Whenever that limit was reached and optimality had not been proven, the instance was left unsolved by the corresponding algorithm.

All tables of detailed computational results we refer to in the next sections are not presented in the main text body. Instead, all the tables are presented in an accompanying online supplement (available as supplemental material at http://dx.doi.org/10.1287/ijoc.2013.0589).

#### 7.1. Linear Programming Lower Bounds

Table 1 presents a number of different LP relaxation bounds for the MCDSP and MLSTP. Entries in the first column identify the test instances. The last six instances in the table, with names starting with the prefixes IEEE and RTS, are taken from the computational study in Fan and Watson (2012). All the others were introduced in Lucena et al. (2010). The instance name is followed, in the next three columns, by MCDSP LP relaxation bounds, respectively, implied by polytopes \( \mathcal{P}_0^+ \), \( \mathcal{P}_1^- \), and \( \mathcal{P}_1^+ \). For the computation of these three bounds, we first call the separation heuristic for GSECs outlined earlier. Therefore, not only GSECs, but also inequalities (13) and (22)–(24) are separated. If no violated cut is found by the heuristic, the exact separation of GSECs, through minimum cut algorithms, is carried out next. During the application of the GSEC exact separation procedure in Padberg and Wolsey (1983), each vertex set (and its complement) that results from a minimum cut computation is checked (when applicable) for the violation of (13) and (22)–(24). Therefore, the LP relaxation bounds quoted for \( \mathcal{P}_0^+ \) and \( \mathcal{P}_1^+ \) represent a lower bound on the values that would otherwise be obtained if exact separation were used. Entries in the following two columns of Table 1 give LP relaxation bounds for the two different MLSTP reformulations
investigated in Lucena et al. (2010), namely, the directed graph (DGR) and the Steiner tree reformulation (STR). MLSTP bounds, in this case, are expressed in terms of their corresponding MCDSP bounds. The column “MTZ” indicates the LP bounds implied by the MTZ MCDSP reformulation suggested in Fan and Watson (2012). Finally, optimal MCDSP integer solution values are presented in the last column. For any given instance, whenever a particular LP relaxation bound could not be computed within the 3,600 CPU-second time limit imposed, character “-” appears in the corresponding entry.

These results show that the bounds for $P_1^+$ are always better than or equal to corresponding DGR and MTZ bounds and are, most of the time, weaker than their STR counterparts. However, STR bounds are typically very expensive to compute, with the time limit exceeded for 13 of the 47 tested instances. Bounds for $P_1$ significantly improve on those obtained for $P_0^+$. However, only marginal gains are obtained when going from $P_1$ to $P_2^+$. This fact, as we will see next, partially explains why, with very few exceptions, neither SABC nor IPBC significantly improve on the results obtained in Simonetti et al. (2011).

For all instances in Fan and Watson (2012), the lower bounds implied by $P_1^+$ match corresponding optimal solution values, i.e., the implied duality gap is zero. Additionally, for all but two of these instances (IEEE-300-Bus and RTS96), corresponding solutions are integer feasible.

### 7.2. Comparison of Algorithms

Table 2 compares the CPU times obtained for each of the six algorithms proposed in this study with those attained by the branch-and-cut methods in Simonetti et al. (2011) and Lucena et al. (2010). Additionally, MIP solver XPRESS also takes part in the experiments, running under default settings for the XPRESS Mosel programming language implementation of the MTZ reformulation in Fan and Watson (2012). Computational experiments for that algorithm and also for the six methods introduced here were conducted under the same no multithreading computational environment. Whenever an algorithm hits the imposed time limit of 3,600 seconds without producing an optimality certificate, character “-” is used in the corresponding column entry.

Detailed computational results for the branch-and-cut, Benders decomposition, and hybrid algorithms are presented, respectively, in Tables 3—5.

The first column entries in Table 3 identify the tested instances. Initial upper bounds (IUB) appear in the second column. These bounds, computed with the heuristic method described in §6, are used to initialize all methods introduced here. For the next two columns, respectively, under the headings UB and $t(s)$, the following computational results are presented for SABC: the best upper bound available on termination and the CPU time spent, in seconds. Whenever the time limit is reached and the instance is left unsolved, “-” appears in the corresponding $t(s)$ entry. The next three columns apply to IPBC. Column “Iter” shows the number of probing iterations carried out, i.e., the number of times a new best incumbent feasible solution was found and, consequently, the right-hand side of the $d$-cut equation (5) had to be reduced. Finally, the last two columns in Table 3, respectively, apply to the branch-and-cut algorithm in Simonetti et al. (2011) and the DGR algorithm in Lucena et al. (2010). They indicate the CPU time, in seconds, taken by these algorithms to find proven optimal solutions, respectively, for the MCDSP and the MLSTP. Given that the same machine was used here and in Simonetti et al. (2011), the previously defined CPU time limit directly applies to the algorithm in Simonetti et al. (2011). The last two columns in Table 3 relate to the branch-and-cut algorithm based on the MTZ reformulation. They, respectively, give the best upper bound (UB) found throughout the implicit enumeration and the CPU time taken by the algorithm. Since the algorithm relies on the XPRESS solver and is tested under the same machine used for the algorithms introduced here, CPU times are directly comparable among them. If, for a given test instance, an algorithm does not find a feasible integer solution within the imposed time limit, “$\infty$” appears in the corresponding UB entry.

Computational results for Benders decomposition algorithms SABE and IPBE appear in Table 4. For Table 4, with the single exception of the “Iter” heading, definitions previously introduced for Tables 1 and 3 apply. Entries under “Iter” indicate the number of Benders master problems (BMP) that had to be solved before the incumbent solution was proven to be optimal or else when the time limit was reached. In the subsequent “Cuts” column we indicate the number of feasibility cuts introduced after the execution of the master problems. Table 5 presents the computational results for the hybrid algorithms SAHY and IPHY; the format is similar to that of Table 4, as well as the meanings of each column entries. Solving the master problems is responsible for nearly 100% of the total CPU time. Hence, we do not present the time needed to solve the subproblems and run the heuristic. Only the total CPU time is indicated in the tables.

These results show that SABC and IPBC manage to solve, respectively, 38 and 36 of the 47 tested instances (120_d20 and 150_d30 are solved by SABC, but not by IPBC). When we focus on those instances that both algorithms manage to solve, it appears that IPBC outperforms SABC when densities $d \geq 50\%$ apply. Conversely, SABC outperforms IPBC when $d \leq 30\%$ holds. However, this somewhat general trend does not
always hold; for example, for instance 120_d5, where CPU times for SABC and IPBC are, respectively, 1,465.05 and 199.01 seconds. For that instance, since initial upper bounds are already very close to optimal solution values, just a few probing iterations are required for IPBC to find a proven optimal solution.

We now focus on comparing SABC and a preliminary version of the same algorithm investigated in Simonetti et al. (2011). Although the algorithm in Simonetti et al. (2011) already attempts to heuristically separate lifted GSECs (22), SABC goes further in that direction and additionally attempts to separate feasibility cuts (13) and cut set inequalities (23)–(24). In spite of that, the two algorithms appear to perform similarly, as the set of instances solved by both do not vary a lot. Additional inequalities separated by SABC do not seem to pay off computationally.

Before comparing SABC to DGR, we remark that the DGR algorithm in Lucena et al. (2010) was tested on a 3.00 GHz Intel XEON X5472-based machine with 16 GB of RAM. Nevertheless, for comparison purposes, the differences in the machines used here and in Lucena et al. (2010) do not favor our computational results. Therefore, for any test instance that could not be solved in less than 3,600 time seconds by DGR, with the machine considered in Lucena et al. (2010), a "-" also appears in its corresponding CPU time entry in the tables. From the computational results shown in Table 2, the two algorithms provide optimality certificates for essentially the same set of instances. For only four cases, SABC did not manage to solve an instance solved by DGR within the time limit. However, DGR tends to be faster than SABC, sometimes much faster, for low density instances, whereas the reverse is true for high density ones. In some cases, one algorithm turns out to be one order of magnitude faster than the other (see, for example, results for instances 120_d5 and 200_d50). The only instance coming from Fan and Watson (2012) that could not be solved by SABC is IEEE-Bus-300. That happens despite the fact that LP bound \( R^*_1 \) matches the optimal integer solution value of 129. However, that bound is only obtained when the exact separation of GSECs is carried out. Therefore, this particular instance is in disagreement with our previous observation that, overall, SABC results tend to be better when only the heuristic separation of GSECs is enforced. However, under that supposedly better setting, SABC does not solve that particular instance and does not even improve the initial upper bound provided by the heuristic. Contrary to that, if the exact separation of GSECs is enforced, SABC solves IEEE-Bus-300 easily, in 230 seconds.

We now discuss how the two Benders decomposition algorithms compare to each other. Optimality certificates are produced by SABE and IPBE, respectively, for 37 and 39 of the 47 tested instances. As one may observe from the results in Table 4, and as it is normally the case for Benders decomposition algorithms, SABE and IPBE usually perform well if just a few feasibility cuts are required to attain optimality. For the instances in our test bed, IPBE is usually faster than SABE. Additionally, four new optimality certificates are attained by IPBE for instances where \( n = 200 \) and \( d \leq 30 \). Although 200_d10 is not solved by either IPBE or by SABE within the time limit imposed, these algorithms, respectively, require 29,450 and 24,550 CPU seconds to find the proven optimal solution (\( z = 16 \)). Instance 200_d5, which is not solved by SABE within the time limit, is actually solved by that algorithm after 4,460 seconds. These two instances cannot be solved by branch-and-cut algorithms, including those found in the literature, even after relaxing the time limit constraints. Instances in Fan and Watson (2012) with 57 or more vertices (i.e., all instances in that reference except for IEEE-14-Bus and IEEE-30-Bus) seem to be difficult for both SABE and IPBE. This is probably explained by the very low densities involved (for example, the input graph for instance IEEE-300-Bus has density less than 1%).

In this study, with five exceptions among the instances introduced in Lucena et al. (2010) (namely, 30_d10, 50_d5, 50_d10, 70_d5, and 100_d5) and four exceptions among those coming from Fan and Watson (2012), Benders decomposition algorithms outperform branch-and-cut algorithms. Although the Benders decomposition algorithms are, in many cases, up to four orders of magnitude faster than branch-and-cut algorithms, they require far more time to solve two of the instance exceptions quoted earlier and do not manage to solve the remaining three. Contrary to that, instances 30_d10, 50_d5, 50_d10, and 70_d5 and all instances in Fan and Watson (2012), except for IEEE-300-Bus, are solved quite easily, within three seconds, by the branch-and-cut algorithms SABC and IPBC. On the other hand, larger previously unsolved MCDSP instances, coming from Lucena et al. (2010), can now be solved to optimality by our Benders decomposition algorithms.

We now discuss the computational results obtained by the hybrid algorithms. SAHY managed to solve 40 of the 47 test instances available. IPHY, in addition to these 40 instances, also managed to solve 100_d5, thus reaching a total of 41 instances solved. Accordingly, the hybrid algorithms attained the highest success rates among the algorithms investigated in this study. Considering the instances solved to optimality, once again, the iterative probing strategy produces much better results, in terms of average CPU times. For the instances that are not solved within the time limit by at least one of the two approaches, IPHY always finds a better, or at least equal, upper bound (see the results for instances IEEE-300-Bus, 150_d5, and 200_d5
The hybrid algorithms usually perform better than the Benders algorithms for those instances in Fan and Watson (2012) and also for the remaining test instances with 100 vertices or less (for which the Benders methods need to solve a large number of master problems). When the number of vertices is larger, the hybrid algorithms struggle to solve the corresponding instances, in a similar way, but not to the same extent, as the branch-and-cut algorithms do. Consider instance IEEE-300-Bus as an example; although IPHY fails to solve it within the imposed time limit, it is the only algorithm together with IPBC, among the ones tested here, that managed to improve on the initial upper bound.

It is noteworthy that, sometimes, when the initial upper bounds provided by the heuristic method are already the optimal values, the Benders algorithms need to solve fewer master problems than the hybrid methods, despite the fact that each master problem in SAHY and IPHY separates constraints that reinforce connectivity. For example, the results for instance 70d_30 in Tables 4 and 5 show that SABE and IPBE need just one master problem to solve this instance, whereas SAHY and IPHY require two.

Finally, the branch-and-cut algorithm based on the MTZ model does not seem to be competitive with the other approaches investigated here. It only managed to solve 35 of the 47 test instances available, i.e., the overall lowest success rate. Furthermore, compared to the other algorithms, it presents the worst CPU running times for about 70% of the test instances.

To summarize, our computational experiments show that our stand-alone branch-and-cut algorithm is competitive with other branch-and-cut algorithms in the literature, and better overall results are obtained by the iterative probing variant of Benders decomposition, although very sparse instances are solved much faster by the branch-and-cut method. The iterative probing version of the hybrid algorithm provides a robust method: it scores the best success rate among all algorithms tested, and although it is rarely the fastest algorithm for any given instance, it is rarely the worst. In fact, for instances with up to 120 vertices, it is the only algorithm among the six we investigated that finds proven optimal solutions for all of them. For instances in Lucena et al. (2010) with 150 and 200 vertices, the iterative probing version of the Benders algorithm gives the best results among all algorithms, including those proposed in the literature. However, for the instances in Fan and Watson (2012), all of them having very low densities, the Benders algorithms do not perform well and are dominated by the branch-and-cut and hybrid algorithms introduced here.

8. Conclusions
In this paper, we presented exact algorithms for solving the minimum connected dominating set problem (MCDSP). Two fundamental approaches were described: Benders decomposition and branch-and-cut algorithms. Hybrid algorithms were also developed to take advantage of the best features of both methods. Two variants of the resulting three approaches were designed: a stand-alone version and an iterative probing variant. The latter variant is based on a simple property of the MCDSP, which states that if no connected dominating set of a given cardinality \( d > 0 \) exists, then there are no connected dominating sets of cardinality \( d - 1 \). Overall, six exact algorithms were developed and tested: the stand-alone and iterative probing variants of the Benders decomposition, the branch-and-cut, and the hybrid methods. Our computational experiments showed that the iterative probing variant of Benders and hybrid algorithms performed well on our set of tested instances: for instances with 120 vertices or less, the iterative probing hybrid method is the only algorithm among the six suggested ones that proved optimality for all the instances. For dense instances with more than 120 vertices, the iterative probing Benders approach provided the best results among the six algorithms.

Future work includes the development of specialized separation algorithms for strengthened GSECs (22), cut constraints (23), and \( k \)-partition inequalities (24), which, in the current version of the branch-and-cut algorithm, are generated during the separation procedures for GSECs (17). We also want to investigate the possibility of strengthening the formulation of the Benders master problems by incorporating valid inequalities from the set-covering polytope. Finally, the Benders algorithm, especially in its iterative probing variant, provided very good results on our tested instances, being extremely effective on dense instances. It would be interesting to explore the behavior of a similar approach on other graph optimization problems.

Supplemental Material
Supplemental material to this paper is available at http://dx.doi.org/10.1287/ijoc.2013.0589.

Acknowledgments
The authors thank two anonymous referees whose comments have helped to improve this paper. The authors thank Dr. Mourad Baiou for indicating that the computation of the optimal \( m_s \) coefficient in the feasibility cut is equivalent to solving a Steiner problem in graphs. The authors also thank Dr. Fan for kindly providing the tested instances in Fan and Watson (2012). The first author is partially funded by the Natural Sciences and Engineering Council of Canada (NSERC) Discovery Grants Program [Grant 184122-2010]. The second author is partially funded by CNPq [Grant 310561/2009-4].
and FAPERJ [Grant E26-110.552/2010]. The third author is partially funded by CNPq [Grants 305423/2012-6, 477863/2010-8], and FAPEMIG [Grants PRONEX APQ-01201-09 and PPM-VII-00164-13]. The fourth author is partially funded by FAPERJ and CNPq [Grants 483.243/2010-8, 304793/2011-6]. All sources of financial support are gratefully acknowledged.

References


Berge C (1973) Graphs and Hypergraphs (North Holland, Amsterdam).


