



Solving diameter-constrained minimum spanning tree problems by constraint programming

Thiago F. Noronha^a, Celso C. Ribeiro^b and Andréa C. Santos^c

^aDepartment of Computer Science, Universidade Federal de Minas Gerais, Av. Antônio Carlos 6627, Belo Horizonte, MG 31270-010, Brazil,

^bDepartment of Computer Science, Universidade Federal Fluminense, Rua Passo da Pátria 156, Niterói, 24210-240, Brazil,

^cBlaise Pascal University, LIMOS, Complexe Scientifique des Cézeaux, Aubière 63173, France
E-mail tfn@dcc.ufmg.br [Noronha]; celso@ic.uff.br [Ribeiro]; andrea@isima.fr [Santos]

Received 1 November 2009; received in revised form 16 April 2010; accepted 27 April 2010

Abstract

The diameter-constrained minimum spanning tree problem consists in finding a minimum spanning tree of a given graph, subject to the constraint that the maximum number of edges between any two vertices in the tree is bounded from above by a given constant. This problem typically models network design applications where all vertices communicate with each other at a minimum cost, subject to a given quality requirement. We propose alternative formulations using constraint programming that circumvent weak lower bounds yielded by most mixed-integer programming formulations. Computational results show that the proposed formulation, combined with an appropriate search procedure, solves larger instances and is faster than other approaches in the literature.

Keywords: Spanning trees; diameter constrained spanning trees; bounded-diameter; constraint programming

1. Introduction

Given an undirected connected graph $G = (V, E)$ with a set V of vertices, a set E of edges, and costs c_{ij} associated to every edge $[i, j] \in E$, with $i < j$, the diameter-constrained minimum spanning tree problem (DCMST) consists in finding a minimum spanning tree $T = (V, E')$ of G , with $E' \subseteq E$, whose diameter does not exceed a given positive integer value D , with $2 \leq D \leq |V| - 1$. The diameter of a tree is equal to the number of edges in the longest path between any pair of its nodes. This problem is *NP*-hard for $4 \leq D < |V| - 1$. DCMST applications appear in telecommunications, data compression, and distributed mutual exclusion in parallel computing

(see, e.g., Bookstein and Klein, 2001; Deo and Abdalla, 2000; Raymond, 1989; Wang and Lang, 1994).

The first DCMST formulations, using single commodity flows, were presented in Achuthan et al. (1992). Improved formulations with valid inequalities and a lifting procedure are found in Achuthan et al. (1994) and Santos et al. (2004). An alternative formulation for the odd D case was also proposed in Santos et al. (2004). Multicommodity flow formulations with tighter linear programming (LP) relaxations are presented in Gouveia and Magnanti (2000, 2003). However, they require more memory and computation time to solve the linear relaxation. Formulations strengthened with valid inequalities were proposed in Gruber and Raidl (2005), where they are compared with other formulations. The computational results showed that no approach dominates any other. The formulations using single commodity flow produce weak LP relaxations, especially for small diameters. Multicommodity flow formulations yield tighter lower bounds, but require much more memory and time to solve the linear relaxation, because of their large number of variables and constraints. Recently, Gouveia et al. (2009) modeled DCMST as a Steiner tree problem on a layered graph.

Constraint programming is a paradigm for formulating and solving combinatorial problems, whose fundamentals can be found, e.g., in Walsh et al. (2006) and Hooker (2007). A successful application to round-robin tournament scheduling appeared in Trick (2003). Constraint programming formulations allow high-level constraints involving variable multiplications, logic implications, and indexation of arrays with variables, besides the linear constraints that can be found in mixed-integer programming (MIP) formulations. Instead of using only the linear relaxation for pruning the search tree, it uses a variety of bounding techniques based on constraint propagation. This strategy consists in operating with constraints to generate new constraints in order to reduce the domain of some variables and, consequently, the size of the search space. The new constraints are further propagated to reduce the domain of other variables. The procedure is repeated until no further constraint can be generated. Each value eliminated from the domain of a variable corresponds to a branch of the search tree that is pruned.

In this paper, we propose a new approach based on constraint programming for solving the DCMST. Some MIP formulations for DCMST are presented in Section 2. The constraint programming algorithm is proposed in Section 3. Computational experiments illustrating the comparison of the two approaches are reported in Section 4. Concluding remarks are drawn in Section 5.

2. MIP formulations

This section presents two of the strongest MIP formulations for DCMST in the literature. The results obtained by the branch-and-cut algorithms based on these formulations are later compared with the constraint programming algorithm proposed in the next section.

The MIP formulations in Santos et al. (2004) implicitly use a property proposed in Handler (1978) to ensure that a *central vertex* $v \in V$ exists in any feasible tree T when D is even, such that no vertex is more than $D/2$ edges away from v . If D is odd, a *central edge* $e = [a, b] \in E$ exists in T , such that no vertex is more than $(D - 1)/2$ edges away from the closest extremity of edge e . These formulations rely on an artificial vertex r to model the central vertex of the spanning tree if D is

even or the central edge of the spanning tree if D is odd. The artificial vertex is connected to exactly one vertex (the central vertex) in any integer solution when D is even. In the odd D case, the artificial vertex is connected to exactly two vertices (the extremities of the central edge).

The following formulations make use of a directed graph $G' = (V', A')$ obtained from the original undirected graph $G = (V, E)$ as follows. For every edge $e = [i, j] \in E$, with $i < j$, there are two arcs $(i, j) \in A$ and $(j, i) \in A$, with costs $c_{ij} = c_{ji}$. Then, $V' = V \cup \{r\}$ and $A' = A \cup \{r, 1\}, \dots, (r, |V|)\}$, where r is an artificial root. Furthermore, $c_{ri} = 0$ for every $i \in V$. We also define a constant $L = (D/2)$.

Let the *parent* of a vertex $i \in V$ be the first node $j \in V$ found in the path from i to the root r and let the *height* of a vertex i be the number of edges in the path from the artificial vertex r to $i \in V$. The binary variable x_{ij} is associated to each arc $(i, j) \in A'$ such that $x_{ij} = 1$ if vertex i is the parent of vertex j in the minimum spanning tree; $x_{ij} = 0$ otherwise. The integer variable $u_i, \forall i \in V$, denotes the height of vertex i . The resulting MIP formulation is

$$\min \sum_{(i,j) \in A} c_{ij}x_{ij}, \tag{1}$$

$$\sum_{j \in V} x_{rj} = 1, \tag{2}$$

$$\sum_{(i,j) \in A'} x_{ij} = 1, \quad \forall j \in V, \tag{3}$$

$$u_i - u_j + (L + 1)x_{ij} \leq L, \quad \forall (i, j) \in A', \tag{4}$$

$$0 \leq u_i \leq L + 1, \quad \forall i \in V', \tag{5}$$

$$x_{ij} \in \{0, 1\}, \quad \forall (i, j) \in A'. \tag{6}$$

The objective function is stated in (1). Constraint (2) ensures that the artificial vertex r is connected to exactly one vertex in V (the central vertex). Constraints (3) establish that exactly one arc is incident to each vertex in V . Constraints (4) ensure that the paths in a feasible spanning tree from the artificial vertex r to each vertex $i \in V$ have at most $L + 1$ arcs. Constraints (5) and (6) express the lower and upper bounds on variables u and the integrality requirements on variables x , respectively. The edges $[i, j] \in E$ such that $x_{ij} = 1$ or $x_{ji} = 1$ in a feasible solution to (2)–(6) define a spanning tree T of G with diameter less than or equal to D .

We now consider the odd D case formulation. Let z_{ij} be a binary variable associated with each edge $[i, j] \in E$, with $i < j$: $z_{ij} = 1$, if edge $[i, j]$ is selected as the central spanning tree edge; $z_{ij} = 0$ otherwise. The MIP formulation for the odd D case is

$$\min \sum_{(i,j) \in A} c_{ij}x_{ij} + \sum_{[i,j] \in E} c_{ij}z_{ij}, \tag{7}$$

$$\sum_{j \in V} x_{rj} = 2, \tag{8}$$

$$\sum_{(i,j) \in A'} x_{ij} = 1, \quad \forall j \in V, \quad (9)$$

$$\sum_{[i,j] \in E} z_{ij} = 1, \quad (10)$$

$$z_{ij} \geq x_{ri} + x_{rj} - 1, \quad \forall [i,j] \in E, \quad (11)$$

$$z_{ij} \leq x_{ri}, \quad \forall [i,j] \in E, \quad (12)$$

$$z_{ij} \leq x_{rj}, \quad \forall [i,j] \in E, \quad (13)$$

$$u_i - u_j + (L + 1)x_{ij} \leq L, \quad \forall (i,j) \in A', \quad (14)$$

$$0 \leq u_i \leq L + 1, \quad \forall i \in V', \quad (15)$$

$$x_{ij} \in \{0, 1\}, \quad \forall (i,j) \in A', \quad (16)$$

$$z_{ij} \in \{0, 1\}, \quad \forall [i,j] \in E. \quad (17)$$

The objective function is stated in (7). Constraint (8) establishes that the artificial vertex r is connected to exactly two vertices in V . Constraints (9) establish that there is exactly one arc incident to each vertex in V . Constraints (10)–(13) ensure that $z_{ij} = 1$ only if edge $[i,j]$ is selected as the central spanning tree edge. Constraints (14) ensure that the path from the artificial vertex r to any vertex $i \in V$ in a feasible spanning tree has at most $L+1$ arcs. Constraints (15)–(17) express the lower and upper bounds on variables u and the integrality requirements on variables x and z .

Valid inequalities and lifting procedures that improve the efficiency of both formulations were proposed in Santos et al. (2004). First, the Miller–Tucker–Zemlin (Miller et al., 1960) inequalities (constraints (4) and (14)) were lifted according to Desrochers and Laporte (1991), resulting in

$$u_i - u_j + (L + 1)x_{ij} + (L - 1)x_{ji} \leq L, \quad \forall (i,j) \in A'. \quad (18)$$

Next, generalized upper bounds for variables u_i were proposed. Constraints (19) state that if $i \in V$ is the central vertex then $u_i = 1$, while constraints (20) ensure that if $i \in V$ is not a leaf of the spanning tree, then $u_i \leq L$:

$$u_i \leq (L + 1) - Lx_{ri}, \quad \forall i \in V, \quad (19)$$

$$u_i \leq (L + 1) - x_{ij}, \quad \forall (i,j) \in A. \quad (20)$$

Finally, generalized lower bounds for the variables u_i were applied. Constraints (21) state that if $i \in V$ is the central vertex ($x_{ri} = 1$), then $u_i \geq 1$; else $u_i \geq 2$:

$$u_i \geq x_{ri} + 2 \sum_{j \in V} x_{ji}, \quad \forall i \in V. \quad (21)$$

Gruber and Raidl (2005) proposed two MIP formulations for the even and odd cases, also based on a central vertex (in the even D case) or in a central edge (in the odd D case). However, they do not introduce an artificial vertex. In their formulations, the height of a vertex i is the number of edges in the path from the central vertex v to vertex $i \in V$ when D is even (respectively the extremities of the central edge $e = [a, b]$ to vertex i when D is odd). The height of the central vertex (even case) and the height of the extremities of the central edge (odd case) are equal to one in a feasible tree T . Their formulations make use of a directed graph $G'' = (V, A)$, where V is the set of vertices and A is the set of bidirectional arcs described above. The same binary variables x_{ij}

are associated with every arc $(i, j) \in A$: $x_{ij} = 1$, if the vertex i is the parent of the vertex j in the minimum spanning tree; $x_{ij} = 0$ otherwise. Furthermore, $h_{i\ell} = 1$ if the height of vertex i is equal to ℓ , for all $i \in V$ and $\ell \in 1, \dots, L+1$. The MIP formulation for the even D case is given below:

$$\min \sum_{(i,j) \in A} c_{ij}x_{ij}, \tag{22}$$

$$\sum_{i \in V} h_{i1} = 1, \tag{23}$$

$$\sum_{\ell=1}^{L+1} h_{i\ell} = 1, \quad \forall i \in V, \tag{24}$$

$$\sum_{(i,j) \in A} x_{ij} = 1 - h_{j1}, \quad \forall j \in V, \tag{25}$$

$$x_{ij} \leq 1 - h_{j\ell} + h_{i,\ell-1}, \quad \forall (i,j) \in A, \ell \in \{2, \dots, L+1\}, \tag{26}$$

$$h_{i\ell} \in \{0, 1\}, \quad \forall i \in V, \ell \in \{1, \dots, L+1\}, \tag{27}$$

$$x_{ij} \in \{0, 1\}, \quad \forall (i,j) \in A. \tag{28}$$

The objective function is stated in (22). Constraint (23) establishes that there is exactly one central vertex. Constraints (24) guarantee that each node has exactly a single height. Constraints (25) ensure that exactly one arc is incident to each vertex in V , with the exception of the central vertex. Constraints (26) guarantee that the height of every vertex $i \in V$ in the tree is equal to one plus the height of its parent. Constraints (27) and (28) express the integrality requirements on variables h and x , respectively. The edges $[i, j] \in E$ such that $x_{ij} = 1$ or $x_{ji} = 1$ in a feasible solution to (23)–(28) define a spanning tree T of G with diameter less than or equal to D .

We now consider the odd D case formulation. Let z_{ij} be a binary variable associated to each edge $[i, j] \in E$, with $i < j$: $z_{ij} = 1$, if edge $[i, j]$ is selected as the central spanning tree edge; $z_{ij} = 0$ otherwise. The MIP formulation for the odd D case is

$$\min \sum_{(i,j) \in A} c_{ij}x_{ij} + \sum_{[i,j] \in E} c_{ij}r_{ij}, \tag{29}$$

$$\sum_{i \in V} h_{i1} = 2, \tag{30}$$

$$\sum_{[i,j] \in E} z_{ij} + \sum_{[j,i] \in E} z_{ji} = h_{i1}, \quad \forall i \in V, \tag{31}$$

$$\sum_{\ell=1}^{L+1} h_{i\ell} = 1, \quad \forall i \in V, \tag{32}$$

$$\sum_{(i,j) \in A} x_{ij} = 1 - h_{j1}, \quad \forall j \in V, \tag{33}$$

$$x_{ij} \leq 1 - h_{j\ell} + h_{i,\ell-1}, \quad \forall (i,j) \in A, \ell \in \{2, \dots, L+1\}, \tag{34}$$

$$h_{i\ell} \in \{0, 1\}, \quad \forall i \in V, \ell \in \{1, \dots, L+1\}, \tag{35}$$

$$x_{ij} \in \{0, 1\}, \quad \forall (i, j) \in A, \quad (36)$$

$$z_{ij} \in \{0, 1\}, \quad \forall [i, j] \in E. \quad (37)$$

The objective function is stated in (29). Constraint (30) establishes that the endnodes of the central edge are the only two nodes whose height is equal to one. Constraints (31) ensure that if the heights of nodes $i, j \in V$ are equal to one, then $z_{ij} = 1$. Constraints (32)–(34) are similar to (24)–(26) in the previous formulation. Constraints (35)–(37) are the integrality requirements on variables h , x , and z .

Constraints (30) and (31) imply

$$\sum_{[i,j] \in E} z_{ij} = 1. \quad (38)$$

Although the latter does not strengthen the LP relaxation, computational experiments in Gruber and Raidl (2005) showed that it speeds up the integer optimization.

Gouveia and Magnanti (2003) pointed out that whenever D is odd, the vertices whose heights are equal to two are always connected to the nearest extremity of the central edge. For every $(i, j) \in A$, let $K(i, j)$ be the set of potential central edges $e = [i, p]$ such that $K(i, j) = \{[i, p] \in E : i < p \text{ and } j \neq p \text{ and } c_{ij} \leq c_{pj}\}$. Constraints (39) ensure that if $h_{j1} = 1$, then x_{ij} may only be set to one if the central edge is in $K(i, j)$:

$$x_{ij} \leq 1 - h_{j1} + \sum_{e \in K(i,j)} z_e, \quad \forall (i, j) \in A. \quad (39)$$

The LP relaxation of the above formulations can be strengthened by constraints

$$h_{j1} \leq 1 - x_{ij}, \quad \forall (i, j) \in A, \quad (40)$$

$$h_{i,L+1} \leq 1 - x_{ij}, \quad \forall (i, j) \in A. \quad (41)$$

These constraints state, respectively, that a vertex with height equal to one has no parent and that vertices with height equal to $L+1$ are not parents of any other vertices.

Achuthan et al. (1994) strengthened the LP relaxation by adding cycle elimination constraints, because there are no cycles in a tree. Let C be any subset of edges from E defining a cycle. Then, the corresponding cycle elimination constraint can be formulated as

$$\sum_{[i,j] \in C} (x_{ij} + x_{ji} + z_{ij}) \leq |C| - 1 \quad (42)$$

in the odd D case, or as

$$\sum_{[i,j] \in C} (x_{ij} + x_{ji}) \leq |C| - 1 \quad (43)$$

in the even D case.

Gruber and Raidl (2005) strengthened the LP relaxation by adding connection constraints, because all vertices in the tree are connected. For any non-empty set $S \subset V$ of vertices, the

corresponding connection constraint can be formulated as

$$\sum_{(i,j):i \in S, j \in V \setminus S} (x_{ij} + z_{ij}) \geq 1 \quad (44)$$

in the odd D case, or as

$$\sum_{(i,j):i \in S, j \in V \setminus S} x_{ij} \geq 1 \quad (45)$$

in the even D case.

Two branch-and-cut algorithms have been developed in Gruber and Raidl (2005). The first progressively adds violated cycle elimination cuts (42)–(43) and violated connection cuts (44)–(45) to the formulation of Santos et al. (2004) defined by (7)–(21) for the odd D case and by (1)–(6), (18)–(21) for the even D case. The second algorithm adds the same violated cuts to the formulation of Gruber and Raidl (2005) defined by (29)–(41) for the odd D case and by (22)–(28), (39)–(41) for the even D case.

Four versions of each branch-and-cut algorithm have been evaluated in the computational experiments performed in Gruber and Raidl (2005): (i) without any cuts, (ii) using only connection cuts, (iii) using only cycle elimination cuts, and (iv) using both types of cuts. No approach dominated the others in the computational experiments. However, the algorithms based on the formulation of Gruber and Raidl (2005) performed better for most of the instances with small values of D , while the algorithm based on the formulation of Santos et al. (2004) performed better for most of the instances with large values of D .

3. Constraint programming formulation

In this section, we propose a new approach, based on constraint programming, to tackle DCMST for both the odd and even D cases. It is motivated by the fact that some MIP formulations provide weak lower bounds, while others require a huge amount of memory because of the large number of variables and constraints. Constraint programming alleviates these drawbacks by allowing concise formulations with a small number of variables and by using constraint propagation for pruning the search space, instead of the linear relaxation. As it is based on a backtracking procedure, constraint programming requires much less memory than branch-and-bound algorithms.

The directed graph $G' = (V', A')$ is obtained from the original undirected graph $G = (V, E)$ as described in the previous section. In addition, we define $\delta^-(i)$ as the set of all vertices $j \in V'$ such that $(j, i) \in A'$ and $\delta^+(i)$ as the set of all vertices $j \in V'$ such that $(i, j) \in A'$.

We give below the alternative formulation of DCMST using the OPL language (Van Hentenryck, 1999). Variables a and b denote the central vertices of the spanning tree: in the odd D case, $e = [a, b]$ denotes the central edge of the spanning tree, while in the even D case, $a = b$ denotes the unique central vertex. We represent the parent of vertex $i \in V$ by variable $y_i \in V'$. Furthermore, variable $u_i \in \{0, \dots, L+1\}$ represents the height of vertex $i \in V'$ in the tree. Sets V , V' , $\delta^+(i)$, and $\delta^-(i)$ are designated by nodes, nodes_p, forwardStar[i], and backwardStar[i], respectively.

The artificial vertex is represented by r . Each cost c_{ij} is denoted by $\text{cost}[i, j]$. In case $i = j$, then $\text{cost}[i, j]$ returns zero:

```

var nodes a;
var nodes b;
var nodes_p y[nodes];
var int u[nodes_p] in 0..L + 1;
minimize
    sum(i in nodes) cost[i, y[i]] + cost[a, b] (46)
subject to {
    forall(i in nodes) u[i] = u[y[i]] + 1; (47)
    forall(i in nodes) y[i] in backwardStar[i]; (48)
    sum(i in nodes) (y[i] = r) = 1 + (D mod 2); (49)
    if D mod 2 = 1 then a < b else a = b end if; (50)
    y[a] = y[b] = r; (51)
};

```

The objective function is handled by (46). Constraints (47) establish that the height of every vertex $i \in V$ in the tree is equal to one plus the height of its parent. Constraints (48) ensure that there is an edge $e \in E$ connecting every vertex i with its parent. Constraint (49) specifies that the artificial vertex r is connected either to two vertices (i.e., the extremities of the central edge) when D is odd, or to exactly one vertex (i.e., the central vertex) when D is even. If D is even, constraint (50) determines that $a = b$, otherwise it ensures that a and b are different, with a being smaller than b to break symmetries. Constraints (51) establish that vertex r is the parent of vertices a and b .

Solving a combinatorial optimization problem by constraint programming involves two steps: generating the set of constraints that must be satisfied and describing how to search for solutions. The above formulation in OPL gives the set of constraints that must be satisfied, i.e., it describes the search space.

The search procedure in Fig. 1 implicitly enumerates all solutions in the search space. It defines a search tree, in which each node introduces a new constraint that sets a value to a variable u_i or eliminates the value from its domain. Only the values of variables u_i have to be enumerated, because the parent of each vertex $i \in V$ is the closest vertex j such that $u_j = u_i - 1$. The search tree is explored by a depth-first search strategy. The new constraint is propagated at each node, making an attempt to reduce the domain of other variables and, consequently, pruning nodes of the search tree. If the new constraint leads the subproblem to infeasibility or a feasible solution, then the algorithm backtracks.

Line 2 sets the height of the artificial vertex to zero. Lines 3–4 investigate every pair of vertices a and b to act as the central vertices. Lines 5–7 define the order in which the nodes of the search tree are explored. The constraints handled by lines 8–11 are redundant, but act as useful cuts to reduce the number of nodes in the search tree. They eliminate from the domain of y_j every vertex k such that there is another vertex i which is a better parent to j than k , i.e., $c_{ij} < c_{kj}$ and $u_i \leq u_k$.


```

1  search {
2    u[r]=0;
3    tryall(<A,B> in nodePair : (D mod 2=1 & A<B) \\/ (D mod 2=0 & A=B)
         ordered by increasing sumShortestPaths[A,B]) {
4      a=A & b=B;
5      forall(h in 1..L+1 ordered by increasing h) {
6        forall(i in nodes : not bound(u[i]) & isInDomain(u[i],h)
              ordered by decreasing minl(shortestPath[i,a],shortestPath[i,b]))
7          try u[i]<>h | u[i]=h endtry;
8          forall(i in nodes : bound(u[i]) & u[i]=h)
9            forall(j in forwardStar[i] : isInDomain(y[j],i))
10             forall(k in dominatedBy[i,j] : isInDomain(y[j],k))
11               y[j]<>k;
12     };
13 };
14 };

```

Fig. 1. Search procedure for DCMST.

There are $O(2^{|E|})$ nodes to be evaluated in a search tree generated by a branch-and-bound algorithm based on binary variables associated with the edges of the minimum spanning tree. When solving DCMST by constraint programming, we make use of a different representation of the search space that reduces the size of the search tree for instances with small diameters. In the even D case, the minimum spanning tree can be represented by the central vertex and by the heights of the other vertices, which take at most $L = (D/2)$ different values in the range $[2, L+1]$. As any vertex in V may act as the central vertex, there are $O(|V| \cdot L^{|V|-1})$ nodes to be explored. Analogously, the minimum spanning tree can be represented in the odd D case by the two extremities of the central edge and by the heights of the other vertices. As any of the $|E|$ edges may act as the central edge, there are $O(|E| \cdot L^{|V|-2})$ nodes to be explored in the search tree in this case.

Therefore, the search space considered by constraint programming is much smaller than the search tree generated by a branch-and-bound algorithm whenever the diameter D is small, which makes the approach very attractive in such a situation.

4. Computational results

The computational experiments were carried out on a Pentium 4 with 3.0 GHz clock and 1MB of RAM memory, using OPL Studio 3.7.1 as the constraint programming solver. We compared our results with the best known solutions obtained by MIP approaches in Gouveia and Magnanti (2003), Gruber and Raidl (2005), and Santos et al. (2004).

Tables 1 (even D case) and 2 (odd D case) present numerical results using the same instances considered in Gruber and Raidl (2005) and Santos et al. (2004). Tables 1 and 2 display results for complete and sparse graph instances, while Table 3 shows results exclusively for sparse graph

Table 1

Comparisons with the best among the eight formulations in Gruber and Raidl (2005) and Santos et al. (2004) for even diameters (results extracted from table 1 of Gruber and Raidl, 2005)

V	E	D	Constraint programming (CP)			Best MIP approach		
			Nodes	Memory	Time (s)	Time (s)	Version	CP/MIP (%)
15	105	4	1044	463,780	0.08	0.7	ILP	11.43
15	105	6	6960	479,800	0.28	8.1	ILP	3.46
15	105	10	11,830	511,840	0.41	1.0	Santos+	41.00
20	190	4	3143	607,960	0.2	2.5	ILP	8.00
20	190	6	35,383	672,040	2.03	95.0	ILP	2.14
20	190	10	151,969	672,040	6.08	29.7	Santos+*	20.47
25	300	4	28,842	800,200	1.48	12.0	ILP	12.33
25	300	6	534,222	864,280	39.14	26.4	ILP	148.26
25	300	10	1,126,130	944,380	55.47	254.8	Santos+	21.77
20	50	4	389	466,784	0.05	0.2	ILP	25.00
20	50	6	2678	498,824	0.13	0.8	Santos+	16.25
20	50	10	17,937	514,844	0.64	0.2	Santos+	320.00
40	100	4	130,480	911,340	5.44	1.9	ILP	286.32
40	100	6	91,022	943,380	4.72	13.2	ILP+	35.76
							Average:	68.01

Table 2

Comparisons with the best among the eight formulations in Gruber and Raidl (2005) and Santos et al. (2004) for odd diameters (results extracted from table 1 of Gruber and Raidl, 2005)

V	E	D	Constraint programming (CP)			Best MIP approach		
			Nodes	Memory	Time (s)	Time (s)	Version	CP/MIP (%)
15	105	5	2850	463,780	0.22	3.0	ILP	7.33
15	105	7	8240	527,860	0.38	20.0	ILP+*	1.90
15	105	9	11,743	527,860	0.47	6.2	Santos+*	7.58
20	190	5	18,283	640,000	1.06	8.1	ILP	13.09
20	190	7	19,142	688,060	0.97	5.5	ILP*	17.64
20	190	9	119,906	752,140	5.01	66.7	ILP+*	7.51
25	300	5	37,608	864,280	2.83	64.3	ILP+*	4.40
25	300	7	812,957	979,424	56.06	770.5	ILP	7.28
25	300	9	2,655,810	1,043,504	114.14	246.0	Santos+	46.40
20	50	5	3611	466,784	0.17	1.0	ILP	17.00
20	50	7	1975	498,824	0.14	0.8	Santos+	17.50
20	50	9	13,040	495,820	0.45	0.7	Santos+	64.29
40	100	5	161,961	927,360	7.31	6.4	ILP	114.22
40	100	7	778,699	975,420	34.38	212.4	ILP	16.19
40	100	9	769,161	1,007,460	40.16	979.8	ILP*	4.10
							Average:	23.10

instances. For each instance, the first three columns present its number of vertices, its number of edges, and the value of its diameter, respectively. The three next columns summarize the results obtained using the constraint programming approach: the number of nodes visited in the search

Table 3

Comparisons with the formulations in Gouveia and Magnanti (2003) and Gruber and Raidl (2005) (results extracted from table 2 of Gruber and Raidl, 2005)

V	E	D	Constraint programming (CP)			MIP (Gouveia and Magnanti, 2003)		Best MIP (Gruber and Raidl, 2005)			
			Nodes	Memory	Time (s)	Time (s)	CP/MIP (%)	Time (s)	Version	CP/MIP (%)	
Random											
20	100	4	157	444,976	0.09	0.5	18.00	0.9	ILP+	10.00	
20	100	5	886	465,076	0.23	6.3	3.65	2.4	ILP*	9.58	
20	100	6	2602	473,116	0.27	5.8	4.66	2.9	ILP	9.31	
20	100	7	3875	489,196	0.38	94.0	0.40	2.6	ILP*	14.62	
30	100	4	213	712,359	0.19	0.8	23.75	3.5	ILP	5.43	
30	200	5	1862	753,500	0.86	58.6	1.47	283.8	ILP+	0.30	
30	200	6	2189	761,540	0.47	2.9	16.21	5.7	ILP	8.25	
30	200	7	1535	789,680	0.80	529.4	0.15	53.9	ILP*	1.48	
Euclidean											
20	100	4	667	461,056	0.11	0.1	110.00	1.1	ILP	10.00	
20	100	5	2027	473,116	0.27	5.3	5.09	1.7	ILP	15.88	
20	100	6	2628	485,176	0.22	3.1	7.10	7.3	ILP	3.01	
20	100	7	3988	493,216	0.38	49.5	0.77	10.0	ILP	3.80	
30	200	4	4556	725,360	0.50	130.8	0.38	59.5	ILP*	0.84	
30	200	5	27,468	769,580	2.13	25.1	8.49	36.1	ILP	5.90	
30	200	6	1,235,639	785,660	95.24	1381.9	6.89	348.0	ILP	27.37	
30	200	7	1,570,520	809,780	132.67	6912.1	1.92	1014.4	ILP*	13.08	
							Average:			13.06	8.68

tree, the amount of memory (in bytes) used by the algorithm, and the computation time in seconds required to prove the optimality of the best solution found. The two next columns present the best results obtained with the original MIP formulations in Gruber and Raidl (2005) and Santos et al. (2004) on a Pentium 4 with a 2.8 GHz clock and 2MB of RAM memory, using CPLEX 8.1 as the MIP solver. For each instance, we report the processing time in seconds required to prove optimality, together with an indication of the fastest algorithm version. The best MIP formulations of Gruber and Raidl (2005) and Santos et al. (2004) are denoted by ILP and Santos, respectively. The symbol “+” denotes the use of connection cuts, while an “*” indicates the use of cycle elimination cuts (Gruber and Raidl, 2005). The last column shows the ratio between the processing times needed to find the optimal solution by constraint programming and by the best MIP approach.

In the following discussion, we refer to our constraint programming approach simply as *constraint programming*. It was able to find provably optimal solutions for all instances in Tables 1 and 2. Furthermore, constraint programming dominated the MIP approaches involved in this study, obtaining smaller computation times. On average, constraint programming took only 23% of the time taken by the best MIP approach for the instances with odd diameters, which are often considered as the hardest. For the instances with even diameters, constraint programming required, on average, about 68% of the time taken by the best MIP version. We remark that we compare the results obtained by constraint programming with the best MIP approach for each

individual instance. If the comparisons involved one specific MIP approach, it would be far more favorable to constraint programming.

In general, the larger the time required to prove optimality, the better the performance of constraint programming relatively to all MIP versions. For example, constraint programming required 56 s to find a provably optimal solution for the instance with $|V| = 25$, $|E| = 300$, and $D = 7$, while the best MIP approach required 770 s. For complete graphs, constraint programming took, on average, 21% of the time observed with the best MIP version. The search strategy implemented within the constraint programming solver significantly reduces the size of the search space when the number of nodes is much smaller than the number of edges. This is most likely due to the fact that branching in the constraint programming search tree is performed on the u variables, and not on variables x or z . Furthermore, constraint propagation allows a reduction in the search space.

Table 3 summarizes comparative results involving sparse instances used in Gouveia and Magnanti (2003) and Gruber and Raidl (2005): eight are randomly generated, while the other eight are Euclidean instances as shown in Gouveia and Magnanti (2003). For each instance, the first column indicates its type: random or Euclidean. Its number of vertices, its number of edges, and its diameter are given next. The following three columns summarize the results obtained using the constraint programming approach: the number of nodes visited in the search tree, the amount of memory (in bytes) used by the solver, and the computation time in seconds needed to prove the optimality of the best solution found. The results obtained by the formulations described in Gouveia and Magnanti (2003) follow: the time required to prove optimality (obtained on a different machine) and the ratio between the processing times required to find the optimal solution by constraint programming and by the previous MIP approach, using the times reported in Gruber and Raidl (2005) as references. The last three columns show the results obtained by the formulations described in Gruber and Raidl (2005): the processing time in seconds required to prove optimality, together with an indication of the fastest algorithm version (once again, the symbol “+” stands for the use of connection cuts, while an “*” denotes the use of cycle elimination cuts) and the ratio between the processing times required to find the optimal solution by constraint programming and by the best of the previous MIP formulations, using the times in Gruber and Raidl (2005) as references. Constraint programming performed even better for the instances in Table 3, systematically reducing the time required to find optimal solutions and to prove their optimality. It consumed, on average, 13% of the time taken by the MIP formulation in Gouveia and Magnanti (2003) and only 8% of the time required by the best MIP formulation in Gruber and Raidl (2005).

The algorithms in Gruber and Raidl (2005) are more appropriate to small-diameter instances, while those in Santos et al. (2004) perform better on instances with large diameters. However, the constraint programming approach outperformed both MIP approaches on small- and large-diameter instances. Furthermore, no instance required more than 1MB of RAM memory to be solved by constraint programming, because the search tree is explored by a depth-first search algorithm and no node is stored to be further explored.

5. Conclusions

We proposed a new approach based on constraint programming to solve the DCMST. Constraint programming was able to overcome the main drawbacks of MIP approaches for the instances

considered in this study: first, by using more concise formulations with a smaller number of variables, and second, by using constraint propagation for pruning the search space, instead of using exclusively the bound provided by the linear relaxation. The same constraint programming algorithm is able to handle instances with odd or even diameters.

Constraint programming yielded better results (i.e., smaller computation times to find exact optimal solutions and to prove their optimality) than all MIP approaches for 40 out of the 45 test instances. On average, the computation times required by constraint programming amounted to only 35% of those observed with the best MIP approach.

References

- Achuthan, N.R., Caccetta, L., Caccetta, P.A., Geelen, J.F., 1992. Algorithms for the minimum weight spanning tree with bounded diameter problem. In Phua, K.H., Wand, C.M., Yeong, W.Y., Leong, T.Y., Loh, H.T., Tan, K.C., Chou, F.S. (eds) *Optimization Techniques and Applications*. World Scientific, Singapore, pp. 297–304.
- Achuthan, N.R., Caccetta, L., Caccetta, P.A., Geelen, J.F., 1994. Computational methods for the diameter restricted minimum weight spanning tree problem. *Australasian Journal of Combinatorics* 10, 51–71.
- Bookstein, A., Klein, S.T., 2001. Compression of correlated bitvectors. *Information Systems* 16, 110–118.
- Deo, N., Abdalla, A., 2000. Computing a diameter-constrained minimum spanning tree in parallel. *Lecture Notes in Computer Science* 1767, 17–31.
- Desrochers, M., Laporte, G., 1991. Improvements and extensions to the Miller–Tucker–Zemlin subtour elimination constraints. *Operations Research Letters* 10, 27–36.
- Gouveia, L., Magnanti, T.L., 2000. Modelling and solving the diameter-constrained minimum spanning tree problem. Technical Report, Departamento de Estatística e Investigação Operacional, Faculdade de Ciências, Lisboa.
- Gouveia, L., Magnanti, T.L., 2003. Network flow models for designing diameter-constrained minimum-spanning and Steiner trees. *Networks* 41, 159–173.
- Gouveia, L., Simonetti, L., Uchoa, E. Modeling hop-constrained and diameter-constrained minimum spanning tree problems as Steiner tree problems over layered graphs. *Mathematical Programming*, to appear. Doi: 10.1007/s10107-009-0297-2.
- Gruber, M., Raidl, G.R., 2005. A new 0-1 ILP approach for the bounded diameter minimum spanning tree problem. In Gouveia, L., Mourao, C. (eds) *Proceedings of the 2nd International Network Optimization Conference*. Lisbon, pp. 178–185.
- Handler, G.Y., 1978. Minimax location of a facility in an undirected graph. *Transportation Science* 7, 287–293.
- Hooker, J.N., 2007. *Integrated Methods for Optimization*. Springer, New York.
- Miller, C.E., Tucker, A.W., Zemlin, R.A., 1960. Integer programming formulations and traveling salesman problems. *Journal of the ACM* 7, 326–329.
- Raymond, K., 1989. A tree-based algorithm for distributed mutual exclusion. *ACM Transactions on Computers* 7, 61–77.
- Santos, A.C., Lucena, A.P., Ribeiro, C.C., 2004. Solving diameter constrained minimum spanning tree problem in dense graphs. *Lecture Notes in Computer Science* 3059, 458–467.
- Trick, M.A., 2003. Integer and constraint programming approaches for round-robin tournament scheduling. *Lecture Notes in Computer Science* 2740, 63–77.
- Van Hentenryck, P., 1999. *The OPL Optimization Programming Language*. MIT Press, Cambridge.
- Walsh, T., Rossi, F., Van Beek, P., 2006. *Handbook of Constraint Programming*. Elsevier, Amsterdam.
- Wang, S., Lang, S.D., 1994. A tree-based distributed algorithm for the k-entry critical section problem. In: Ni, L. M. (ed) IEEE Computer Society Press. *International Conference on Parallel and Distributed Systems*. Taiwan, pp. 592–599.