Available online at www.sciencedirect.com

Electronic Notes in Discrete Mathematics 30 (2008) 93–98

Electronic Notes in DISCRETE MATHEMATICS

www.elsevier.com/locate/endm

Constraint Programming for the Diameter Constrained Minimum Spanning Tree Problem

Thiago F. Noronha, Andréa C. Santos

Department of Computer Science, Catholic University of Rio de Janeiro
Rua Marquês de São Vicente 225, Rio de Janeiro, RJ 22453-900, Brazil

Celso C. Ribeiro

Department of Computer Science, Universidade Federal Fluminense
Rua Passo da Pátria 156, Niterói, RJ 24210-240, Brazil
{tfn,cynthia,celso}@inf.puc-rio.br

Abstract

We propose a new formulation for the Diameter Constrained Minimum Spanning Tree Problem using constraint programming. Computational results have shown that this formulation combined with an appropriate search procedure solves larger instances and is faster than the other approaches in the literature.

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 Minimum Spanning Tree Problem* (DCMST) consists in finding a minimum spanning tree $T = (V, E')$, with $E' \subseteq E$, where the *diameter* required does not exceed a given positive integer value D, where $2 \leq D \leq |V| - 1$. The *diameter* of a tree T is equal to the number of edges in the

1571-0653/$ – see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.endm.2008.01.017

longest path between any two nodes $i, j \in V$ in T. This problem is NP-hard when $D \geq 4$. DCMST applications appear in telecommunications, data compression and distributed mutual exclusion in parallel computing.

Some mixed integer programming (MIP) formulations for DCMST implicitly use a property [5] that ensures that a *central vertex* $c \in V$ exists in any feasible tree T when D is even, such that no vertex is more than $D/2$ edges away from c. 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 e.

The first DCMST formulations, using single commodity flows, were presented in [2]. Improved formulations with valid inequalities and a lifting procedure are found in [1,6]. An alternative formulation for the odd D case was also proposed in [6]. Multicommodity flow formulations with tighter linear programming relaxations are presented in [3]. However, they require more memory and computation time to solve the linear relaxation. Formulations strengthened with valid inequalities were proposed in [4]. A comparison with other formulations appears in [4]. The computational results showed that no approach dominates any other. The formulations using single commodity flow produce weak linear programming relaxations, especially for small diameters. On the other hand, the multicommodity flow formulations give tighter lower bounds, but require much more memory and time to solve the linear relaxation, because of their large number of variables and constraints.

Constraint Programming (CP) is a programming paradigm for formulating and solving combinatorial problems. Instead of using only the linear relaxation for pruning the search tree, it uses a variety of bounding techniques based on constraint propagation, which consists in operating with the constraints to generate new constraints that reduce the domain of some variables and, consequently, the size of the search space.

In the next section, we propose a new approach based on constraint programming for solving DCMST. It is capable to handle both the odd and even diameter cases in the same formulation, contrary to most approaches in the literature. Computational experiments are reported in Section 3. Concluding remarks are drawn in Section 4.

2 Constraint programming formulation

We propose a new approach to DCMST based on constraint programming that tackles both the odd and even D cases. It was motivated by the fact that some MIP formulations have weak lower bounds, while others require a huge amount of memory because of the large number of variables and constraints.

CP reduces 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. Since it is based on a backtracking procedure, CP requires less memory than branch-and-bound algorithms.

The directed graph $G' = (V', A')$ is obtained from the original undirected graph $G = (V, E)$ as follows. Let r be an artificial vertex and $V' = V \cup \{r\}$. For every edge $[i, j] \in E$, with $i < j$, there exist two arcs (i, j) and $(j, i) \in A$ with costs $c_{ij} = c_{ji}$. Then, $A' = A \cup \{(r, 1), \ldots, (r, |V|)\}$, with costs $c_{ri} = 0$ for every $i \in V$. For any $i \in V$, we define $backwardStar[i]$ as the set of all vertices $j \in V'$ such that $(j, i) \in A'$ and $forwardStar[i]$ as the set of all vertices $j \in V$ such that $(i, j) \in A'$. Let $L = \lfloor D/2 \rfloor$. The number of edges in the path from the artificial vertex r to $i \in V$ is said to be the *height* of vertex i.

We give below the formulation of DCMST using the ILOG OPL language. Variables a and b denote the central vertices of the spanning tree. When D is odd, $[a, b]$ denotes the central edge of the spanning tree. In the even D case, $a = b$ denotes the central vertex. Variable $y_i \in V'$ denotes de parent of vertex $i \in V$. Furthermore, variable $u_i \in \{0, ..., L+1\}$ represents the height of vertex $i \in V'$. Sets V and V' are designated by `nodes` and `nodes_p`, respectively. The costs c_{ij} are denoted by `cost[i,j]`. In case $i = j$, then `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
(1)    sum(i in nodes) cost[i,y[i]] + cost[a,b]
subject to {
(2)    sum(i in nodes) (y[i]=r) = 1 + (D mod 2);
(3)    forall(i in nodes) u[i]=u[y[i]]+1;
(4)    forall(i in nodes) y[i] in backwardStar[i];
(5)    if D mod 2 = 1 then a<b else a=b endif;
(6)    y[a]=y[b]=r;
};
```

The objective function is handled by (1). Constraint (2) ensures that the artificial vertex r is connected 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. Constraints (3) establish that the height of every vertex $i \in V$ in the tree is equal to one plus the height of its parent. Constraints (4) ensure that there is an edge $e \in E$ connecting every vertex i with its parent. If D is even Constraint (5) establishes that a is equal b, otherwise it ensures that a is smaller (different) than b. Constraint (6) establishes 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 gives the set of constraints that must be satisfied, i.e., it describes the search space. For sake of conciseness, the search procedure is omitted.

3 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 over all those obtained by any of the MIP approaches in [4,6], according with the computation times presented in Table 1 of [4] for the same 29 instances used in our study: 18 on complete graphs and 11 on sparse graph, with diameters equal to 4, 5, 6, 7, 9, and 10.

Numerical results are presented on Table 1. For each instance, the first three columns give its number of vertices, its number of edges, and its maximum diameter, respectively. The next three columns give statistics for the CP formulation: the number of nodes visited in the search tree, the amount of memory (in bytes) used by the algorithm, and the computation time in seconds to prove optimality. The next two columns give the time in seconds (on a Pentium 4 with 2.8 GHz clock and 2Mb of RAM memory, using CPLEX 8.1 as MIP solver) to prove optimality by the best MIP formulation in [4,6] and the corresponding algorithm version. The best MIP formulations of [4] and [6] are denoted by `ILP` and `Santos`, respectively (the symbol '+' denotes the use of connect cuts, while an '*' indicates the use of cycle elimination cuts [4]). The last column shows the ratios between the computation times to find the optimal solution with constraint programming and the best MIP approach.

CP performed better than the best MIP approach on all but four out of the 29 test instances. On average, the CP approach run on 45% of the time of the best MIP variant. The computation times were particularly remarkable for the instances with odd diameter: for these instances, the time needed by the CP algorithm to prove optimality was only 23% of that taken by the best MIP variant, on average. The larger is the time to prove optimality, the better is the performance of the CP algorithm when compared with all MIP variants.

				CP		Best MIP		
$\|V\|$	$\|E\|$	D	nodes	memory	time (s)	time (s)	version	CP/MIP
15	105	4	1,044	463,780	0.08	0.7	ILP	11.43%
15	105	5	2,850	463,780	0.22	3.0	ILP	7.33%
15	105	6	6,960	479,800	0.28	8.1	ILP	3.46%
15	105	7	8,240	527,860	0.38	20.0	ILP+*	1.90%
15	105	9	11,743	527,860	0.47	6.2	Santos+*	7.58%
15	105	10	11,830	511,840	0.41	1.0	Santos+	41.00%
20	190	4	3,143	607,960	0.2	2.5	ILP	8.00%
20	190	5	18,283	640,000	1.06	8.1	ILP	13.09%
20	190	6	35,383	672,040	2.03	95.0	ILP	2.14%
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%
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	5	37,608	864,280	2.83	64.3	ILP+*	4.40%
25	300	6	534,222	864,280	39.14	26.4	ILP	148.26%
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%
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	5	3,611	466,784	0.17	1.0	ILP	17.00%
20	50	6	2,678	498,824	0.13	0.8	Santos+	16.25%
20	50	7	1,975	498,824	0.14	0.8	Santos+	17.50%
20	50	9	13,040	495,820	0.45	0.7	Santos+	64.29%
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	5	161,961	927,360	7.31	6.4	ILP	114.22%
40	100	6	91,022	943,380	4.72	13.2	ILP+	35.76%
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:	44.78 %

Table 1
Numerical results.

For dense graphs, the CP algorithm run on average in 21% of the time of the best MIP variant: the search strategy implemented within the CP algorithm significantly reduces the size of the search space, when the number of nodes is much smaller then the number of edges.

The algorithms in [4] are more appropriate to small diameter instances, while those in [6] perform better on large diameter instances. However, the CP approach surpassed both MIP approaches on small and large diameter instances. Furthermore, no instance required more than 1 Mbyte of RAM memory to be solved, because the search tree is explored by a depth first search algorithm and no node is stored to be further explored.

4 Conclusions

We proposed a new approach based on constraint programming to solve the degree constrained minimum spanning tree problem. Constraint programming was capable to overcome the main drawbacks of MIP: 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 the bound provided by the linear relaxation. One single algorithm was capable to handle both the odd and even diameter instances.

Constraint programming obtained better results (i.e., smaller computation times to find exact optimal solutions and to prove their optimality) than all MIP approaches for most (25 out of the 29) of the test instances. On the average, the constraint programming computation times were only 45% of those observed with the MIP approach. The advantage of the CP approach was even stronger for the instances with odd diameter and for those on dense graphs, for which the previous ratio was equal to 23% and 21%, respectively.

References

[1] Achuthan, N., L. Caccetta, P. Caccetta and J. Geelen, *Computational methods for the diameter restricted minimum weight spanning tree problem*, Australasian Journal of Combinatorics **10** (1994), pp. 51–71.

[2] Achuthan, N., L. Caccetta, P. Caccetta and J. F. Geelen, *Algorithms for the minimum weight spanning tree with bounded diameter problem*, in: K. Phua, C. Wand, W. Yeong, T. Leong, H. Loh, K. Tan and F. Chou, editors, *Optimization Techniques and Applications*, World Scientific, Singapore, 1992 pp. 297–304.

[3] Gouveia, L. and T. Magnanti, *Network flow models for designing diameter-constrained minimum-spanning and Steiner trees*, Networks **41** (2003), pp. 159–173.

[4] Gruber, M. and G. Raidl, *A new 0-1 ILP approach for the bounded diameter minimum spanning tree problem*, in: L. Gouveia and C. Mourao, editors, *Proceedings of the 2nd International Network Optimization Conference*, Lisbon, 2005, pp. 178–185.

[5] Handler, G., *Minimax location of a facility in an undirected graph*, Transportation Science **7** (1978), pp. 287–293.

[6] Santos, A., A. Lucena and C. Ribeiro, *Solving diameter constrained minimum spanning tree problem in dense graphs*, Lecture Notes in Computer Science **3059** (2004), pp. 458–467.