DISCRETE
APPLIED
MATHEMATICS

Solving the feedback vertex set problem on undirected graphs

Lorenzo Brunetta, Francesco Maffioli *, Marco Trubian

Dipartimento di Elettronica e Informazione, Politecnico di Milano, Piazza Leonardo da Vinci 32, 20133 Milano, Italy

Received 5 August 1998; revised 21 May 1999; accepted 24 May 1999

Abstract

Feedback vertex problems consist of removing a minimal number of vertices of a directed or undirected graph in order to make it acyclic. The problem is known to be $\mathcal{NP}$-complete. In this paper we consider the variant on undirected graphs. The polyhedral structure of the feedback vertex set polytope is studied. We prove that this polytope is full dimensional and show that some inequalities are facet defining. We describe a new large class of valid constraints, the *subset inequalities*. A branch-and-cut algorithm for the exact solution of the problem is then outlined, and separation algorithms for the inequalities studied in the paper are proposed. A local search heuristic is described next. Finally, we create a library of 1400 randomly generated instances with the geometric structure suggested by the applications, and we computationally compare the two algorithmic approaches on our library.

Keywords: Feedback vertex set; Branch-and-Cut; Local search heuristic; Tabu search

1. Introduction

We consider undirected graphs $G = (V, E)$, where E is the edge set and V is the vertex set, with vertex weights $c_v \in \mathbb{R}$, $v \in V$. We denote by uv the edge of E having u and v as end-points. Two vertices u and v are *adjacent* if there exists an edge $uv \in E$ connecting them. A sequence of vertices $v_1, v_2, \ldots, v_k$ of G is called a *path* if $v_{i-1}v_i \in E$, for $i = 2, \ldots, k$. Vertex v_1 is the *origin* and vertex v_k is the *end* of the path. If $v_1 = v_k$ a path is said to be a *cycle*. A graph G is *acyclic* if it does not contain cycles. A *chord* is an edge $v_iv_j \in E$ connecting two non-consecutive vertices in a cycle. The *vertex degree* of vertex v in G is the number of the edges $uv \in E$ having v as end-point, and it is denoted by $d(v)$.

* Corresponding author.
E-mail address: maffioli@elet.polimi.it (F. Maffioli)

0166-218X/00/$ - see front matter
PII: S0166-218X(99)00180-8

For $y \in \mathbb{R}^V$ and $S \subseteq V$, we indicate with $y(S)$ the sum $\sum_{v \in S} y_v$.

The (undirected) *feedback vertex set* (*UFVS*) problem consists of removing a vertex subset F of minimum weight $c(F) = \sum_{v \in F} c_v$ ($F \subseteq V$) in an undirected graph in order to make it acyclic. We assume that a graph made of a single vertex v does not contain cycles, and its feedback vertex set is the empty set.

The problem is known to be $\mathcal{NP}$-complete (see [14], since the *vertex cover* problem can be reduced to it; there exist polynomial time algorithms for particular topologies [7,16,18].

The relevance of this problem arises in several areas. For example, it models the problem of removing *deadlocks* in a system of processors, see [21]. Applications of the Feedback Vertex Set Problem to constraint satisfaction and Bayesian inference are reported in [2,8]. In telecommunications it is helpful in finding the minimum number of vertices of control for monitoring a network. Another application of the problem with vertex weights equal to 1 (cardinality case) is in the context of operating systems for the removal of *deadlocks* created by cyclical processes' requests of already allocated resources. Finally, it is relevant in the study of "monopolies" in synchronous distributed systems, as introduced in [19,20], where connection networks are undirected graphs of bounded degree, namely, grids and toroidal grids.

Several approximate and heuristic approaches have appeared in the literature on the problem on directed and undirected graphs, see for example [1–3,5,13,17,21]. A polyhedral approach to the FVS problem on directed graphs is presented in [11].

In this paper we describe a system based on a branch-and-cut algorithm for finding a UFVS of minimum weight in an undirected graph and a local search heuristic.

The components of our system are: a set of exact and heuristic procedures for the separation of violated inequalities belonging to a partial description of the UFVS polytope, an enumeration procedure that combines branching with cutting-planes techniques, and an exploring tabu search ($\mathcal{X}-\mathcal{TS}$) procedure (see [9]) to find a good upper bound on the objective function. We describe these components in the following sections. We create a library of 1400 randomly generated instances with the geometric structure suggested by the applications. Finally, we report on our computational experience in Section 5.

2. Polyhedral results

A *subgraph* of G is the graph $G' = (S, E')$, where $S \subseteq V$ and $E' \subseteq E$. The subgraph $G' = (S, E(S))$ is said to be *induced* by the vertex set $S \subseteq V$ if $E(S)$ is the set of edges having both end vertices in S.

We denote by $\mathbb{R}^V$ the real vector space whose components are indexed by the elements of V. With every subset $U \subseteq V$ we associate a *vector* $x^U \in \mathbb{R}^V$ where a component x_v^U is equal to 1 if $v \in U$, and to 0 otherwise.

For a given vertex set A and $B \subseteq A$ the *incidence vector* $\chi^B \in \{0,1\}^A$ of B is defined by setting $\chi_a^B = 1$ if $a \in B$ and $\chi_a^B = 0$ if $a \notin B$.

The *feedback vertex set polytope* $Q(G)$ of the graph G is the convex hull of incidence vectors of all feedback vertex sets of G. Therefore, the minimum weight UFVS of G can be found, in principle, by solving the following linear program:

$$\begin{aligned} &\min \quad cx \\ &\text{s.t.} \quad x \in Q(G). \end{aligned} \tag{1}$$

Recall that, given a vertex set W of G, the set $\delta(W) \subseteq E$ of the edges between W and $V \setminus W$ is called a *cut*. A graph G is *q-connected* if $|\delta(W)| \geqslant q$ for all $\emptyset \subset W \subset V$. This is equivalent to saying that G contains q edge-disjoint paths between any pair of vertices (by definition, a graph made of a single vertex is not q-connected for all q). A *bridge* of G is a cut of size 1.

The affine hull of $Q(G)$ is characterized in the following theorem.

Theorem 1. *If $d(v) \geqslant 2$ for all $v \in V$ of G, then $Q(G)$ is full dimensional, i.e., $dim(Q(G)) = |V|$.*

Proof. If there exists a subset of the vertex set $U_2 = \{v \in V: |\delta(v)| < 2\}$, by definition of feedback vertex set we can reduce the size of the vertex set V by removing the set U_2: let $\tilde{V}$ be the resulting graph. We can repeat this reduction, since the vertices of degree lower than 2, and the edges having at least one end vertex in U_2 do not belong to any cycle of V. This fact reduces the dimension of $Q(G)$ by $|U_2|$.

If G is 2-connected, we can construct a feasible solution for all $v \in V$, since $\chi^{V \setminus \{v\}}$ is a UFVS, thus obtaining $|V|$ solutions. The other point is given by χ^V that is, obviously, a UFVS.

Suppose that G is not 2-connected, then G can be decomposed in 2-connected components linked by bridges. Assume, w.l.o.g., that there exists only one bridge and two shores. The shores of the bridge are the 2-connected graphs $G_1 = (V_1, E(V_1))$ and $G_2 = (V_2, E(V_2))$, with $V = V_1 \cup V_2$ and $V_1 \cap V_2 = \emptyset$, then $Q(G_1)$ and $Q(G_2)$ are full dimensional, and $dim(Q(G)) = dim(Q(G_1)) + dim(Q(G_2))$ is full dimensional. □

The hypothesis of $d(v) \geqslant 2$ for all $v \in V$ seems not to be restrictive considering that the applications described in the literature are usually on 2-connected graphs. The optimal solution to (1) is the incidence vector of an optimal UFVS. Let $\mathscr{C}$ be the set of all cycles C in G. An LP relaxation of $Q(G)$ is given by the following system of inequalities:

$$x(C) \geqslant 1 \quad \text{for all cycles } C \in \mathscr{C}, \tag{2}$$

$$0 \leqslant x \leqslant 1. \tag{3}$$

Consequently, inequalities (2) and (3) define an LP relaxation $\tilde{Q}(G)$ of $Q(G)$ and provide an integer programming formulation for the UFVS problem on G.

Inequalities (2) are called *cycle inequalities*: their polyhedral analysis is in Theorem 4. We first examine the trivial inequalities $0 \leqslant x_v \leqslant 1$ for every $v \in V$.

Theorem 2. *For every vertex $v \in V$, the inequality $x_v \geqslant 0$ is facet defining.*

Proof. Let v be a vertex of V, then $\chi^{V\setminus\{v\}}$ is a UFVS that satisfies $x_v = 0$. We can construct a feasible solution for all $w \in V\setminus\{v\}$, since $\chi^{V\setminus\{w,v\}}$ is a UFVS that satisfies $x_v = 0$. □

Theorem 3. *For every vertex $v \in V$, the inequality $x_v \leqslant 1$ is facet defining.*

Proof. One point is given by χ^V that is a UFVS that satisfies $x_v = 1$. Let v be a given vertex of V, then for all $w \in V\setminus\{v\}$, $\chi^{V\setminus\{w\}}$ is a UFVS that satisfies $x_v = 1$. □

Theorem 4. *For all cycles $C \in \mathscr{C}$, the inequality $x(C) \geqslant 1$ is facet defining if it has no chords.*

Proof. Suppose that a given cycle $C = \{v_1, v_2, \ldots, v_k, v_1\}$ has a chord, i.e., an edge $v_i v_j \in E$ connecting two non-consecutive vertices of C v_i and v_j. Then C can be decomposed in the cycle $C' = \{v_1, v_2, v_i, v_j, \ldots, v_k, v_1\}$ and in $C\setminus C' \cup \{v_i, v_j\}$: $x(C) \geqslant 1$ is just the sum of the inequalities $x(C') \geqslant 1$ and $x_v \geqslant 0$, for all $v \in C\setminus C'$.

Let C be a given cycle with no chords, then for all $v \in C$, $\chi^{V\setminus C\cup\{v\}}$ is a UFVS that satisfies $x(C) = 1$, and we obtain $|C|$ feasible points.

The other $|V\setminus C|$ points can be obtained in the following way. Observe that, if there exists a point $w \in V\setminus C$ and $v_1, v_2 \in C$ such that wv_1 and wv_2 are in E, and $v_3 \in C$, then $\chi^{V\setminus C\setminus\{w\}\cup\{v_3\}}$ satisfies $x(C) = 1$, but it is not a UFVS (since the cycle containing w, v_1, v_2 is not covered by any vertex). In this case we must select a vertex of C adjacent to w, i.e., either v_1 or v_2.

For all the other $w \in V\setminus C$ for which there are no two vertices $v_1, v_2 \in C$ such that wv_1 and wv_2 are in E, and for any $v \in C$, $\chi^{V\setminus C\cup\{w,v\}}$ satisfies $x(C) = 1$ and it is a UFVS. □

To produce an LP relaxation that is tighter than $\tilde{Q}(G)$ we add a large class of inequalities introduced by Goemans and Williamson (see Lemma 9.1 in [13]): let $d_S(v)$ denote the *vertex degree* of vertex v in S, then

$$\sum_{v\in S}(d_S(v) - 1)x_v \geqslant |E(S)| - |S| + 1 \quad \text{for all subsets } S \subseteq V. \tag{4}$$

We call inequalities (4) the *subset inequalities*.

Lemma 5. *The subset inequalities* (4) *are valid for all subsets $S \subseteq V$.*

Proof. Let $\tilde{F}$ be a feasible UFVS for the subgraph $G' = (S, E(S))$ induced by $S \subseteq V$. Removing $\tilde{F}$ from G' leaves a forest, then the induced subgraph $G'' = (S\setminus\tilde{F}, E'')$ may have at most $|S| - |\tilde{F}| - 1$ edges, i.e., $|E''| \leqslant |S| - |\tilde{F}| - 1$. Then

$$|E(S)| \leqslant |S| - |\tilde{F}| - 1 + \sum_{v\in\tilde{F}} d(v) - |E(\tilde{F})| \tag{5}$$

and $\sum_{v\in\tilde{F}} d(v) - |\tilde{F}| = \sum_{v\in\tilde{F}}(d(v)-1)$ which leads to

$$\sum_{v\in\tilde{F}}(d(v)-1)x_v \geqslant |E(S)| - |S| + 1 + |E(\tilde{F})| \geqslant |E(S)| - |S| + 1. \quad \square \tag{6}$$

Corollary 6. *If $S\subseteq V$ is a cycle that has no chords, then* (4) *is facet defining.*

Proof. Trivially, inequality (4) reduces to the cycle inequality and the proof is that of Theorem 4. □

Remark 7. If $G=(S,E(S))$ is a complete graph (a clique), the subset inequalities (4) are not facet defining. In fact, let $G=(S,E(S))$ be a complete graph of four vertices, $S=\{a,b,c,d\}$, $|S|=4$ and $|E(S)|=6$, the corresponding subset inequality reads $2x_a+2x_b+2x_c+2x_d\geqslant 3$ that is satisfied at equality only by fractional points.

However, if $S=K_n$ is the vertex set of a clique of n vertices then any feasible UFVS $\tilde{F}$ has cardinality $|\tilde{F}|\geqslant n-2$; in fact, if $|\tilde{F}|\leqslant n-3$ it is always possible to find at least a cycle of 3 vertices not covered by any vertex. Therefore the following inequalities

$$\sum_{v\in K_n} x_v \geqslant n-2 \tag{7}$$

are trivially valid. Inequalities (7) are the *clique inequalities*. In the following proposition, we do not only show that it is possible to derive this family of inequalities from the subset inequalities, but that they represent a strengthening of them, since we are able to determine the exact value of $|E(\tilde{F})|$.

Proposition 8. *Inequality* (7) *is a strengthened subset inequality.*

Proof. By (6) we know that $\sum_{v\in\tilde{F}}(d_S(v)-1)x_v \geqslant |E|-|S|+1+|E(\tilde{F})|$ holds. Since $S=K_n$ and $|K_n|=n$, then $d_{K_n}(v)=n-1$. We observe that if $\tilde{F}$ is a minimal UFVS in K_n, it is a complete graph of $n-2$ vertices, i.e., $|\tilde{F}|=n-2$, and $|E(\tilde{F})|=(n-3)(n-2)/2$. Substituting in (6) we obtain $\sum_{v\in\tilde{F}}(n-2)x_v \geqslant 2(n-2)(n-2)/2$ and the result follows. □

Theorem 9. *If $K_n\subseteq V$ $(n\geqslant 3)$, then inequality* (7) *is facet defining.*

Proof. By induction. Trivial for $n=3$, since it reduces to a cycle. Suppose it holds when K_{n-1}. Let v be the vertex of $V\backslash K_{n-1}$ and $\mathscr{F}(K_{n-1})$ a set of $n-1$ independent UFVSs of K_{n-1}, we can construct $n-1$ points considering $\forall F\in\mathscr{F}(K_{n-1})$, $\chi^{F\cup v}$. The other point can be obtained by considering $\chi^{K_{n-1}\backslash\{u\}}$, where u is any vertex of K_{n-1}. □

Remark 10. We may observe that if G is a complete graph finding the minimal undirected feedback vertex set becomes an easy problem. UFVS can be obtained from V by sorting its vertices in non-decreasing order of their weights and considering the first $n-2$ vertices.

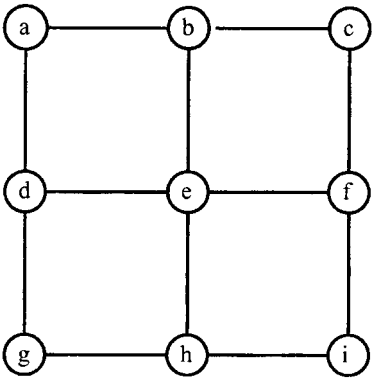

Fig. 1. A 3×3 rectangular planar grid.

We prove with Theorems 6 and 9 that the subset inequalities (4) are a large class of inequalities that comprise the cycle inequalities (2) and the clique inequalities (7). However, we consider the cycle and the clique inequalities as distinct from the subset inequalities since the subset inequalities are separated using a heuristic procedure and the cycle and clique inequalities can be separated using an exact procedure (see Section 3).

The following result, that will not be used for producing the LP relaxation of the problem, is reported to provide an example in which inequality (4) is facet defining, even if S is not a cycle or a clique.

Proposition 11. *If $S \subseteq V$ is a 3×3 rectangular planar grid, inequality* (4) *is facet defining.*

Proof. Suppose that the vertex set is $S = \{a, b, c, d, e, f, g, h, i\}$ as in Fig. 1, where $|S| = 9$ and $|E(S)| = 12$. The corresponding subset inequality reads $x_a + 2x_b + x_c + 2x_d + 3x_e + 2x_f + x_g + 2x_h + x_i \geqslant 4$. We can construct 9 affinely independent points that are UFVS as follows. The first four points contain the only vertex of degree 4: $\chi^{\{a,e\}}$, $\chi^{\{c,e\}}$, $\chi^{\{g,e\}}$, $\chi^{\{i,e\}}$. Two points contain only vertices of degree 3, i.e., $\chi^{\{b,h\}}$ and $\chi^{\{d,f\}}$. The last three points contains also vertices of degree 2: $\chi^{\{b,g,i\}}$, $\chi^{\{a,c,h\}}$ and $\chi^{\{a,c,g,i\}}$. □

We decided not to include in the description of our polytope the many inequalities that have been proposed for the set covering polytope in the previous literature. In fact, on one hand, the majority of valid inequalities need to be lifted using exact procedures to be computationally effective: these lifting procedures are very expensive in terms of CPU time. On the other hand, approximate lifting procedures often lead to inequalities that are not effective, thus increasing the density of the constraint matrix and negatively affecting the LP computation time [4].

3. The branch-and-cut algorithm

An inequality $\alpha x \leqslant \alpha_0$ is *violated* by the current LP solution, say $\tilde{x}$, if we have $\alpha\tilde{x} > \alpha_0$. The addition of violated inequalities belonging to a specific class is obtained by solving the separation problem discussed in this section.

The constraint generator is the most important part of our branch-and-cut algorithm. The inequalities produced by the generator fall into one of the following three categories: (a) clique inequalities, (b) cycle inequalities and (c) subset inequalities.

As a heuristic rule, we skip the violated cuts with degree of violation less than 0.01 (for the clique inequalities and cycle inequalities) or 0.1 (for the subset inequalities).

The input to the cut generator is the optimal solution $\tilde{x}$ of the current LP relaxation.

3.1. Separation of clique inequalities

The separation of clique inequalities is exact and it is performed at the beginning of the optimization process. All the maximal cliques are identified using the algorithm described in [15]. As the number of clique inequalities is reasonably small we usually add them all to the first LP.

3.2. Separation of cycle inequalities

The separation of cycle inequalities is exact and it is reduced to the shortest path problem. To reduce the shortest path problem with cost on vertices to the usual one with costs on edges, we use the transformation $w_{uv} = (\tilde{x}_u + \tilde{x}_v)/2$. The cost of a cycle C on edges is then $w(E(C))$ which is equal to $x(C)$. Since we are looking for violated inequalities, we restrict our attention to finding shortest paths among pairs of vertices u and v with $\tilde{x}_u + \tilde{x}_v < 1$. If a violated inequality is detected, we first check in the pool if it has been already separated before saving it.

Algorithm 1. Cycle separation algorithm

Initialize: **for** all $uv \in E$ **do** set $w_{uv} = (\tilde{x}_u + \tilde{x}_v)/2$.
for all pairs of vertices u and v with $\tilde{x}_u + \tilde{x}_v < 1$ **do**
  Enumerate all paths P_{uv} in G that have u and v as origin and end;
  for each path P_{uv} in G **do**
    set $w(E(C)) = w(P_{uv}) + w_{uv}$
    if $w(E(C)) < 1$ and constraint $x(C) \geqslant 1$ has not been already separated,
    then save corresponding constraint;

In our implementation we used the Ye shortest path algorithm to enumerate all the possible paths. We generate at most 100 violated cycle inequalities.

3.3. Separation of subset inequalities

When the constraint generator is no longer able to identify violated cycle inequalities, it searches for subset inequalities (4) of any size. As far as we know, the complexity of the separation of subset inequalities is not known. Being computationally too heavy to check all inequalities (4), since there is an exponential number of subsets S of V, we have to adopt an heuristic procedure: we first check all the vertex sets obtained from V by deleting one vertex, than those obtained deleting two vertices, then three vertices, and so forth until we generate at most 100 violated subset inequalities, or when $|V|^2$ vertex sets have been tested without finding a violated inequality. From our computational experience, the latter case never happens in practice.

Algorithm 2. Subset separation algorithm

for all subsets $S \subseteq V$ **do**
  if $\sum_{v \in S} (d_S(v) - 1)\tilde{x}_v < |E(S)| - |S| + 1$,
  then save the corresponding constraint;

From our computational experiences the subset inequalities are less effective than the cycle inequalities, they increase the density of the constraint matrix and decrease the efficiency of the LP solver, but they often lead to an LP relaxation that is tighter than that provided using only (2) and (3). This can be seen by looking at the third and fourth columns of Table 1 in Section 5.

3.4. Variable fixing and setting

Fixing variables reduces the number of variables to be handled and tightens the LP formulation of all the subproblems of the branch-and-cut algorithm. It is well known that a variable fixing criterion can be obtained by considering the value of the reduced costs associated with an optimal basic solution of an LP relaxation. These fixed variables are constrained to be equal to 0 and 1, respectively. Since these constraints are only "locally" valid, i.e., are valid only for the current subproblem and its "descendants" in the branch-and-cut tree, we say that these variables are *set* to their corresponding values. These settings are valid, of course, only for the subproblem and for its descendants.

Once some variables have been set, either when a subproblem is solved for the first time or when the reduced cost criterion has been applied, the following observations are used to construct a simple algorithm to possibly set more variables: if a vertex $v \in V$ has degree lower than 2 it does not belong to any cycle; if two vertices are adjacent in a 2-connected graph, they belong to, at least, a common cycle.

We consider the original graph $G = (V, E)$ and the variable $\delta(v)$ equal to the vertex degree, i.e., $\delta(v) = d(v)$, for all $v \in V$. Then this step can be applied to each vertex

Table 1
Results on a small subset of instances

n	m	CRT	SRT	HB	OPT	CYC	SUB	CLI	MR	TBC	TLS	BC
20	56	6	6	7	7	101	52	0	146	1	0	5
20	59	7	7	8	8	108	67	0	152	1	0	7
40	75	7	8	9	9	210	132	0	180	4	1	27
40	84	8	9	10	10	200	147	0	199	2	1	9
40	93	9	11	12	12	297	159	0	302	5	2	19
40	98	9	11	12	12	345	177	0	327	5	2	17
40	157	13	15	18	18	424	3967	0	402	55.71	2.27	647
60g	104	46	48	50	50	135	53	0	141	5	5	13
60	524	1447	1447	1917	1917	0	16 026	119	1397	1566	12	6101
60	512	1309	1309	1747	1747	0	10 424	108	1241	776	11	3473
60	527	1433	1433	1979	1979	0	22 133	100	1716	2182	11	8099
60	545	1257	1271	1891	1891	206	15 464	73	2239	1290	11	5269
60	520	1242	1243	1803	1803	44	34 903	82	2927	5008	11	17 785
60	558	1044	1044	1460	1460	0	18 774	145	1540	1669	12	6931
60	537	1323	1324	1849	1849	100	17 913	81	1730	1581	11	6941
60	504	1366	1369	1975	1975	16	26 642	80	2751	3692	11	14 227
60	699	182	182	245	245	0	48 671	195	3055	7664	13	26 273
70	1950	519	521	897	897	112	26 376	200	3077	1595	34	3243
70	2178	432	433	670	670	45	18 008	0	3242	836	30	2707
80p	225	518	518	557	556	102	223	0	253	26	14	71
80	2527	801	801	1224	1224	0	52 121	201	2334	4878	63	8495
100g	180	65	65	70	70	83	168	0	175	45	29	147
100	2502	272	272	464	464	2	45 807	200	3297	22 593	84	34 749
120p	346	494	494	512	512	315	0	0	223	36	55	9
160g	292	79	79	82	82	113	0	0	102	80	138	9
160p	292	79	79	82	82	0	113	0	102	80	138	9
200p	585	835	835	891	891	667	0	0	415	608	305	887
200g	376	116	116	119	119	238	0	0	154	247	311	77
300p	879	446	446	478	478	1381	0	0	766	7019	1252	5911

$v \in V$: if v is adjacent to a vertex u corresponding to a variables set to 1, the cycle common to the two vertices is already covered by vertex u, and $\delta(v)$ can be reduced by 1. At the end of the process, any variable corresponding to a vertex $w \in V$ having $\delta(w)$ lower than 2 can be set to 0. In fact, $\delta(w) < 2$ means that w belongs to cycles already covered by vertices corresponding to variables set to 1, and w will certainly not be in the optimal solution.

3.5. Node and variable selection strategies

Two elements that are critical for the efficiency of a branch-and-cut algorithm are the criterion used to select the next unsolved subproblem and the criterion used to select the branching variable. Due to the good quality of the solutions produced by the heuristic algorithm, we decided to select the simplest among the possible known criteria. Therefore, we visit the branch-and-cut tree in a "depth first" manner and we pick the variable with highest vertex degree, among those of value closest to 0.5, as the branching variable. It may be worth reporting that, from our computational experience,

we reduced by a half the number of vertices in the branch-and-cut tree by picking the variable with highest vertex degree together with variable fixing and setting.

Furthermore, a heuristic algorithm is applied at the beginning of the root vertex of the enumeration tree to find an initial "good" feasible solution to Problem (2)–(4). This algorithm is based on the local search approach described in the next section.

4. The local search algorithm

The local search algorithm described in this section is a simplified version of the exploring tabu search ($\mathscr{X}-\mathscr{TS}$) presented in [9] which has been already applied with success to other combinatorial problems such as the equicut (see [10]). In this section we describe the procedure adopted to build feasible solutions, the neighborhood function and the basic elements of the tabu search algorithm.

4.1. Generating feasible solutions

We generate feasible solutions using the 2-approximated modified greedy algorithm (MGA) due to Becker and Geiger [3].

Given a subgraph $G'=(S,E')$ of a graph G, we define two operations on the vertices of G': vertex *removal* and vertex *insertion*. We say that we *remove* a vertex v from G' when we eliminate v from S and all the edges of the star $\delta(v)$ from E'. On the contrary, we *insert* a vertex v into a subgraph of G' when we add a vertex $v \in V\backslash S$ to S and all the edges $vu \in E$ having as end-points v and a vertex u already in S.

Algorithm 3. Procedure MGA (input G, output F)

Step 1. $G_c = G$; $F = \emptyset$.
Step 2. Remove from G_c all the vertices u with degree lower than two (since they cannot cover any cycle); **if** $G_c = \emptyset$, **then** goto 4.
Step 3. Choose in G_c the vertex v of minimum ratio value r ($r = c_v/d(v)$): remove v from G_c and put v in F; goto 2.
Step 4. Re-insert in G_c all the vertices removed in step 2.
Step 5. **Repeat** for all vertices v of F, selected in order reverse of that of their inclusion in F: **if** $\{v\} \cup G_c$ does not contain any cycle **then** remove v from F and insert v in G_c.

Let us observe that steps 4 and 5 are required since F could be non-minimal at the iteration in which G_c becomes empty.

In step 3, for any vertex v, $d(v)$ is used as an estimate of the number of cycles which traverse v. Since $d(v)$ can both overestimate or underestimate that value, we have compared the performances of the algorithm MGA with a modified version of it, which we call algorithm G1. In algorithm G1 we substitute in step 1 the value $d(v)$ with a lower bound $l(v)$ on the number of cycles traversing v. The value $l(v)$ is

computed by adding a 1 for any pair of vertices u and w of $\delta(v)$ connected by a path in G_c not traversing v, i.e., if there exists a cycle through u, v, w.

On a set of 1200 randomly generated instances MGA dominated G1 in the 75% of the cases. As a consequence, since the former algorithm is even faster than G1 we adopted MGA in our code.

4.2. The neighborhood

Let F be a UFVS ($F \subset V$) in a graph $G = (V, E)$ and let $\bar{F}$ be its complement in G (a forest resulting from the removal of the vertices in F).

Definition 12. Let v_i be a generic vertex in F: a subset $A_i \subseteq \bar{F}$ is an *exchange set* of v_i if the subgraph $\bar{F} \cup \{v_i\} \backslash A_i$ is a forest. Hence, $F \backslash \{v_i\} \cup A_i$ is a UFVS for graph G.

Given a UFVS F let us denote with $A(v_i)$, for any $v_i \in F$, the set of all exchange sets: $A(v_i) = \{A_i \subseteq \bar{F} : \bar{F} \cup \{v_i\} \backslash A_i$ is a forest$\}$. Given a feasible solution S, represented by a UFVS F, the solutions in $N(S)$ are generated *moving* from F to $F \backslash \{v_i\} \cup A_i$ in all the possible ways. Formally, our neighborhood is defined as follows:

$$N(S) = \{F \backslash \{v_i\} \cup A\colon\ v_i \in F,\ A \in A(v_i)\}.$$

Let us observe that the size of $N(S)$ is exponential in the size of $\bar{F}$ and that looking for the *exchange set* of v_i of minimum cost in $A(v_i)$ can be formulated either as a *set covering* problem or as a special feedback vertex set problem on a restricted graph. Since no polynomial algorithm is known for solving this particular problem, whose complexity is open, we look for a good solution in $N(S)$ by exploring $A(v_i)$ with the following three step heuristic procedure.

Algorithm 4. UFVS heuristic procedure

Step 1. Reduce the size of graph $\bar{F} \cup \{v_i\}$ by recursively removing all the vertices u with degree less than two; let F_c be the resulting graph.

Step 2. Solve the problem of finding the minimum cost vertex set, A_i, between all the vertex sets not containing vertex v_i on F_c; set $F_i = F \backslash \{v_i\} \cup A_i$.

Step 3. Since F_i is not guaranteed to be minimal, check for each vertex $v_f \in F_i$ if it belongs to a cycle in $\bar{F} \cup v_f$: if it does v_f is kept in the vertex set F_i, otherwise v_f is moved from F_i to $\bar{F}$.

We have compared the choice of solving the problem formulated in the second step, either with a modified version of algorithm MGA (we have to avoid to insert v_i in A_i), or with the Chvátal heuristic, see [6], for the set covering problem. To formulate our problem as a set covering problem we generated a covering matrix M in the following way: if for any two vertices u and w in $\delta(v_i) \cap F_c$ there exists a path in F_c there is

a row in M that corresponds to the characteristic vector $\chi^{P_{uw}(F_c)}$ of the vertices in the (unique) path.

We compared the two procedures within a simple tabu search framework on a test set of 480 instances, with up to 100 vertices. The two implementations have given the same results in 89.8% of the cases. In 6% of the case the MGA version has found better solutions than those obtained with the set covering heuristic version, while in the remaining 4.2% of the cases we obtained the opposite ranking. Since the MGA heuristic appeared to be faster than the set covering heuristic, we adopted the former one.

4.3. $\mathscr{X} - \mathscr{TS}$

In this section we consider the reader to be familiar with the main components of a tabu search algorithm, see e.g. [12].

Since in each move we exchange two subsets of vertices, we associated with each visited solution two *attributes*: the set $S_1 = \{v_i\} \cup R_i$ of vertices moved from F and the set $S_2 = A_i$ of vertices moved from $\bar{F}$, where R_i denotes the set of vertices removed from F_i in step 3 of the procedure which computes the neighborhood.

In practice, we consider two lists: for each vertex v, *out_fvs*(v) contains the last iteration in which v has been inserted in S_1 and *out_forest*(v) contains the last iteration in which v has been inserted in S_2. In those iterations in which $|F| > |\bar{F}|$ we forbid the moves which try to insert in $\bar{F}$ a set $S_1 \subseteq F$ whose elements were inserted in the feedback vertex set too recently, i.e., if *out_forest*$(v)+tt \geqslant it$ $\forall v \in S_1$, where it is the number of the current iteration and tt is the *tabu tenure* of a move. In those iterations in which $|F| \leqslant |\bar{F}|$ we forbid the moves which try to insert in F a set $S_2 \subseteq \bar{F}$ whose elements were inserted in the forest too recently, i.e., if *out_fvs*$(v) + tt \geqslant it$ $\forall v \in S_2$.

A *first tool* of the $\mathscr{X} - \mathscr{TS}$ approach is a dynamic updating of the tabu tenure. The value of tt is initialized to a given value *start_tt* and is modified according to the evolution of the search. By allowing tt to vary only within an interval $[l_m, l_M]$, we decrease tt by a unit, if in the last iteration we improved the objective function value and we increase tt by a unit, otherwise. We set l_m to 3 and we set l_M to $\max\{|F|/2, |\bar{F}|/2\}$. Indeed, if $tt > \max\{|F|, |\bar{F}|\}$ in a few iterations, no moves are allowed. The value *start_tt* is initialized to the half of the interval $[l_m, l_M]$.

A *second tool* of the $\mathscr{X} - \mathscr{TS}$ approach is a long-term memory based on the memorization of "good" solutions. It is used for intensifying the search into regions analyzed, but not completely explored, and for diversifying the search from the current local optimum. We store in a fixed length list, called *Second*, the L best solutions whose objective function value was the second best value in all the visited neighborhoods. If one of the following three conditions holds we remove from *Second* the best solution s (and all the relevant information present when s was added to *Second*) and we continue the search from s. The conditions are: (a) all the solutions in the current neighborhood are forbidden; (b) the current objective function value has not been improved in the last MC iterations; (c) the best objective function value has not been improved in

the last *MB* iterations. With preliminary computational experiments we choose to set $L = 5$, $MC = 15$ and $MB = |N|$.

A *third tool* of the $\mathscr{X} - \mathscr{TS}$ approach is a global restarting technique. This is a brute force way to implement a diversification technique. If it is necessary to use a solution from *Second*, but either the list is empty or it has been already used L times from the last restart, we continue the search from a new generated solution. We need to generate a set of random starting solutions to apply this third tool: we randomized the greedy algorithm by substituting step 3 of MGA with the following one:

Step 3′. *Sort the vertices in* G_c *in non decreasing order of the ratio* $r = c_v/d(v)$; *choose randomly a vertex* v *among the first three vertices in the given order*; *put* v *in* F *and remove* v *from* G_c; *goto* 2.

Let us observe that in step 3′, there always exist at least three vertices, since G_c contains at least one cycle.

On a set of 240 randomly generated instances, with 100 vertices, we have compared the performances of $\mathscr{X} - \mathscr{TS}$ against three algorithms: TS, which adopts only the first tool, TSR, which adopts the first and the third tools and MS which adopts the same randomization technique of $\mathscr{X} - \mathscr{TS}$ to generate different starting solutions, but which does not implement any memory structure. We gave to all the algorithms the same amount of time. $\mathscr{X} - \mathscr{TS}$ has found solutions that were better than those obtained with TS in more than 12% of the cases, that were equal in the remaining cases, but never worse. $\mathscr{X} - \mathscr{TS}$ has found better solutions than those obtained with MS in about 6% of the cases, but it was beaten once: the results were the same in all the remaining cases. $\mathscr{X} - \mathscr{TS}$ and TSR did not obtain the same results in three cases only.

Since TSR does not require the overhead of managing the list *Second* and it gives the same performances of $\mathscr{X} - \mathscr{TS}$ we preferred this simplified version of $\mathscr{X} - \mathscr{TS}$ as the local search-based algorithm for the UFVS problem.

5. Experimental results

The algorithm described in the previous sections was implemented in C and tested on randomly generated instances. We used the CPLEX CALLABLE LIBRARY, VERSION 3.0. For the computational experiments we used an Sun Sparc 20/71. Since we could not find small or large hard instances available in the literature, we created a library of 1400 instances, divided in four sets: *random*, *geometric*, *planar*, and *planar grid* graphs.

The vertex weights of all instances are of three types: randomly generated integer weights in a range between 1 and 10, between 1 and 100, or all equal to 1 (UFVS of minimum cardinality).

We have tested our algorithm on a library constituted of 600 instances on *random graphs*, and of 600 on *geometric graphs*. These instances had the following structure: *random graphs* $G_{n,p}$, where n is the number of vertices and $0 < p < 1$ is the probability that the edge between a given pair of vertices exists; *geometric graphs* $U_{n,d}$, generated drawing from an uniform distribution n points in an unit square, associating a vertex

to each point and adding edge $[u, v]$ to the graph if and only if the Euclidean distance between the u and v is less or equal to d. The expected average vertex degree $\hat{v}$ is equal to $p(n-1)$ for the random graphs, whereas it is approximately $n\pi d^2$ for the geometric graphs. The instances considered have $n = 20, 40, 60, 80, 100$; d and p are such that $\hat{v} \in [10, 90]$. As we already observed in Remark 10 in Section 2, the UFVS in a complete graph is a trivial problem, so we decided not to provide results on this kind of instances. As one could expect the local search-based algorithm produces always better solutions than our implementation of the MGA greedy approximation algorithm by Becker and Geiger [3]: from 3% on the 20 vertices graphs to 6% on the 100 vertices graphs. We have solved at optimality all the instances up to 60 vertices, three 80 vertex instances and one 100 vertex instance: the heuristic solution value very often coincides with the optimal solution.

Because of the relevance of the problem in the study of "monopolies" in synchronous distributed systems, as introduced in [19,20], we created also a library of 100 *planar grid graph* and 100 *planar graph* instances. *Planar grid graphs* $G_{h\times k}$ are $h \times k$ rectangular grid graphs of hk vertices and $2hk - h - k$ edges; the number of vertices ranges from 40 to 200. The other are *planar graphs* where the number of vertices is from 40 to 300. The size of an instance in Table 1 is ended with a 'g' for the planar grid instances and with a 'p' for the planar. We have solved all the instances (up to 300 vertices) in very low CPU time and, again, the heuristic solution value very often coincides with the optimal solution. Therefore, we are confident that instances arising in practice will be successfully solved by our system.

In the following Table 1 we give the results of our experiments using the following abbreviations:

- n, m: number of vertices and number of edges;
- CRT: best objective function value found at the root node with clique (7) and cycle inequalities (3) only;
- SRT: best objective function value found at the root node using also the subset inequalities (4);
- HB, OPT: objective function value of the best feasible solution found by the heuristic and the optimal one;
- CYC, CLI, SUB: total number of cycle, clique and subset inequalities;
- MR: maximum number of rows in the matrix;
- TBC, TLS: CPU time (in s) spent by the branch and cut algorithm and by the local search algorithm;
- BC: total number of nodes of the search tree.

Acknowledgements

We are particularly grateful to Alessandro Termignone who has implemented the C code of the local search algorithms. We would also like to thank Giovanni Righini for providing us an implementation of the algorithm described in [15]. The research

of the second author was partially supported by a NATO research grant (contract NATO CRG 971550). The research of the other authors was partially supported by a MURST research grant. We also thank the anonymous Referees for their comments and suggestions that improved the quality of the paper.

References

[1] V. Bafna, P. Berman, T. Fujito, Constant ratio approximations of the feedback vertex set problem for undirected graphs, in: J. Staples P. Eades, N. Katoh, A. Moffat (Eds.), ISAC 95, Algorithms and Computation, 1995, pp. 142–151.
[2] R. Bar-Yehuda, D. Geiger, J. Naor, R.M. Roth, Approximation algorithms for the feedback vertex set problem with applications to constraint satisfaction and Bayesian inference, Proceedings of the 5th Annual ACM SIAM Symposium on Discrete Algorithms, 1994, pp. 344–354.
[3] A. Becker, D. Geiger, Approximation algorithms for the loop cutset problem, in: Proceedings of the 10th Conference on Uncertainty in Artificial Intelligence, 1994, pp. 60–68.
[4] S. Ceria, P. Nobili, A. Sassano, Set covering problem, in: M. Dell'Amico, S. Martello, F. Maffioli (Eds.), Annotated Bibliographies in Combinatorial Optimization, Wiley, Chichester, 1997, pp. 415–428.
[5] F.A. Chudak, M.X. Goemans, D.S. Hochbaum, D.P. Williamson, A primal–dual interpretation of two 2-approximation algorithms for the feedback vertex set problem in undirected graphs, Oper. Res. Lett. 22 (1998) 111–118.
[6] V. Chvátal, A greedy heuristic for the set covering, Math. Oper. Res. 7 (1979) 515–531.
[7] S.R. Coorg, C.P. Rangan, Feedback vertex set on cocomparability graphs, Networks 26 (1995) 101–111.
[8] R. Dechter, Enhancement schemes for constraint processing: backjumping, learning and cutset decomposition, Artificial Intell. 41 (1990) 273–312.
[9] M. Dell'Amico, A. Lodi, F. Maffioli, Solution of the cumulative assignment problem with a new tabu search method, J. Heuristics (1997), to appear.
[10] M. Dell'Amico, M. Trubian, Solution of large weighted equicut problems, European J. Oper. Res. 106 (1998) 500–521.
[11] M. Funke, G. Reinelt, A polyhedral approach to the feedback vertex set problem, in: Proc. 5th Int. IPCO Conf., Vancouver, 1996, pp. 445–459.
[12] F. Glover, Tabu search — Part 1, ORSA J. Comput. 1 (1989) 190–206.
[13] D.S. Hochbaum, Various notions of approximations: good, better, best, and more, in: D.S. Hochbaum (Ed.), Approximation Algorithms for NP-hard Problems PWS, Boston, 1996, pp. 347–363.
[14] R.M. Karp, Reducibility among combinatorial problems, in: R. Miller, J. Thatcher (Eds.), Complexity of Computer Computations, Plenum Press, New York, 1972, pp. 85–103.
[15] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, Generating all maximal independent sets: NP-hardness and polynomial time algorithms, SIAM J. Comput. 9/3 (1980) 559–565.
[16] E.L. Llyod, M.L. Soffa, On locating minimum feedback vertex sets, J. Comput. Systems Sci. 37 (1988) 292–311.
[17] F.L. Luccio, Almost exact minimum feedback vertex set in meshes and butterflies, Inform. Process. Lett. 66 (1998) 59–64.
[18] C. Lu, C. Tang, A linear-time algorithm for the weighted feedback vertex problem on interval graphs, Inform. Process. Lett. 61 (1997) 107–112.
[19] D. Peleg, Local majority voting, small coalitions and controlling monopolies in graphs: a review, in: Proceeding of 3rd Colloqium on Structural Information and Communications Complexity, 1996, pp. 152–169.
[20] D. Peleg, Size bounds for dynamic monopolies, Proceeding of 4th Colloqium on Structural Information and Communications Complexity, 1997, pp. 165–175.
[21] C. Wang, E. Lloyd, M. Soffa, Feedback vertex set and cyclically reducible graphs, J. ACM 32 (1985) 296–313.