アルゴリズム 1－7
(1988. 5. 23)

強ユニモジュラ行列による集合パッキング、集合カバーおよび集合分割問題について

Y. Crama (University of Dalaware)
P. L. Hammer (Rutgers University)
茨木　俊秀（京都大学）

0-1行列Aは、その任意の正則部分行列が（行と列を適当に置換すれば）3角行列であるとき、強ユニモジュラであるという。本報告では、強ユニモジュラ行列Aを係数とする0-1計画問題である集合パッキング、集合カバーおよび集合分割問題に対し、多項式オーダーの効率よいアルゴリズムを与える。このアルゴリズムは、強ユニモジュラ行列の制限ユニモジュラ行列への分解定理と、制限ユニモジュラ行列を係数とする0-1計画問題に対するYannakakisのアルゴリズムを利用するものである。

PACKING, COVERING AND PARTITIONING PROBLEMS WITH STRONGLY UNIMODULAR CONSTRAINT MATRICES

Yves CRAMA, *Department of Mathematical Sciences, University of Delaware, Newark, DE 19716, USA*
Peter L. HAMMER, *RUTCOR–Rutgers Center for Operations Research, Rutgers University, New Brunswick, NJ 08903, USA*
Toshihide IBARAKI, *Department of Applied Mathematics and Physics, Faculty of Engineering, Kyoto University, Kyoto, Japan*

A 0-1 matrix A is called strongly unimodular if its nonsingular square submatrices are all triangular. We present an efficient algorithm for linear programming problems in binary variables, when all constraints are of the packing, covering, or partitioning type, and the constraint matrix is strongly unimodular. The algorithm uses a certain decomposition of strongly unimodular matrices into their so-called "restricted unimodular" components, and an efficient optimization algorithm for linear programs with restricted unimodular constraints.

1 INTRODUCTION

We consider the following optimization problem P :

$$\max \quad wx$$

$$\text{s.t.} \quad A_1x \leq e \tag{1}$$

$$A_2x \geq e \tag{2}$$

$$A_3x = e \tag{3}$$

$$x \in \{0,1\}^n,$$

where A_1, A_2, A_3 are 0-1 matrices, $w \in R^n$, and e denotes a vector of all 1's of appropriate dimension. Constraints of type (1), (2) and (3) are called *packing*, *covering* and *partitioning* constraints, respectively.

Packing, covering and partitioning problems arise naturally in combinatorial optimization theory, as well as in a multitude of real-world applications. "Mixed" formulations of the type described above also allow to take into account a large variety of nonlinear 0-1 optimization problems. To see this, consider the problem :

$$\max \quad F_1(x) = \sum_{i \in X} a_i x_i + \sum_{k \in P} b_k T_k + \sum_{k \in N} c_k T_k \tag{4}$$

$$\text{s.t.} \quad x_i \in \{0,1\} \text{ for all } i \in X = \{1,2,...,n\}, \tag{5}$$

where :

$b_k > 0$ for all $k \in P$;

$c_k < 0$ for all $k \in N$;

$T_k = \prod_{i \in S(k)} x_i$ and $S(k) \subseteq X$ for all $k \in P \cup N$.

The objective function (4) can be linearized as follows. We introduce new 0-1 variables y_k associated with the nonlinear terms T_k ($k \in P \cup N$), and constrain them to take the value of the product of the variables in these terms. We then arrive at the following linear 0-1 program :

$$\max \quad F_2(x,y) = \sum_{i \in X} a_i x_i + \sum_{k \in P} b_k y_k + \sum_{k \in N} c_k y_k \tag{6}$$

$$\text{s.t.} \quad y_k - x_i \leq 0 \text{ for all } i \in S(k), \; k \in P \tag{7}$$

$$-y_k + \sum_{i \in S(k)} x_i \leq |\, S(k) \,| -1 \text{ for all } k \in N \tag{8}$$

$$x_i \in \{0,1\} \text{ for all } i \in X \tag{9}$$

$$y_i \in \{0,1\} \text{ for all } k \in P \cup N. \tag{10}$$

If we let now $\overline{x}_i = 1 - x_i$ ($i = 1,...,n$) everywhere in (6)–(10), we obtain the program :

$$\max \quad F_3(\overline{x},y) = -\sum_{i \in X} a_i \overline{x}_i + \sum_{k \in P} b_k y_k + \sum_{k \in N} c_k y_k$$

$$\text{s.t.} \quad y_k + \overline{x}_i \leq 1 \text{ for all } i \in S(k), \; k \in P$$

$$y_k + \sum_{i \in S(k)} \overline{x}_i \geq 1 \text{ for all } k \in N$$

$$\overline{x}_i \in \{0,1\} \text{ for all } i \in X$$

$$y_k \in \{0,1\} \text{ for all } k \in P \cup N.$$

This last formulation is of type P.

We call 0-1 matrix *strongly unimodular* (SU) if all its nonsingular square submatrices are triangular, up to a permutation of their rows and columns (this is the "Dantzig property" of Balas and Padberg [1]). We gave in a previous paper several alternative characterizations of SU matrices, notably for constraint matrices arising as explained above from nonlinear optimization problems ([3]; see also [2]). The purpose of the present note is to complement these results by giving an efficient optimization algorithm for problem P when the constraints (1), (2), (3) define an SU matrix.

The paper is organized as follows. We survey in Section 2 some results of Yannakakis [5] and Conforti and Rao [2]. In Section 3, we present a "reduction" procedure, which allows in certain situations to transform problem P into an equivalent problem of smaller size. Finally, we show in the last Section how this reduction step can be combined with the results presented in Section 2 in order to solve completely problem P, under the strong unimodularity assumption.

2 SURVEY OF PREVIOUS RESULTS

We start by introducing some useful graph-theoretical interpretation of the concepts discussed in the Introduction. Our terminology is standard, and follows for instance that of [5]. Since all graphs considered in this paper are bipartite, we shall often drop the latter qualifier without any risk of confusion. We denote by $V(G)$ (resp. $E(G)$) the vertex-set (resp. edge-set) of a graph G. If $v \in V(G)$, then $N(v)$ is the neighborhood of v (i.e., the set of vertices adjacent to v, not including v), and $G\backslash v$ is the subgraph of G induced by $V(G)\backslash v$. If R and C are disjoint, we denote by $K(R,C)$ the complete bipartite graph on $R \cup C$. A star with center v is a graph $K(R,C)$ for which $R = \{v\}$ or $C = \{v\}$.

Let A be an arbitrary 0-1 matrix. The *graph of* A is the bipartite graph $G(A)$ with *adjacency matrix* A : so, we think of $G(A)$ as having *row-* and *column-vertices*, and we put an edge between row-vertex i and column-vertex j if $a_{ij} = 1$. If A is the constraint matrix of problem P, then each row-vertex of $G(A)$ is either of the packing, covering or partitioning type, and each column-vertex has some weight w_i attached to it. Every feasible solution of P is the incidence vector of some subset of column-vertices of G. Conversely, we can completely define a problem of type P by simply giving a bipartite graph G on the vertex-set $R \cup C$, together with a partition of R into packing, covering and partitioning vertices, and a weighting w of the vertices in C. This will be called, for short, the *problem associated with* G.

A bipartite graph is called *restricted unimodular* (RU) if all its cycles have length 0 modulo 4. A 0-1 matrix is RU if its graph is RU. In [5], Yannakakis presents an efficient procedure to solve problem P when the constraint matrix defined by (1), (2), (3) is RU (actually, he even allows constraints with arbitrary right-hand sides). This result relies on the proof that a matrix is RU if and only if it can be constructed from the incidence matrices of bipartite graphs and their transposes, glued together by some simple operations. Call a function $f(n,m)$ *subadditive* if $f(n,m) + f(k,l) \leq f(n+k, m+l)$, for all n, m, k, l. Then, Yannakakis proves :

Proposition 1 [5] *Suppose that the b-matching problem and the maximum weight independent set problem can be solved in time $f(n,m)$ and $g(n,m)$, respectively, on a bipartite graph with n vertices and m edges, where f and g are subadditive functions. Then, P can be solved for an $m \times n$ RU constraint matrix in time $O(f(m,n) + g(n,m))$.*

A bipartite graph is called *strongly unimodular* (SU) if its adjacency matrix is SU. From the results in [3] and [2], it follows that a graph is SU if and only if all its cycles of length 2 modulo 4 have at least two chords. In particular, every RU graph is SU.

Consider now two bipartite graphs G_1, G_2 on disjoint vertex-sets. Let $v_1 \in V(G_1)$, $v_2 \in V(G_2)$ be two distinguished vertices such that $\mid N(v_1) \mid \geq 2$, $\mid N(v_2) \mid \geq 2$. The *composition* of G_1 and G_2 along v_1, v_2 is by definition the graph G with vertex-set :

$$V(G) = (V(G_1)\backslash v_1) \cup (V(G_2)\backslash v_2)$$

and with edge-set :

$$E(G) = E(G_1\backslash v_1) \cup E(G_2\backslash v_2) \cup \{\{u_1, u_2\} \mid u_i \in N(v_i), i = 1, 2\}.$$

Equivalently, we say that (G_1, G_2) is a *decomposition* of G, and v_1, v_2 are the *markers* of the decomposition (see [4], and Fig. 1; here, and in all other figures in this paper, row-vertices are denoted by o's and column-vertices by x's). Observe that $N(v_1) \cup N(v_2)$ induces in G a complete bipartite subgraph whose removal disconnects G into $G_1\backslash v_1$ and $G_2\backslash v_2$. Also if G_1 and G_2 are RU, then their composition is easily seen to be SU. Conversely, Conforti and Rao proved :

Proposition 2 [2] *A graph is SU if and only if it is RU, or it is the composition of two SU graphs. If G is SU, then every maximal complete bipartite subgraph $K(R, C)$ of G with $\mid R \mid, \mid C \mid \geq 2$ disconnects G into G_1 and G_2, so that :*

(a) $R \subseteq V(G_1), C \subseteq V(G_2)$;

(b) if $c \in C$ and $r \in R$, then the subgraphs of G induced by $V(G_1) \cup c$ and $V(G_2) \cup r$ are SU, and G is (isomorphic to) the composition of these subgraphs along c and r.

Using Proposition 2, one can naturally associate with every SU graph G a *tree-decomposition* of G into RU subgraphs in the following sense. With G, we associate a tree $T = (I, F)$ and a family of RU graphs $H_i (i \in I)$. To each $e = \{i, j\}$ in F correspond two markers $v_{e,i} \in H_i$ and $v_{e,j} \in H_j$. The original graph G can be recovered from this decomposition by repeatedly carrying out the following elementary operation, until T is reduced to one vertex :

- Pick a leaf i of T, denote by j its unique neighbor in T, and let $e = \{i, j\}$.
- Replace H_j by the composition of H_i and H_j along $v_{e,i}$ and $v_{e,j}$.
- Replace T by $T\backslash i$.

See [4], [5] for similar tree-decompositions of graphs. Conforti and Rao [2] state in more precise terms an algorithm yielding such a decomposition. If G has m row- and n column- vertices, then the number of subgraphs in the decomposition is easily proven (by induction) to be at most $m + n - 2$. Using Yannakakis' linear time recognition algorithm for RU graphs

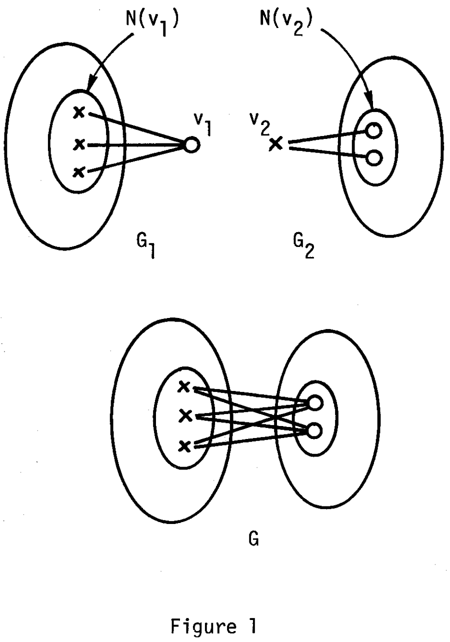

Figure 1

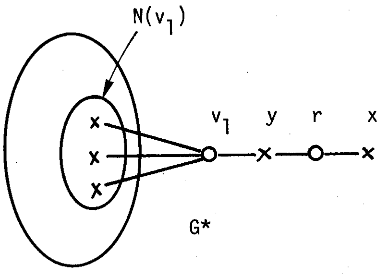

Figure 2

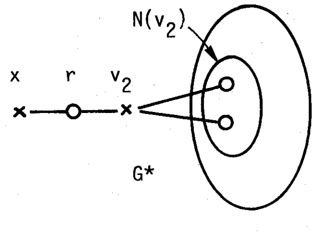

Figure 3

as a subroutine ([5]), the decomposition procedure can be implemented to run in time $O((m+n)mn)$. Our goal here is to indicate how this decomposition can be used to solve efficiently problem P when the constraint-matrix is SU.

3 THE REDUCTION STEP

The main step of our procedure consists in solving a problem of type P, where the associated graph G is the composition of two graphs G_1 and G_2, and either G_1 and G_2 is RU. We describe now an efficient way to carry out this step.

Let $v_1 \in V(G_1)$ and $v_2 \in V(G_2)$ be the markers of the decomposition (G_1, G_2). We assume without loss of generality that the vertices in $N(v_1)$ are column-vertices of G. Accordingly, v_1 and $N(v_2)$ will henceforth be considered row-vertices of G_1, and v_2 will be considered a column-vertex of G_2. If all vertices of $N(v_2)$ are packing in G, then we define v_1 to be packing in G_1; if they are all covering in G, then v_1 is covering in G_1; otherwise, v_1 is partitioning in G_1. The weight of v_2 in G_2 is set to some arbitrary value α, which can temporarily be thought of as being zero : the specific value of α will only turn out to be relevant if G_1 is RU, and will be redefined in that case (see Section 3.2). All vertices in $V(G_1)\backslash v_1$ and $V(G_2)\backslash v_2$ retain the same weight or type (i.e., packing, covering or partitioning) as in G.

3.1 G_2 is RU.

Let $N(v_2) = \{p_1, ..., p_s, c_1, ..., c_t\}$, where $p_1, ..., p_s$ are either packing or partitioning vertices of G, and $c_1, ..., c_t$ are covering vertices of G. Denote by P_0 the problem associated with $G_2\backslash v_2$ (i.e., the problem obtained by restricting P to the rows and columns associated with vertices of $G_2\backslash v_2$), and denote by ω_0 the optimal value of P_0. Similarly, let P_1 be the problem associated with $G_2\backslash N$, where :

$$N = v_2 \cup N(v_2) \cup N(p_1) \cup \cdots \cup N(p_s),$$

and let ω_1 be the optimal value of P_1. Intuitively, P_0 and P_1 are derived from the problem associated with G_2 by fixing to 0 or to 1 the variable corresponding to v_2.

Observe that both $G_2\backslash v_2$ and $G_2\backslash N$ are RU. Hence, by Proposition 1, ω_0 and ω_1 can be computed efficiently. If $G_2\backslash N$ has no vertices, then ω_1 is simply set to zero. In case P_0 (resp. P_1) is infeasible, we set ω_0 (resp. ω_1) to $-\infty$. If both P_0 and P_1 are infeasible, then P is infeasible, and we are done.

Now, we consider the graph G^* with vertex-set

$$V(G^*) = V(G_1) \cup \{x, y, r\},$$

(where x, y, r are new vertices), and with edge-set

$$E(G^*) = E(G_1) \cup \{\{y, v_1\}, \{x, r\}, \{y, r\}\}$$

(see Fig. 2). We define a problem P^* on G^*, in the following way :

(a) all vertices in $V(G_1)\backslash v_1$ have the same weights or types as in G;

(b) y has weight ω_0;

(c) x has weight ω_1;

(d) r is a partitioning vertex.

Proposition 3 *P and P^* have the same optimal value.*

Proof.

1. Consider a feasible solution of P^* with value ω^*, and the corresponding set S of column-vertices of G^*. Since r is a partitioning vertex of G^*, exactly one of x or y is in S. Assume first that $x \in S$, and let S_1 be an optimal solution of P_1. Then, $(S\backslash x) \cup S_1$ is a feasible solution of P, with value ω^*. Similarly, if $y \in S$, let S_0 be an optimal solution of P_0. Then, $(S\backslash y) \cup S_0$ is feasible for P, and has value ω^*. So, we can conclude at this point that the optimal value of P is always at least at large as the optimal value of P^*.

2. Conversely, consider now a feasible solution of P with value ω, and the corresponding set S of column-vertices of G. Let $Z = S \cap V(G_2)$. Assume first that S contains no vertex from $N(v_1)$. Then, it is easy to see that Z defines a feasible

solution of P_0. Hence, the total weight of Z is at most ω_0. It follows that $(S \cap V(G_1)) \cup \{y\}$ is a feasible solution of P^* with value at least ω. On the other hand, if S intersects $N(v_1)$, then Z is a feasible solution of P_1, and hence has weight at most ω_1. So, in that case, $(S \cap V(G_1)) \cup \{x\}$ is a feasible solution of P^* with value at least ω. Thus, the optimal value of P^* is at least the optimal value of P, and the proof is complete. □

One should observe at this point that, if G_1 is RU, then so is G^*. Hence, in that case, P^* can be solved efficiently, and Proposition 3 provides a complete solution to problem P.

3.2 G_1 is RU.

In this case, we denote by P_0 the restriction of P to $G_1\backslash v_1\backslash N(v_1)$, and by ω_0 the optimal value of P_0. Observe that ω_0 is the best value that can be achieved for G_1, under the restriction that all variables corresponding to vertices in $N(v_1)$ be fixed at 0. Similarly, we let P_1 be the problem associated with G_1, and ω_1 be its optimal value. As before, ω_0 and ω_1 can be computed efficiently, since both G_1 and $G_1\backslash v_1\backslash N(v_1)$ are RU. If $G_1\backslash v_1\backslash N(v_1)$ has no vertices, then ω_0 is set to zero. If P_0 (resp. P_1) is infeasible, then ω_0 (resp. ω_1) is set to $-\infty$.

Consider now the graph G^* defined by :

$$V(G^*) = V(G_2) \cup \{x, r\},$$

(where x, r are new vertices), and

$$E(G^*) = E(G_2) \cup \{\{r, v_2\}, \{x, r\}\}$$

(see Fig. 3). We associate with G^* the problem P^* in which :

(a) all vertices of $V(G_2)\backslash v_2$ have the same weights or types as in G;

(b) v_2 has weight ω_1;

(c) x has weight ω_0;

(d) r is a partitioning vertex.

Proposition 4 *P and P^* have the same optimal value.*

Proof. The proof is similar to that of Proposition 3, and is left to the reader. □

Notice that, just as in the previous case, G^* is RU if G_2 is RU, and P^* can then be solved by Yannakakis' procedure.

4 THE OPTIMIZATION ALGORITHM

An efficient procedure to solve problem P on an SU graph G can now be inferred from the results stated above. Assume that a tree-decomposition of G into RU graphs is available. Say $T = (I, F)$, $H_i (i \in I)$ is such a decomposition, with markers $v_{e,i} \in H_i$ and $v_{e,j} \in H_j$ for each edge $e = \{i, j\}$ in F (see Section 2). All vertices in $\bigcup_{i \in I} V(H_i)$ (i.e., including the markers) have some weights or types, as defined at the beginning of Section 3. Then, the following algorithm yields an optimal solution of problem P.

SU ALGORITHM

Step 1. If T contains at least two vertices, then pick a leaf i of T, denote by j its unique neighbor and let $e = \{i, j\}$. Else, go to Step 4.

Step 2. If $v_{e,i}$ is a column-vertex, then let $G_1 = H_j, G_2 = H_i, v_1 = v_{e,j}, v_2 = v_{e,i}$; let G^* be the graph obtained by applying to (G_1, G_2) the reduction step described in Section 3.1 ; let $H_j \leftarrow G^*$. Else, if $v_{e,i}$ is a row-vertex, then let $G_1 = H_i, G_2 = H_j, v_1 = v_{e,i}, v_2 = v_{e,j}$; let G^* be the graph obtained by applying to (G_1, G_2) the reduction step described in Section 3.2 ; let $H_j \leftarrow G^*$.

Step 3. Replace T by $T\backslash i$, and go to Step 1.

Step 4. Let i be the unique vertex of T, solve the problem associated with H_i, return its optimal solution, and stop.

Let us illustrate the use of the SU algorithm on a small example.

Example. Problem P is associated as follows with the SU graph G represented in Fig. 4 :
(i) vertices 1, 6, 7 are covering, vertex 8 is partitioning;
(ii) the weight of the remaining vertices are : $w_2 = -5, w_3 = 2, w_4 = 2, w_5 = 3, w_9 = -4, w_{10} = -3$.

A tree-decomposition $T = (\{t_1, t_2, t_3\}, F)$ of G into H_1, H_2, H_3 is indicated in Fig. 5. The composition of H_1 and H_2 is to be taken along v_1, v_2; the composition of H_2 and H_3 is along v_3, v_4. Observe that H_1, H_2 and H_3 are RU. In accordance with the rules described in Section 3, v_1 is a partitioning vertex, v_3 is a covering vertex, and v_2, v_4 have arbitrary weights.

We start the execution of the SU algorithm by picking first leaf t_3 of T. The optimal value of problem P_0 (resp. P_1) associated with $H_3 \backslash v_4$ (resp. $H_3\backslash\{6,7,v_4\}$) is $\omega_0 = -2$ (resp. $\omega_1 = 3$). The graph H_2 is updated as indicated in Fig. 6, where r is a partitioning vertex, y has weight -2 and x has weight 3.

We pick next leaf t_1 of T. Since $H_1\backslash\{9, 10, v_1\}$ is empty, we let $\omega_0 = 0$. The optimal value of problem P_1 defined on H_1 is $\omega_1 = -3$. Two new vertices are added to H_2 as shown in Fig. 7 : s is a partitioning vertex and z has weight 0. Also, the weight of v_2 (which has left so far undefined) is set to -3.

The optimal solution of H_2 is $S^* = \{3, 4, x, v_2\}$, and has weight $\omega = 4$. The corresponding optimal solution of P on G is easily traced back : it is given by $S = \{3, 4, 5, 10\}$. □

Proposition 5 *Suppose that problem P can be solved in time $h(m,n)$ for an $m \times n$ RU constraint matrix, where h is a subadditive function. Then, the SU algorithm solves P in time $O(h(m,n))$ for an $m \times n$ SU constraint matrix.*

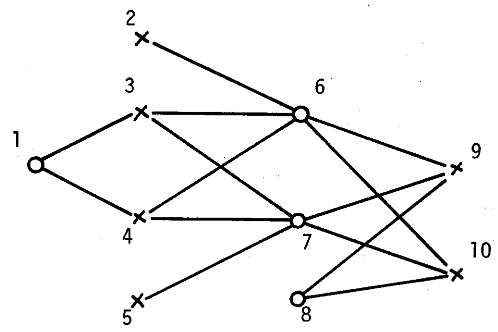

Figure 4

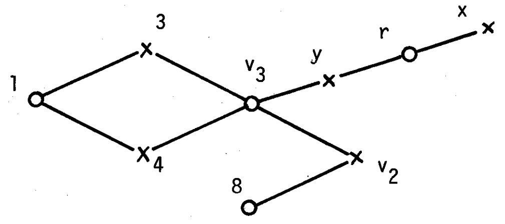

Figure 6

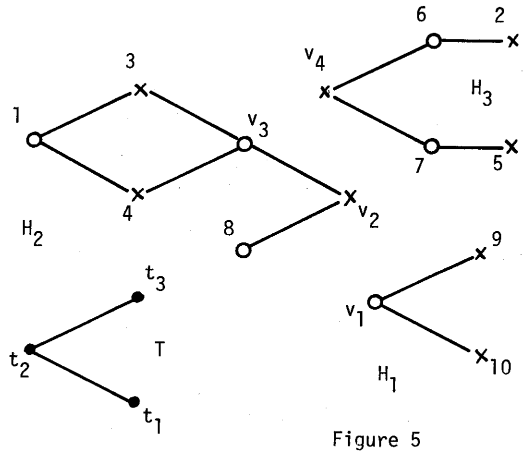

Figure 5

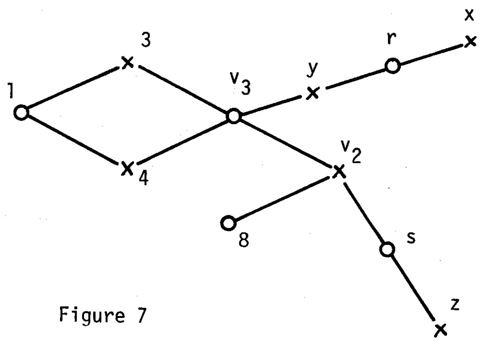

Figure 7

Proof.

1. Observe that, whenever the SU algorithm picks a leaf i of T in Step 1, the associated graph H_1 is RU, and contains exactly one marker, namely $v_{e,i}$: this is easily verified to hold at the first iteration of Step 1 (e.g. by induction on the number of vertices of T), and to remain true throughout the execution of the algorithm. Hence, the problems P_0 and P_1 to be solved in Step 2 of the algorithm are always associated with RU graphs, for which all relevant weights are precisely known (and are independent of the arbitrary weights associated with some markers at the beginning of the algorithm). The validity of the whole procedure follows then from Propositions 3 and 4. Notice that an optimal solution to P can be reconstructed from the solution returned by the algorithm (as in the proof of Proposition 3).

2. So. P is solved by solving at most two RU subproblems for each subgraph in the tree-decomposition, and the complexity of the SU algorithm is of the order of $\sum_{i\in I} h(m_i,n_i)$, where m_i (resp. n_i) is the number of row-vertices (resp. column-vertices) of the graph H_i considered in Step 2 of the algorithm. Let now $I = J \cup K$, where $i \in J$ if H_i is a star in the original decomposition $T = (I, F)$ of G, and $K = I \backslash J$.

Claim 1. $\sum_{i\in J} h(m_i,n_i) = O(m+n)$.
Notice that, if $i \in J$, i.e., H_i is a star, then H_i contains only one marker, namely its center (since a marker always has at least two neighbors). So, if $i \in J$, then i is a leaf of T. It follows that, when i is eventually picked in Step 1 of the algorithm, H_i is still as in the original decomposition, i.e. H_i is still a star. Now, when the underlying graph is a star, problem P can trivially be solved in linear time. Hence, the subgraphs H_i with $i \in J$ contribute for $O(\sum_{i\in J} h(m_i+n_i))$ time to the total complexity of the algorithm. Since each star has only one marker, and $|J| \le |I| \le m+n-2$, it follows that :

$$\sum_{i\in J} h(m_i,n_i) = O(\sum_{i\in J}(m_i+n_i)) = O(m+n).$$

Claim 2. $\sum_{i\in K} h(m_i,n_i) = O(h(m,n))$.
Let $k = |K|$. We first show that $k \le min(m,n)$. Indeed, being a tree on k vertices, $T\backslash J$ has $k-1$ edges. So, $\bigcup_{i\in K} V(H_i)$ contains $2(k-1)$ markers, $k-1$ of which are row-vertices. Therefore, the total number of row-vertices in $\bigcup_{i\in K} V(H_i)$ is at most $m+k-1$. On the other hand, for all $i \in K$, H_i has at least two row-vertices (else, H_i would be a star). It follows that $2k \le m+k-1$, and hence $k \le m$. A similar argument shows that $k \le n$. The number of new vertices (of type x, y, r) added to the H_i's during the reduction steps performed by the algorithm is clearly at most linear in k. Hence, $\sum_{i\in K} m_i = O(m+k) = O(m)$, $\sum_{i\in K} n_i = O(n+k) = O(n)$, and :

$$\sum_{i\in K} h(m_i,n_i) = h(\sum_{i\in K} m_i, \sum_{i\in K} n_i) = O(h(m,n)).$$

From Claim 1 and Claim 2 (and under the harmless assumption that $m+n = O(h(m,n))$, we conclude that the overall complexity of the SU algorithm is $O(h(m,n))$. □

References

[1] E. Balas and M. W. Padberg, On the set covering problem, *Operations Research* 20 (1972) 1152–1161.

[2] M.Conforti and M. R. Rao, Structural properties and recognition of restricted and strongly unimodular matrices, *Mathematical Programming* 38 (1987) 17–27.

[3] Y.Crama, P. L. Hammer and T. Ibaraki, Strong unimodularity for matrices and hypergraphs, *Discrete Applied Mathematics* 15 (1986) 221–239.

[4] W. H. Cunningham, Decomposition of directed graphs, *SIAM Journal on Algebraic and Discrete Methods* 3 (1982) 214–228.

[5] M.Yannakakis, On a class of totally unimodular matrices, *Mathematics of Operations Research* 10 (1985) 280–304.