Global constraint for the set covering problem

Sébastien Mouthuy **Yves Deville** **Grégoire Dooms**

Department of Computing Science and Engineering
Université Catholique de Louvain
B-1348 Louvain-la-Neuve - Belgium

sebastien.mouthuy@student.uclouvain.be,{yves.deville,dooms}@info.ucl.ac.be

Abstract

This paper proposes a constraint programming approach to solve the set covering problem. This combinatorial optimisation problem is very useful to formulate a lot of real-life applications (crew assignment, scheduling, construction of optimal logical circuits) and a lot of other well-known combinatorial optimisation problem (vertex cover, dominating set, independent set). The set covering problem is NP-Complete. This paper proposes a constraint SC for the set covering problem and a propagator using a lower bound based on an IP relaxation. We also present an incremental algorithm for computing this lower bound whose internal data structure can be updated very quickly for small changes in the problem, making it particularly well-suited for search-tree schemes. Our approach will be compared with two other propagators based on the linear relaxation and on a greedy approach.

1 Introduction

The set covering problem (SC) can be defined as follows. Let $\mathcal{U} = \{1, \dots, m\}$ be a set of m elements. Let X be a collection of subsets of $\mathcal{U}$, i.e. $X = \{S_1, \dots, S_n\}$ where $S_i \subseteq \mathcal{U}, (1 \leq i \leq n)$ and let c_j be a weight associated to each subset $S_j, 1 \leq j \leq n$. SC calls for a subset T of indices of X covering $\mathcal{U}$: $T \subseteq \{1, \dots, n\} \mid \bigcup_{i \in T} S_i = \mathcal{U}$ and such that $\sum_{j \in T} c_j$ is minimum. An example of such a problem is illustrated in Fig.1.

A lot of real-life applications can be formulated in terms of a SC problem : crew assignment and scheduling [25, 11, 6], construction of optimal logical circuits [22], location of emergency facilities [12, 26], routing problems. Additionally, a lot of well-known combinatorial problems can be easily formulated by mean of

$$\mathcal{U} = \{1,2,3,4,5\}$$
$$S_1 = \{1,3,5\},\ S_2 = \{1,2,4\},\ S_3 = \{5,2\},$$
$$S_4 = \{1,3,2\}$$
$$c_j = 1, \quad (1 \leq j \leq 4)$$

$$Sol = \{1,2\}$$

FIG. 1 – Example of a set covering problem. It is obvious that there is no subset S_i covering $\mathcal{U}$ in itself. We observe $S_1 \cup S_2$ covers entirely $\mathcal{U}$, thus $\{S_1, S_2\}$ is a set cover of $\mathcal{U}$ of minimum cardinality.

the SC problems such as vertex cover, dominating set, independent set.

A lot of work has been done to solve this problem either to optimality or to a nearly optimal solution, usually using integer programming. Two good reviews are [5, 7]. The integer programming formulation of SC is

$$SC^* = \min_{x \in \mathbb{B}^n} \sum_{1 \leq j \leq n} c_j x_j \qquad (SC)$$

subject to

$$\sum_{1 \leq j \leq n} a_{ij} x_j \geq 1 \quad \forall i \in \mathcal{U} \qquad (1)$$

$$x_j \in \{0,1\} \quad \forall j = 1 \dots, n \qquad (2)$$

where $a_{ij} = 1$ iff $i \in S_j$. $\mathbb{B} = \{0,1\}$.

SC problems are very difficult. The results in [1] imply that it does not have a polynomial time approximation scheme unless P = NP ; that is, there exists a constant $\epsilon > 0$ such that the problem cannot be approximated in polynomial time with a ratio smaller than $1+\epsilon$ unless P = NP. More negative results about the complexity of this problem can be found in [19].

This paper defines the SC constraint, a global constraint for the set covering. A propagator for this constraint is also presented. It uses a shaving technique and a lower bound based on an IP relaxation. The main advantage of the algorithm implemented in the propagator is that its internal data structure can be updated very quickly along the search tree. This allows fast recomputation of the lower bound between adjacent nodes in the search tree.

This algorithm was implemented in a CP framework. This allows to easily express problems related to the set covering problem as CSPs. The advantage of this paradigm is that once there exists a global constraint for SC, one can use it to model real-life problems and add many arbitrary auxiliary constraints, as it is often required in such problems (scheduling, circuit construction).

In Section 2, we describe a general CP approach to SC and existing work that has already been done in this direction. Section 3 describes a relaxation of SC and presents an incremental algorithm to solve it efficiently. Section 4 presents computational results of our CP approach.

2 A CP approach to SC

2.1 General approach

The aim of this CP approach is to design a global constraint for SC having the form $SC(N,T,\mathcal{U},X = \{S_1,\ldots,S_n\},Cost)$ where

- $\mathcal{U} = \{1,\ldots,m\}$ is a set of elements
- N is an integer
- X is a collection of subsets $S_i \subseteq \mathcal{U}, (1 \le i \le n)$
- T is a subset of the indices of X : $T \subseteq \{1,\ldots,n\}$
- $Cost$ is a function giving a weight for each subset S_i ; $Cost : X \to \mathbb{R}$. For sake of simplicity, we will denote $Cost(S_i)$ by c_i.

This constraint holds if and only if

$$\mathcal{U} = \bigcup_{i \in T} S_i \tag{3}$$

$$\sum_{i \in T} c_i \le N \tag{4}$$

In the following, N and T will be considered as variables and $\mathcal{U}$, $S_i (i = 1,\ldots,n)$, $Cost$ will be considered constant. N is a finite-domain (FD) variable and T is a set variable [10, 23]. $\overline{N}$ denotes the upper bound on N and $\underline{N}$ denotes its lower bound : $\underline{N} \le N \le \overline{N}$. We use the same notations for T : $\underline{T} \subseteq T \subseteq \overline{T}$.

We observe that (3) is easy to satisfy, while, for small values of N, finding a T satisfying (4) is hard. This is why we need good lower bounds on N in order to prune the domains of N and T.

The objective of the filtering algorithm for this global constraint SC is to prune the domain of N and T by removing values that do not belong to any solution of this constraint.

2.1.1 Pruning of N

From the structure of SC, it is easy to see that the tightest upper bound on N is $N \le \sum_{i \in \overline{T}} c_i$. Indeed if there exists a solution to the constraint (i.e. an assignment of variables N and T such that (3) and (4) hold), then taking $N = \sum_{i \in \overline{T}} c_i, T = \overline{T}$ would be a solution to the global constraint SC too. Hence the pruning rule is $\overline{N} = \min(\overline{N}, \sum_{i \in \overline{T}} c_i)$.

However computing a lower bound on N is more difficult. Computing the tightest lower bound on N requires to solve SC to optimality, i.e. to compute SC^*. This problem is NP-Complete [13]. In this paper we use a lower bound of SC^* to prune the domain of the variables. To compute such a lower bound one solves a relaxation of SC. Let denote such a relaxation by $SCRel$ and its optimal value by $SCRel^*$. Then we have

$$lb_{SC} = SCRel^* \le SC^*$$

The pruning rule is $\underline{N} = \max(\underline{N}, lb_{SC})$.

2.1.2 Pruning of T

T is a set domain variable. The pruning algorithm should prune the domain of T according to constraints (3) and (4). From the first constraint we can derive the following two pruning rules :

(P1) The constraint SC does not hold if $\mathcal{U}$ cannot be covered by the available subsets, thus $\exists e \in \mathcal{U}, \forall i \in T \; e \notin S_i \to fail$

(P2) Because the indices in T should cover $\mathcal{U}$, when there is only one subset S_i covering an element e, S_i should be part of the cover. So we have $\exists e \in \mathcal{U}, \exists! i : \; e \in S_i \to i \in T$

Without constraint (4), these pruning rules would achieve arc-consistency (all partial solution could be extended to a feasible solution). In order to prune constraint (4), two other pruning rule should be considered.

(PA) Remove from T the indices of subsets that do not belong to any cover of weight in $[\underline{N};\overline{N}]$: $\overline{T} \leftarrow \overline{T} \setminus \{i \in \overline{T} \setminus \underline{T} \mid \neg SC(N, T \cup \{i\}, \mathcal{U}, X)\}$

(PB) Put in T all indices such that S_i belongs to all covers of weight in $[\underline{N};\overline{N}]$: $\underline{T} \leftarrow \underline{T} \cup \{i \in \overline{T} \setminus \underline{T} \mid \neg SC(N, T \setminus \{i\}, \mathcal{U}, X)\}$

Unfortunately, we saw that achieving arc-consistency for the SC constraint is NP-Complete, thus these last two rules can only be approximated. We will use a lower bound to prune according to constraint (4) :

(P'A) $\overline{T} \leftarrow \overline{T} \setminus \{i \in \overline{T} \setminus \underline{T} \mid lb_{SC}(N, T \cup \{i\}, \mathcal{U}, X) > \overline{N}\}$

(P'B) $\underline{T} \leftarrow \underline{T} \cup \{i \in \overline{T} \setminus \underline{T} \mid lb_{SC}(N, T \setminus \{i\}, \mathcal{U}, X) > \overline{N}\}$

Pruning (P'A) (resp. (P'B)) can be done by mean of a shaving technique : we remove from $\overline{T}$ (resp. add in $\underline{T}$) each value $i \in \overline{T} \setminus \underline{T}$ at turn and recompute a new lower bound lb_i on N. If $lb_i > \overline{N}$, then we must put i in $\underline{T}$ (resp. remove i from $\overline{T}$).

2.2 Existing CP approaches

As far as we know, SC has never been tackled directly in CP. However some work has been done for the constraint $NValue$ counting the number of distinct values used by a vector of variables, that is close to SC. This constraint was first introduced in [21]. This is a generalization of the well-known $AllDiff$ constraint [24, 28].

Bessière et al. [4] decomposed $NValue$ into two constraints : $AtLeastNValue(N, X = \{X_1, \ldots, X_m\})$ and $AtMostNValue(N, X = \{X_1, \ldots, X_m\})$ where N is an integer and X_i are FD variables having $D(X_i)$ as domain. $AtMostNValue$ holds if the assignment of the m variables X_i uses at most N different values. Formally AtMostNValue holds iff $|\{X_i : 1 \leq i \leq m\}| \leq N$ and AtLeastNValue holds iff $|\{X_i : 1 \leq i \leq m\}| \geq N$. We now show that $AtMostNValue$ is equivalent to SC with uniform weights ($c_j = 1, \forall j$).

Unicost SC is strictly equivalent to $AtMostNValue$ (NV) Define $\mathcal{U} = \{1, \ldots, m\}$ and $S_i = \{j \mid i \in D(X_j)\}, \forall i \in \bigcup_{1 \leq i \leq m} D(X_i)$. Entailment of the original $AtMostNValue$ is equivalent to entailment of $SC(N, T, \mathcal{U}, X)$ and vice versa.

This shows that every filtering algorithm developped for one of these two problems can directly be used to prune variables of the other problem.

2.2.1 Computing a lower bound on N in $AtMostNValue$

Before describing the existing approaches to compute lower bounds on N, we introduce the intersection graph $G_X = (V, E)$ of a set of variables $X = \{X_1, \ldots, X_m\}$, where $V = \{1, \ldots, m\}$ and $E = \{(i,j) \mid D(X_i) \cap D(X_j) \neq \emptyset\}$. We can observe that, for a given instance of SC, finding the largest independent set (set of vertices being pairwise non-adjacent) of G_X is equivalent as solving the dual problem of SC, that is the set packing problem. In the case of a unicost SC ($c_j = 1, \forall j = 1, \ldots, n$), the formulation of the set packing problem can be formulated as

$$SP^* = \max_{y \in \mathcal{B}^m} \sum_{1 \leq i \leq m} y_j \qquad (SP)$$

such that

$$y_i + y_j \leq 1 \quad \forall (i,j) \in E$$

$$y_i \in \{0,1\} \quad \forall 1 \leq i \leq m$$

The set packing problem is an NP-Hard problem. If we denote SC^*(resp. SP^*) as the optimal value for the set covering problem (resp. set packing problem), SCL^*(resp. $SPLin^*$) as the optimal solution of the linear relaxation of the set covering problem (resp. set packing problem), we have from optimisation theory [29] that

$$SP^* \leq SPLin^* = SCL^* \leq SC^* \qquad (5)$$

Thus a lower bound on SP^* is also a lower bound lb_{SC} of SC^*. If we denote the independance number of a graph G by $\alpha(G)$, then from (5),

$$lb_{SC} = \alpha(G_X) = SP^* \leq SC^* \qquad (6)$$

2.3 Four lower bounds on N

Four approaches exist for computing a lower bound on SC^*.

Linear relaxation of SC (SCL^*) If we relax the integrality constraint of SC, i.e. replacing $x_i \in \{0,1\}$ by $0 \leq x_i \leq 1$ we obtain the linear relaxation of SC, that we denote SCL. From (5) we have that SCL^* is a lower bound on SC^*. This bound is better than any bound on SP^*. However it is relatively expensive to compute.

Ordered interval algorithm (OI) In [2], Beldiceanu deals with NValue where the domain of each variables X_i are intervals. In this special case, he achieves bound-consistency in polynomial time. If we denote m domain variables whose domains are intervals by $I = \{I_1, \ldots, I_m\}$, then this algorithm computes $\alpha(G_I)$. If we denote the smallest enclosing interval of the domain of each variable X_i by I_i, then this algorithm can be used to compute a lower bound of $\alpha(G_X)$, as G_I contains at least all edges of G_X and eventually some others, thus $\alpha(G_I) \leq \alpha(G_X)$.

Greedy algorithm (MD - for Minimum Degree) Consider the graph $G_X = (V, E)$. Let $\Gamma(v), v \in V$ be the set of neighbours of node v and

$\Gamma(S) = \bigcup_{v \in S} \Gamma(v)$. Initially, set $S = \emptyset$. Choose the node $v \in V \setminus (S \cup \Gamma(S))$ of minimum degree. Add v to S and loop until $S \cup \Gamma(S) = V$. At the end S will be an independent set of G_X and $lb_{SC*} = |S|$ will be a lower bound of $\alpha(G_X) = SP^* \leq SC^*$.

Turán's approximation (TA) Use the lower bound for $\alpha(G_X)$ proposed by Turán in [27] : $\left\lceil \frac{n^2}{2m+n} \right\rceil \leq \alpha(G_X) = SP^* \leq SC^*$.

From equation(5), we deduce that SCL^* is a tighter bound than the other three. MD is at least as strong as TA. OI is sometimes better, sometimes worse, than MD and TA (see [4] for details).

Since AtMostNValue is equivalent to SC, we can use TA, SCL^*, MD or OI to compute a valid lower bound on N and to prune the domain of T as explained in section 2.1.

3 2SC, another relaxation of SC

In this section we present another relaxation of SC. It can be used in a CP framework to compute a lower bound of SC^* and to prune the domain of N as explained in the previous section. The main advantage of this relaxation is that it can be solved incrementaly, being a good choice in a search-tree scheme as will be pointed out in the experimental section.

3.1 Principles

An interesting relaxation is based on the fact that a set covering problem with subsets S_i containing exactly two elements, that will be denoted as 2SC, can be solved in polynomial time. This relaxation was explored in a strictly MIP context in [8] for the set covering problem and in [17] for the set partitionning problem (i.e. the set covering with equality constraints instead of inequalities).

To solve $2SC$, we can build a graph G with one vertex representing one element of $\mathcal{U}$ and one edge representing one 2-set (a set with exactly two elements) S_i from the collection X. A minimum weight edge cover of G would represent a minimum set cover of 2SC. Every set covering problem SC can be decomposed into a 2SC problem in such a way that the graph G is bipartite. This is motivated by efficiency as computing minimum edge covers in general graph is computationaly more expensive (as for the matching problem).

Let us define $E(S_i)$ the decomposition of S_i in 2-sets : $E(S_i) = \{S_i^1, \ldots, S_i^k\}$. We have $\forall i : 1 \leq i \leq n$,

$$|S_i^j| \quad = \quad 2 \qquad \forall S_i^j \in E(S_i) \tag{7}$$

$$S_i \quad = \bigcup_{S_i^j \in E(S_i)} S_i^j \tag{8}$$

If we define weight c_i^j on 2-subsets S_i^j such that

$$c_i = \sum_{S_i^j \in E(S_i)} c_i^j, \qquad (1 \leq i \leq n) \tag{9}$$

where c_i is the cost associated to the subset S_i, then 2SC will be a relaxation of SC because any set cover of SC will be a set cover for 2SC with exactly the same weight. Clearly a solution to the relaxation 2SC will provide a lower bound of the minimum weight of the original SC problem.

If we denote the optimal value of the 2SC problem as $2SC^*$, it can be shown that

$$2SC^* \leq SCL^* \leq SC^* \tag{10}$$

This lower bound can thus be used in the propagator for the SC constraint. We now show how to compute this lower bound efficiently.

3.2 Incremental computation of $2SC^*$

The relaxation presented in the previous section would not be of any interest if we could not solve it efficiently to optimality. The aim of this section is to present a new incremental algorithm for the computation of $2SC^*$. We illustrate how we can solve 2SC and how we can update the optimal solution when we put some restriction on the edge cover we want to find, for instance by imposing that all edges from the decomposition of a subset S_i must be part of the edge cover. This is very important in a search-tree scheme in order to propagate domain updates very quickly.

The weighted 2SC problem introduced in the previous section has a lot of similarities with the maximum weight matching of a graph. We therefore present a reasoning based on the Hungarian method [16] to solve this last problem. As in the previous section, we will assume the graph G built from 2SC is bipartite.

First we will give six necessary and sufficient conditions for an edge cover R of a bipartite graph to be of minimum weight. Then we will describe an algorithm maintaining the five first conditions as invariant and looping in order to end with the sixth condition holding. Third we will explain how we can update the data structure used by the algorithm to reflect small changes in the bipartite graph (removal/insertion of edges). Finally we will enumerate a few algorithmic enhancements of this algorithm.

The aim of the 2SC problem is to find a minimum weight edge cover R of a bipartite graph $G = (V, E)$. The edge cover can be restricted : an edge $(i, j) \in$

$SURE$ if and only if (i,j) must be part of the cover, $(i,j) \in REMOVED$ if and only if it cannot be part of the cover. A node i will be defined as SURE if it is the endpoint of an edge being in SURE. A vertex v from G is critical wrt R if exactly one edge from R is incident to v. A vertex v is non-critical otherwise. For now we consider edge weights being constant.

In order to have a strong basis for the following, here is the IP formulation of 2SC :

$$2SC^* = \min \sum_{e \in F} c_i^j y_i^j + \sum_{e \in SURE} c_i^j y_i^j \quad (11)$$

$$\sum_{(i,j)\in E} y_i^j \geq 1 \quad \forall \; vertex \; i | i \notin SURE \quad (12)$$

$$0 \leq y_i^j \leq 1 \quad \forall (i,j) \in F \quad (13)$$

where $F = E \setminus (SURE \cup REMOVED)$ is the set of edges which we do not know whether they are part of an optimal edge cover. Constraint (13) is necessary as we allow negative weights c_i^j. It should be noted that in the previous formulation, we did not constrain y_i^j to be integral. Indeed we can show that there exists an integral optimal solution for this continuous formulation (because the matrix defining the constraint (12)-(13) is totaly unimodular). Results in optimisation theory (commonly called the KKT conditions, see [14, 15]) give necessary and sufficient conditions for a solution of a LP problem to be optimal. Let $\pi(i)$ be the dual variables of constraints (12). The KKT conditions state that a set of edges (defined by y_i^j's) is a cover and is optimal iff

(i). $\pi(i) \geq 0 \quad \forall node \; i, \; \pi(i) = 0 \quad \forall \; i \in SURE$

(ii). $\bar{c}_i^j \leq 0$ if $(i,j) \in F$ and $y_i^j = 1$

(iii). $\bar{c}_i^j \geq 0$ if $(i,j) \in F$ and $y_i^j = 0$

(iv). $\sum_{(i,j)\in E} y_i^j - 1 \geq 0 \quad \forall \; vertex \; i$

(v). $\pi(v) = 0$ if vertex v is non-critical, $\forall v \in V_2$

(vi). $\pi(v) = 0$ if vertex v is non-critical, $\forall v \in V_1$

where the reduced cost $\bar{c}_i^j = c_i^j - \pi(i) - \pi(j)$. Algorithm 1 solves weighted 2SC optimaly and is based on the conditions (i)-(vi). Beginning with a solution respecting (i)-(v), the algorithm loops in such a way that strictly less vertices i don't respect (vi) at each iteration. It ends when no such vertex i exists.

The intuitive idea behind the algorithm (as for matching algorithms) is to select a node v_0 not respecting (vi) and to decrease the number of edges incident to v_0 being in the cover. This is done by finding alternating paths. An alternating path wrt a cover R from v_0 to v_k is a sequence $p = [v_0, v_1, \ldots, v_k]$ of nodes such that edges (v_i, v_{i+1}) are alternatevely from R and from $E \setminus R$. If $v_0 = v_k$ then exactly one edge of (v_0, v_1) and (v_{k-1}, v_k) belongs to R. An admissible alternating path L from v_0 to v_k is defined as an alternating path with

– Either $(v_0, v_1) \in E \setminus R$ or v_0 is non-critical
AND
– Either $(v_{k-1}, v_k) \in E \setminus R$ or v_k is non-critical

The following result, similar to [3], states that a cover augmented by an admissible alternating path remains a cover.

Proposition 1 *If L is an admissible path wrt a cover R of G, then the set of edges $R' = R \oplus L = (R \setminus L) \cup (L \setminus R)$ is a cover of G.*

The principle of the algorithm is to find a node i not respecting (vi) and augmenting the current over by an admissible path from i (with the first edge being in the cover). This would strictly decrease the number of edges incident to i and being in the cover. After some iterations, node i will become critical and will thus respect (vi). The algorithm loops until there is no more such node. The algorithm explores admissible path of increasing weight (shorter admissible path are prefered).

The inner loop (7)-(15) of Algorithm 1 modifies dual values π in order to find new admissible paths from node i. We can observe that if $\delta = \beta$ or $\delta = -\gamma$ then the tree T strictly grows. If $\delta = \alpha$ then an admissible path is found (i is non-critical and $e_k \in E \setminus R$). As the loop finishes if $\delta = \alpha$ (because $\exists w \in V_1^T \mid \pi(w) = 0$) and T cannot grow forever, the inner loop (7)-(15) will iterate a finite number of times.

The outer loop (5)-(18) iterates until node i respects condition (vi). After we modify the current edge cover $R \leftarrow R \oplus p$, there will be one less edge incident to node i in R because the first edge $(v_0, v_1) \in R$ for all admissible path $[v_0, v_1, \ldots, v_k]$ in T (R is still an edge cover from Proposition 1). Thus after a finite number of iterations, node i will become critical and the outer loop (5)-(18) will end.

The main advantage of this algorithm is its incremental behaviour. If we remove or add some edges, we just need to change the current cover locally in order to make constraints (i)-(v) hold. Then applying the algorithm will make (vi) hold. The algorithm should only search for a few new admissible paths of negative weight. Such a local adjustment can also be performed after the weights are changed (see next section). This is very attractive for us because algorithms for propagators should be highly incremental in order to be fast through all the search tree. This advantage is demonstrated in the experimental results (see section 4) where we can observe a small overall time per call to the propagator.

We can improve this algorithm similarly as in [20, p.423-424]. Instead of the inner loop (7)-(15), we can

Algorithm 1 Algorithm for weighted 2SC

```
$computeSCRel^* \equiv computeMin2SC(2SC = (G, C, M))$
PRE: $G = (V_1, V_2, E)$ a bipartite graph, $C = \{c_i^j\}$ : weights for all edge $(i, j) \in E$, $M \subseteq E$
POST: $R \cup M$ is a minimum weight edge cover $EC^*$ of $G$ such that $M \subseteq EC^*$
 1: Let $R \leftarrow E$ and define initial values for $\pi(v) \geq 0$ such that (i)-(v) hold
 2: Verify $R$ cover all nodes of $G$. If not then return IMPOSSIBLE
 3: Let $[v_1, \ldots, v_n]$ be an enumeration of all nodes of $V_1$
 4: for $i \mid \nexists j, (i, j) \in M$ do
 5:   while $v_i$ is non-critical and $\pi(v_i) > 0$ do
 6:     $T = (V_1^T, V_2^T, E^T) \leftarrow computeT(G = (V, E), R, v_i)$
 7:     while $\nexists w \in V_1^T \mid \pi(w) = 0$ and $\nexists w \in V_2^T, w$ non-critical do
 8:       $\alpha \leftarrow \min\left\{\{+\infty\} \cup \{\pi(v) \mid v \in V_1^T\}\right\}$
 9:       $\beta \leftarrow \min\left\{\{+\infty\} \cup \{\bar{c}_i^j \mid (i, j) \in E \setminus R, i \in V_2^T, j \notin V_1^T\}\right\}$
10:       $\gamma \leftarrow \max\left\{\{-\infty\} \cup \{\bar{c}_i^j \mid (i, j) \in R, i \in V_1^T, j \notin V_2^T\}\right\}$
11:       $\delta \leftarrow \min(\alpha, \beta, -\gamma)$
12:       $\pi(i) \leftarrow \pi(i) - \delta, \forall i \in V_1^T$
13:       $\pi(j) \leftarrow \pi(j) + \delta, \forall i \in V_2^T$
14:       $T = (V_1^T, V_2^T, E^T) \leftarrow computeT(G = (V, E), R, v_i)$
15:     end while
16:     Let $p$ be the alternating path from $v_i$ to $w$ in $T$
17:     $R \leftarrow R \oplus p$
18:   end while
19: end for
20: return $R \cup M$
$computeT(G = (V, E), R, v_0)$
PRE: $v_0 \in V_1 \cup V_2, R \subseteq E$
POST: Return a tree built with the alternating paths beginning at $v_0 : T = (V_A^T, V_B^T, E^T)$ such that
  – $V_A^T = \{v_k \in A \mid \exists$ an alternating path wrt $R$ $[v_0, v_1, v_2, \ldots, v_{k-1}, v_k]$ with $(v_0, v_1) \in R, \bar{c}_{v_{i-1}, v_i} = 0, \forall i = 1, \ldots, k$. $V_A^T = \{v_0\}$ if no such path exists.
  – $V_B^T = \{v_k \in B \mid \exists$ an alternating path wrt $R$ $[v_0, v_1, v_2, \ldots, v_{k-1}, v_k]$ with $(v_0, v_1) \in R, \bar{c}_{v_{i-1}, v_i} = 0, \forall i = 1, \ldots, k$. $V_B^T = \emptyset$ if no such path exists.
  – $E^T = \{(v_i, v_{i+1}) \mid \exists$ an alternating path wrt $R$ $[v_0, v_1, v_2, \ldots, v_i, v_{i+1}, \ldots, v_{k-1}, v_k]$ with $(v_0, v_1) \in R, \bar{c}_{v_{i-1}, v_i} = 0, \forall i = 1, \ldots, k$
```

simply build a shortest-paths tree from node i and deriving δ from it. The algorithm we implemented uses this improvement.

In the following section we show how to decompose the initial set covering problem into a 2SC problem in such a way that the underlying graph G is bipartite.

3.3 Construction of the bipartite graph

In order to compute this new lower bound on SC^*, we must build a bipartite graph G as explained in the previous section. Several graphs G are possible for a given instance of SC. For example a heuristic is proposed in [8]. Let $H_j = \{e \in \mathcal{U} \mid e \in S_j \text{ and } q_j = \lfloor |H_j| \rfloor\}$. In order to relax SC to a 2SC, they create a bipartition (R', R'') of the set $\mathcal{U} : \mathcal{U} = R' \cup R'',\ R' \cap R'' = \emptyset$. For a given element i, if $|\{j \mid i \in S_j\}| < q_j$, then $i \in R'$. Otherwise $i \in R''$. They build the bipartition by looping on all the elements in $\mathcal{U}$ and by putting elements in R' or in R'' accordingly. They add two different dummy elements to R' and R''. Next they decompose each set S_i into several 2-sets S_i^j such that they contain exactly one element in R' and one element in R'' and such that (7)-(8) hold $\forall i : 1 \leq i \leq n$. In this case the graph G will be bipartite and 2SC can be solved by finding a minimum weight edge covering on this bipartite graph.

4 Computational results

From the computational results presented in [4], we observe that TA and OI achieve far less pruning than the two other methods (MD and SCL^*) for the same computational cost than MD. Thus we decided to compare $2SC$ with SCL^* and MD. Results are presented in Table-1.

We generated random set covering problems by specifying four different parameters

M The number of elements to cover

N The number of subsets in the collection X

S^- The minimum cardinality of the subsets in the collection X

S^+ The maximum cardinality of the subsets in the collection X

Then we created set covering problems with the following inputs :

- $\mathcal{U} = \{1, \ldots, M\}$
- $X = \{S_1, \ldots, S_N\}, S_i \subseteq \mathcal{U} (1 \leq i \leq N)$ is a collection of randomly generated subsets of $\mathcal{U}$
- $Cost(S_i) = 1 (1 \leq i \leq 1)$

We tested a CSP with only one SC constraint $SC(N, T, \mathcal{U}, X, Cost)$ and try to find the optimal value N (thus computing SC^*) by branch and bound. The three relaxations were used : $2SC$, SCL^* and MD. A naive branching strategy was used ; it branches on $S_1, S_2, \ldots$, always in this order. This is not efficient at all to solve set covering problems, but it allows to be sure that the three different propagators explore the search tree in the same manner. This removes any interaction between propagators and the search strategy that could lead to ambiguous results. Problems with at least one element not covered by any subset were deleted from the table, because they are not interesting for the comparison. These three relaxations were implemented as propagators in Gecode [9]. The SCL^* propagator used the library lp_solve 5.5 [18] to solve the linear relaxation. This library implements the simplex algorithm which allows incremental propagation along the search tree. Because these problems are pure set covering problems (without any side constraints), all methods perform worse when the shaving process is enabled. So we disabled the shaving in order to be more fair when comparing all methods. However, when there are a lot of side constraints, propagators with small time per call should be advantaged (MD and $2SC$).

Neither method outperforms uniformly the two others. However some general observations can be observed. MD performs usualy better when the value $\frac{M}{N}$ is small. This could be explained by the fact that such problems have a lot of solutions. Then there is less pruning to do and because MD is faster to compute the lower bound (even of worse quality), it can explore more solutions.

On the other hand, SCL^* performs much better when this value is higher ($\frac{M}{N} \geq 1$). In such problems there are less solutions and the good quality of the lower bound computed by SCL^* is worth its computational cost.

$2SC$ seems to be situated between MD and SCL^*. When MD performs best, $2SC$ usually achieves a better performance than SCL^*. In the opposite, when SCL^* performs best, $2SC$ is often better than MD. This is due to the fact that for $2SC$, we relax more the original set covering problem than for SCL^*. But MD is even more relaxed. Its computational cost is also situated between both of them. $2SC$ seems to be more stable in function of the input problem. Because $2SC$ decomposes each subsets in 2-sets, $2SC$ performs better when the sizes of the subsets are small.

The advantage of the incremental behaviour of the algorithm $2SC$ is demonstrated by Table-1 where we can observe that the overall time per call to the propagator $2SC$ is often smaller than the one for the SCL^* propagator which is also incremental.

M	**N**	S^-	S^+	$2SC$				SCL^*				MD			
				Sol	Time	failures	TC	Sol	Time	failures	TC	Sol	Time	failures	TC
10	500	2	6	223	> 30	38342	0,38	222	> 30	38760	0,37	**2**	**12,4**	263731	0,02
10	500	2	10	303	> 30	19418	0,73	235	> 30	35103	0,41	**1**	**8,46**	127222	0,03
10	200	2	6	**2**	4,09	19507	0,1	**2**	7,46	19507	0,18	**2**	**1,66**	45025	0,02
10	200	2	10	**1**	6,42	19703	0,16	**1**	8,11	19703	0,2	**1**	**1,18**	20875	0,03
20	200	2	4	**5**	**6,83**	55259	0,06	**5**	10,16	18946	0,26	9	> 30	771069	0,02
20	200	2	6	**4**	**4,5**	21706	0,1	**4**	10,4	19132	0,26	5	> 30	619822	0,02
20	200	2	10	**3**	**8,64**	22412	0,19	**3**	9,64	19317	0,24	**3**	> 30	398993	0,04
20	200	4	14	**2**	12,78	19507	0,32	**2**	12,28	19507	0,3	**2**	**8,97**	49310	0,09
20	200	8	10	**3**	15,27	25454	0,29	**3**	**9,57**	19407	0,24	**3**	> 30	266804	0,05
20	200	8	14	**2**	16,11	20092	0,39	**2**	**10,33**	19507	0,25	**2**	13,13	102288	0,06
50	500	2	4	141	> 30	64416	0,23	228	> 30	37053	0,39	**39**	> 30	551144	0,03
50	500	2	6	233	> 30	35513	0,41	236	> 30	34842	0,42	**28**	> 30	446423	0,03
50	500	2	10	314	> 30	17205	0,82	242	> 30	33228	0,44	**19**	> 30	318013	0,05
50	500	4	14	370	> 30	8405	1,62	260	> 30	28770	0,5	**12**	> 30	226824	0,07
50	500	8	10	365	> 30	9062	1,51	266	> 30	27381	0,53	**15**	> 30	212295	0,07
50	500	8	14	388	> 30	6259	2,12	266	> 30	27442	0,52	**11**	> 30	174868	0,08
10	50	2	6	**2**	**0,1**	1132	0,04	**2**	0,25	1132	0,1	**2**	**0,1**	3153	0,01
10	50	2	10	**1**	0,14	1178	0,05	**1**	0,26	1178	0,1	**1**	**0,09**	1446	0,03
100	500	2	4	**138**	> 30	67100	0,22	236	> 30	34938	0,41	199	> 30	511838	0,03
100	500	2	6	231	> 30	36210	0,4	244	> 30	32772	0,44	**104**	> 30	382419	0,04
100	500	2	10	312	> 30	17646	0,8	254	> 30	30312	0,48	**94**	> 30	269540	0,05
100	500	4	14	372	> 30	8227	1,65	269	> 30	26604	0,54	**45**	> 30	167095	0,09
100	500	8	10	364	> 30	9201	1,49	268	> 30	26897	0,53	**62**	> 30	176621	0,08
100	500	8	14	388	> 30	6257	2,12	277	> 30	24780	0,58	**42**	> 30	153117	0,1
50	200	2	4	24	> 30	291570	0,05	**15**	**13,29**	18405	0,35	36	> 30	749486	0,02
50	200	2	6	22	> 30	215045	0,07	**11**	**22,75**	20429	0,54	24	> 30	500720	0,03
50	200	2	10	11	> 30	126201	0,12	**7**	**12,53**	19119	0,32	15	> 30	393054	0,04
50	200	4	14	7	> 30	58297	0,25	**6**	**25,2**	21042	0,58	10	> 30	203992	0,07
50	200	8	10	10	> 30	62392	0,23	**7**	> 30	20723	0,7	12	> 30	243080	0,06
50	200	8	14	7	> 30	49796	0,29	**6**	> 30	20909	0,69	8	> 30	159690	0,09
400	1000	2	4	**791**	> 30	21935	0,64	862	> 30	9544	1,38	980	> 30	117592	0,1
400	1000	2	6	**836**	> 30	13515	1	876	> 30	7743	1,66	964	> 30	128167	0,09
400	1000	2	10	**885**	> 30	6651	1,88	**885**	> 30	6640	1,89	958	> 30	88089	0,14
400	1000	4	14	920	> 30	3179	3,41	**897**	> 30	5277	2,29	930	> 30	61146	0,22
400	1000	8	10	918	> 30	3330	3,3	**913**	> 30	3769	3	940	> 30	64338	0,2
400	1000	8	14	936	> 30	2029	4,69	**906**	> 30	4393	2,66	934	> 30	49467	0,27
400	500	2	10	385	> 30	15628	0,82	**318**	> 30	16981	0,84	462	> 30	121025	0,1
400	500	4	14	368	> 30	8755	1,56	**325**	> 30	15231	0,93	462	> 30	89955	0,14
400	500	8	10	364	> 30	9265	1,48	**336**	> 30	13385	1,05	463	> 30	73104	0,17
400	500	8	14	389	> 30	6164	2,15	**340**	> 30	12844	1,09	446	> 30	55121	0,23
50	50	2	6	15	> 30	321407	0,04	**14**	**0,28**	732	0,17	21	> 30	473335	0,02
50	50	2	10	10	> 30	218047	0,06	**9**	**0,37**	891	0,18	12	> 30	272193	0,04
50	50	4	14	**6**	> 30	110772	0,12	**6**	**0,41**	997	0,18	8	> 30	183474	0,06
50	50	8	10	**8**	> 30	154642	0,09	**8**	**0,59**	965	0,26	10	> 30	201884	0,06
50	50	8	14	7	> 30	129238	0,11	**6**	**0,61**	1044	0,25	7	> 30	143366	0,08
200	200	4	14	90	> 30	33609	0,42	**36**	> 30	15550	0,93	138	> 30	91178	0,15
200	200	8	14	73	> 30	30358	0,47	**33**	> 30	15642	0,93	84	> 30	89203	0,16
100	50	4	14	25	> 30	94787	0,11	**17**	**0,41**	686	0,27	23	> 30	130224	0,09
100	50	8	14	25	> 30	88691	0,12	**16**	**0,53**	757	0,31	29	> 30	95007	0,11
400	200	8	14	162	> 30	13554	0,75	**74**	> 30	10405	1,39	171	> 30	65742	0,18
50	20	4	14	**7**	0,08	471	0,06	**7**	**0,04**	98	0,15	**7**	0,14	775	0,06
50	20	8	10	**9**	0,42	2956	0,05	**9**	**0,03**	86	0,13	**9**	1,42	9487	0,06
50	20	8	14	**7**	0,47	2567	0,07	**7**	**0,05**	106	0,18	**7**	2,25	9991	0,08

TAB. 1 – Experimental results. M is the number of elements to cover(i.e. the cardinality of $\mathcal{U}$). N is the size of the collection of subsets of $\mathcal{U}$. S^+ (resp. S^-) is the maximum cardinality of subsets S_i (resp. the minimum cardinality of S_i). Sol is the smallest solution found for N. $Time$ is the total search time (search + propagation). A 30 seconds limit was used. $failures$ is the number of times the algorithm failed during search. TC is the time per call of the propagator in milliseconds.

5 Future work

Future work will explore two main directions. First, how information from the propagators can help the search strategies. Using the values of the dual variables with $2SC$ and the fractional values of the primal variables for SCL^*, we could derive interesting hints for the branching strategy.

Second, Some reduction tests exist in the litterature to prune subsets S_i which are not part of the optimal solution. Because the optimal solution of the set covering constraint could not satisfy another side constraint, such tests cannot be performed in a constraint framework. However, when computing a lower bound, we search for an optimal solution of the set covering problem (or a lower bound on it). Reduction tests could thus be done during the first propagation to temporarily remove subsets that could be neglected while computing these lower bounds. This would allow our propagators to perform faster because of the reduction in the problem size. In order not to forget any solution, the challenge is to put back the removed subsets at the right time during search.

6 Conclusions

The set covering problem is a very important problem in combinatorial optimisation. It is useful to model many real-life problems and many set and graph problems. The design of a global constraint for the set covering problem allows to easily express such problems in constraint programming languages, while providing tight bounds to efficiently solve such optimisation problems.

From the structure of the set covering problem, the best way to prune values is to compute tight bounds of a set covering problem. Unfortunately this problem is NP-Hard. In this paper we have proposed a lower bound based on the computation of a edge covering in a bipartite graph. The algorithm used is highly incremental, which allows fast computation of optimal values after a small change in the problem, making it well suited for search-tree scheme. This lower bounds is tight when the sizes of the subsets are small.

Four lower bounds were presented in [4], and two of them (SCL^* and MD) were identified as better than the others. We compared our approach to both of them. The greedy approach achieves good performances when the number of subsets is well bigger than the number of elements to cover. The linear relaxation is generally better when the number of subsets is almost the same as the number of elements to cover. $2SC$ is situated in the middle of SCL^* and MD, when the number of subsets is twice the number of elements to cover. More impotantly $2SC$ achieves better results than SCL^*, when MD is best, and performs better than MD when SCL^* is best. $2SC$ si a good choice when we have to solve problems with very small subsets. With bigger subsets, we could dynamically choose to use MD or SCL^* in function of the value $\frac{M}{N}$.

Références

[1] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *J. ACM*, 45(3) :501–555, 1998.

[2] Nicolas Beldiceanu, Mats Carlsson, and Sven Thiel. Cost-filtering algorithms for the two sides of the sum of weights of distinct values constraint. Technical Report 14, Swedish Institute of Computer Science, 2002. http ://www.sics.se/libindex.html#Technical.

[3] Claude Berge. Two theorems in graph theory. In *Proceedings of the National Academy of Sciences of the United States of America*, volume 43-9, pages 842–844, 1957.

[4] C. Bessière, E. Hebrard, B. Hnich, Z. Kiziltan, and T. Walsh. Filtering Algorithms for the NVALUE Constraint. In Roman Barták and Michela Milano, editors, *Proceedings of the 7th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR-05)*, volume 3524 of *Lecture Notes in Computer Science*, pages 79–93, Prague, Czech Republic, May 2005. Springer-Verlag.

[5] A. Caprara, M. Fischetti, and P. Toth. Algorithms for the set covering problem, 1998.

[6] Alberto Caprara, Matteo Fischetti, and Paolo Toth. A heuristic method for the set covering problem. *Oper. Res.*, 47(5) :730–743, 1999.

[7] S. Ceria, P. Nobili, and A. Sassano. Set covering problem. In M. Dell'Amico, F. Maffioli, and S. Martello, editors, *Annotated Bibliographies in Combinatorial Optimization*, chapter 23. John Wiley, 1997.

[8] Elia El-Darzi and Gautam Mitra. Solution of set-covering and set-partitioning problems using assignment relaxations. *The Journal of the Operational Research Society*, 43(5) :483–493, may 1992.

[9] Gecode : Generic constraint development environment, 2005. library. Available as an open-source from www.gecode.org.

[10] Carmen Gervet. Interval propagation to reason about sets : Definition and implementation of a

practical language. *Constraints*, 1(3) :191–244, march 1997.

[11] Karla L. Hoffman and Manfred Padberg. Solving airline crew scheduling problems by branch-and-cut. *Manage. Sci.*, 39(6) :657–682, 1993.

[12] Current J and O'Kelly M. Locating emergency warning sirens. *Decision Sciences*, 23(1) :221–234, 1992.

[13] Richard M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[14] W. Karush. Minima of functions of several variables with inequalities as side constraints. Master's thesis, Dept. of Mathematics, Univ. of Chicago, Chicago, Illinois, 1939.

[15] H. W. Kuhn and A. W. Tucker. Nonlinear programming. In *Proceedings of 2nd Berkeley Symposium*, pages 481–492. Berkeley : University of California Press, 1950.

[16] Harold W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistic Quarterly*, 2 :83–97, 1955.

[17] Nemhauser George L. and Glenn M. Weber. Optimal set partitioning, matchings and lagrangian duality. *Naval Research Logistics Quarterly*, 26 :553–563, 1979.

[18] lp_solve library.

[19] Carsten Lund and Mihalis Yannakakis. On the hardness of approximating minimization problems. *J. ACM*, 41(5) :960–981, 1994.

[20] Kurt Mehlhorn and Stefan Näher. *LEDA : a platform for combinatorial and geometric computing*. Cambridge University Press, New York, USA, 1999.

[21] François Pachet and Pierre Roy. Automatic generation of music programs. In *CP '99 : Proceedings of the 5th International Conference on Principles and Practice of Constraint Programming*, pages 331–345, London, UK, 1999. Springer-Verlag.

[22] John F. Pierce. Application of combinatorial programming to a class of all-zero-one integer programming problems. *Management Science*, 15(3) :191–209, nov 1968.

[23] J F Puget. Finite set intervals. In *Proceedings Workshop on Set Constraints, held in Conjunction with CP'96*, 1996.

[24] Jean-Charles Régin. A filtering algorithm for constraints of difference in csps. In *AAAI '94 : Proceedings of the twelfth national conference on Artificial intelligence (vol. 1)*, pages 362–367, Menlo Park, CA, USA, 1994. American Association for Artificial Intelligence.

[25] Meinolf Sellmann, Kyriakos Zervoudakis, Panagiotis Stamatopoulos, and Torsten Fahle. Crew assignment via constraint programming : Integrating column generation and heuristic tree search. *Annals of Operations Research*, 115 :207–225, September 2002.

[26] Constantine Toregas, Ralph Swain, Charles ReVelle, and Lawrence Bergman. The location of emergency service facilities. *Operations Research*, 19(19) :1363–1373, 1971.

[27] P. Turán. On an extremal problem in graph theory. *Mat. Fiz. Lapok*, 48 :436–452, 1941.

[28] W. J. van Hoeve. The alldifferent constraint : A survey, 2001.

[29] L.A. Wolsey. *Integer Programming.* Wiley, New York, 1998.