Available online at www.sciencedirect.com
ScienceDirect
Discrete Applied Mathematics 155 (2007) 312–326
DISCRETE APPLIED MATHEMATICS
www.elsevier.com/locate/dam

Solution techniques for the Large Set Covering Problem

Philippe Galinier[a], Alain Hertz[b]

[a] *Ecole Polytechnique - CRT, Département de génie informatique, CP 6079, succ. Centre-ville, Montréal, Que., Canada H3C 3A7*
[b] *Ecole Polytechnique - GERAD, Département de mathématiques et de génie industriel CP 6079, succ. Centre-ville, Montréal, Que., Canada H3C 3A7*

Received 14 July 2003; received in revised form 7 December 2005; accepted 7 April 2006
Available online 7 November 2006

Abstract

Given a finite set E and a family $F=\{E_1,\ldots,E_m\}$ of subsets of E such that F covers E, the famous unicost set covering problem (USCP) is to determine the smallest possible subset of F that also covers E. We study in this paper a variant, called the Large Set Covering Problem (LSCP), which differs from the USCP in that E and the subsets E_i are not given in extension because they are very large sets that are possibly infinite. We propose three exact algorithms for solving the LSCP. Two of them determine minimal covers, while the third one produces minimum covers. Heuristic versions of these algorithms are also proposed and analysed. We then give several procedures for the computation of a lower bound on the minimum size of a cover. We finally present algorithms for finding the largest possible subset of F that does not cover E. We also show that a particular case of the LSCP is to determine irreducible infeasible sets in inconsistent constraint satisfaction problems. All concepts presented in the paper are illustrated on the k-colouring problem which is formulated as a constraint satisfaction problem.

Keywords: Set covering; Constraint satisfaction; Irreducible infeasible sets; k-colouring

1. Introduction

Let E be a set of n elements, and let $E_1,\ldots,E_m$ be m subsets of E such that $\bigcup_{i=1}^{m} E_i = E$. The unicost set covering problem (USCP) is to determine a subset $I \subseteq \{1,\ldots,m\}$ of minimum size such that $\bigcup_{i\in I} E_i = E$. This is a famous NP-hard problem [12]. In this paper, we study a variant of the USCP, called the *Large Set Covering Problem* (LSCP), which differs from the USCP in that E and the subsets E_i are not given in extension because they may be very large, and possibly infinite sets. We assume we are given two procedures:

- Procedure IS-ELEMENT(e,i) returns value "true" if and only if $e \in E_i$. Such a procedure is essential since the subsets E_i are not given in extension and are possibly infinite.
- Given any weighting function ω that assigns a weight $\omega(i)$ to each set E_i, procedure MIN_WEIGHT(ω) returns an element $e \in E$ such that $\sum_{e\in E_i}\omega(i)$ is minimum.

A subset I of $\{1,\ldots,m\}$ such that $\bigcup_{i\in I} E_i = E$ is called a *cover*. The LSCP is to determine a minimal (inclusion wise) cover. We also consider the problem, called *minimum* LSCP, which is to determine a minimum cover.

E-mail addresses: Philippe.Galinier@polymtl.ca (P. Galinier), Alain.Hertz@gerad.ca (A. Hertz).

0166-218X/$ - see front matter © 2006 Published by Elsevier B.V.
doi:10.1016/j.dam.2006.04.043

We show in the next section that the LSCP can help in proving inconsistency of constraint satisfaction problems. As an illustration, we consider the k-colouring problem which is to colour a given graph G with at most k colours, such that any two adjacent vertices have different colours. We then describe in Section 3 two algorithms for finding minimal covers, and one for finding minimum ones. As will be shown, procedure MIN_WEIGHT typically requires the solution of NP-hard problems such as Max-CSP [10]. We analyse in Section 4 the impact of replacing MIN_WEIGHT in the three proposed algorithms by a heuristic procedure. In Section 5 we show how to compute lower bounds on the size of a minimum cover. Section 6 is devoted to a related problem consisting of determining a maximum subset I of $\{1, \ldots, m\}$ such that $\bigcup_{i \in I} E_i \neq E$. Concluding remarks are provided in the last section. All concepts and techniques described in this paper are illustrated on the k-colouring problem.

2. Finding irreducible infeasible sets in inconsistent constraint satisfaction problems

The constraint satisfaction problem (CSP) [16,21] is defined over a constraint network, which consists of a finite set of *variables,* each associated with a *domain* of values, and a set of *constraints*. A constraint specifies, for a particular subset of variables, a set of incompatible combinations of values for these variables. A *solution* of a CSP is an assignment of a value to each variable from its domain such that all constraints are satisfied. A CSP is *consistent* if it has at least one solution; otherwise it is *inconsistent* (or unsolvable, or overconstrained, or infeasible). While the CSP is to determine whether a solution exists, related problems are to find one or all solutions, and to find an optimal solution relative to a given cost function. For example, Max-CSP is the problem of determining an assignment of a value to each variable from its domain such that as many constraints as possible are satisfied. Constraint satisfaction provides a convenient way to represent and solve problems where mutually compatible values have to be assigned to a predetermined number of variables under a set of constraints. Numerous applications arise in a variety of disciplines including machine vision, belief maintenance, temporal reasoning, graph theory, circuit design and diagnostic reasoning [21].

A well-known example of a CSP is the *k-colouring problem*, where the task is to colour, if possible, a given graph G with at most k colours, such that any two adjacent vertices have different colours. If such a colouring exists, then G is called *k-colourable*. A constraint satisfaction formulation of this problem associates the vertices of the graph with variables, the set of possible colours $\{1, \ldots, k\}$ is the domain of each variable, and the disequality constraints between adjacent vertices are the constraints of the problem.

For a given CSP, a *partial* assignment is an assignment of a value from its domain to some variables, but not necessarily all. When all variables get a value, the assignment is called *complete*. We say that a partial assignment *satisfies a constraint* if it can be extended to a complete assignment that satisfies this constraint. A partial assignment is *legal* if it satisfies all constraints.

A subset S of constraints of a CSP is infeasible if no complete assignment satisfies all constraints in S simultaneously, otherwise it is feasible. Following the terminology in [2–5,20,22] a subset of constraints is called an *irreducible infeasible set* (IIS) *of constraints* if it is infeasible, but becomes feasible when any one constraint is removed. Similarly, a subset V of variables of a CSP is infeasible if there is no legal partial assignment of the variables in V, otherwise it is feasible. We define an IIS *of variables* as any subset of variables which is infeasible, but becomes feasible when any one variable is removed.

For illustration of these concepts, consider the graph represented in Fig. 1a. It is 3-colourable, but not 2-colourable. The CSP corresponding to the 2-colouring problem on this graph has $V = \{v_1, \ldots, v_7\}$ as the variable set (vertex set). To each edge (v_i, v_j), we associate a constraint denoted by (i, j) and imposing that v_i and v_j must receive different colours. Hence, the constraint set C is $\{(1, 2), (1, 3), (1, 7), (2, 3), (3, 4), (4, 5), (5, 6), (5, 7), (6, 7)\}$. This CSP has four IISs of constraints, $\{(1, 2), (1, 3), (2, 3)\}$, $\{(5, 6), (5, 7), (6, 7)\}$, $\{(1, 3), (1, 7), (3, 4), (4, 5), (5, 7)\}$ and $\{(1, 2), (1, 7), (2, 3), (3, 4), (4, 5), (5, 6), (6, 7)\}$ and three IISs of variables, $\{v_1, v_2, v_3\}$, $\{v_5, v_6, v_7\}$ and $\{v_1, v_3, v_4, v_5, v_7\}$.

Notice that if the variables of a CSP define an IIS of variables, then the constraints involving these variables do not necessarily define an IIS of constraints. As an example, consider the CSP associated with the 3-colouring problem for the graph in Fig. 1b. The vertex set V of the graph is an IIS of variables since the graph is not 3-colourable while the removal of any vertex produces a 3-colourable graph. However, the edge set is not an IIS of constraints since the removal of the edge linking v_1 to v_2 gives a graph that is still not 3-colourable.

Exhibiting an IIS of constraints or variables can be very useful in practice, especially IISs of small size. For example, when solving a timetabling problem, it often happens that there is no solution satisfying all constraints. An IIS represents

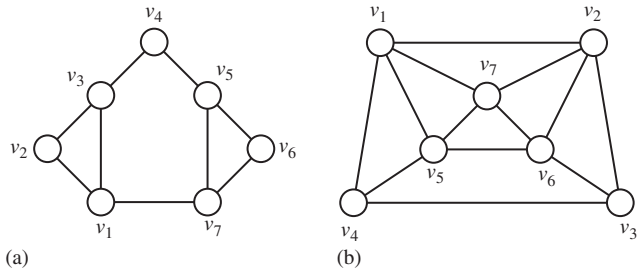

Fig. 1. Two illustrative graphs.

a part of the problem that gives a partial explanation for this infeasibility. An IIS can therefore be very useful to the person in charge of building the timetable since he gets an idea of which data should be changed in order to get a solvable problem. Determining IISs of constraints or variables can also be very helpful in proving the inconsistency of a CSP. Indeed, IISs contain typically a smaller number of constraints and variables when compared to the original problem, and a proof of inconsistency is therefore possibly easier to obtain on an IIS rather than on the original problem. To illustrate this, consider once again the k-colouring problem. Suppose that no heuristic algorithm is able to determine a k-colouring of the considered graph G. One may then suspect that G is not k-colourable. To prove it, it is sufficient to exhibit a partial subgraph G' (obtained by removing edges) or an induced subgraph G'' (obtained by removing vertices and all edges incident to these vertices) which is not k-colourable but which becomes k-colourable as soon as any edge of G' or any vertex of G'' is removed. The edges of the partial subgraph G' correspond to an IIS of constraints while the vertices of the induced subgraph G'' form an IIS of variables. If G' and G'' have fewer edges and vertices than G, then instead of proving that G is not k-colourable, it is hopefully easier to prove that G' or G'' is not k-colourable [14].

Crawford [8] and Mazure et al. [17] have designed algorithms to determine infeasible subsets of constraints for the satisfiability problem. A good review on the detection of IISs of constraints in linear programs is given in [4]. A review of the theory and history of IISs in other types of mathematical programs is given in [5].

The problem of finding IISs of constraints may look very different from the problem of finding IISs of variables. These two problems are, however, two special cases of the LSCP defined in Section 1. Indeed, given an inconsistent CSP, define as an element of E any complete assignment. To each constraint c_i $(i = 1, \ldots, m)$ of the CSP let us associate the subset E_i of E containing all assignments that violate c_i. A set of constraints is infeasible if and only if it covers E. Hence, finding an IIS of constraints is equivalent to solving the LSCP. Procedure IS-ELEMENT(e, i) simply determines if the complete assignment e violates constraint c_i. Procedure MIN_WEIGHT(ω) returns a complete assignment e that minimizes the sum of the weights of the constraints violated in e. In the case of the k-colouring problem, define a *conflicting edge* as an edge having both endpoints with the same colour. Procedure MIN_WEIGHT(ω) returns a colouring that minimizes the sum of the weights of the conflicting edges. Notice that the problem solved by the procedure MIN_WEIGHT corresponds to Max-CSP when all weights equal 1.

Similarly, given an inconsistent CSP, define as an element of E any legal partial assignment. To each variable v_i of the CSP let us associate the subset E_i of E containing all legal partial assignments in which v_i has no value (i.e., v_i is not instantiated). A subset of variables is infeasible if and only if it covers E. Hence, here again, finding an IIS of variables is equivalent to solving the LSCP. Procedure IS-ELEMENT(e, i) returns "true" if and only if variable v_i is not instantiated in the partial assignment e. Procedure MIN_WEIGHT(ω) returns a legal partial assignment e that minimizes the sum of the weights of the variables that are not instantiated in e. In the case of the k-colouring problem, procedure MIN_WEIGHT(ω) returns a partial k-colouring without conflicting edges that minimizes the sum of the weights of the uncoloured vertices.

At this point it is important to observe that procedure MIN_WEIGHT typically solves an NP-hard problem, both in the case of searching for an IIS of constraints or an IIS of variables. While Section 3 contains exact algorithms for the LSCP, Section 4 will be devoted to the analysis of the heuristic algorithms obtained by replacing procedure MIN_WEIGHT by a heuristic version.

While MIN_WEIGHT typically solves an NP-hard problem, there are some cases where MIN_WEIGHT can be implemented in an efficient way. For illustration we mention a CSP which appears in the context of an interactive decision

support problem [1]. A consistent CSP with a set C of constraints is considered, where the solutions represent the catalogue of a company, i.e., all the variants of the articles produced in that company. When configuring a product, the user of the decision support system specifies a series of additional constraints $C_1, C_2, \ldots$ concerning the features of the product he is interested in. At some iteration p, this set of additional constraints may become infeasible (i.e., $C \cup \{C_1, \ldots, C_{p-1}\}$ is feasible while $C \cup \{C_1, \ldots, C_p\}$ is not). In order to guide the user, the system should provide a minimal subset S of $\{C_1, \ldots, C_p\}$ such that $C \cup S$ is already infeasible. Set S explains why constraint C_p generates inconsistency. Thanks to sophisticated data-structures, Amilhaste et al. [1] have developed an efficient procedure that implements procedure MIN_WEIGHT: the procedure provides a solution that violates no constraints in $C \cup \{C_p\}$ and a minimum weighted subset of constraints in $\{C_1, \ldots, C_{p-1}\}$ for any weighting of the constraints. The procedure is used to determine a maximum subset F of $\{C_1, \ldots, C_{p-1}\}$ such that $C \cup F \cup \{C_p\}$ is feasible. The algorithms presented in this paper make it possible to generate explanations while this was an open problem in [1].

3. Exact solution methods

In what follows, we will use the following notations. First of all, given an element $e \in E$ and a subset $I \subseteq \{1, \ldots, m\}$, we denote by $F_I(e)$ the subset of elements $i \in I$ such that $e \in E_i$. Such a subset can easily be generated by means of the IS-ELEMENT procedure. For illustration, consider the k-colouring problem in the context of IISs of constraints, and let I be a set of edges and e a k-colouring: $F_I(e)$ is the subset of edges in I having both endpoints with the same colour. Similarly, in the context of IISs of variables, let I be a set of vertices and let e be a legal partial k-colouring: $F_I(e)$ is the subset of uncoloured vertices in I.

Given a subset $I \subseteq \{1, \ldots, m\}$, we consider procedure COVER(I) that returns the value "true" if $\bigcup_{i \in I} E_i = E$, and "false" otherwise. For example, in the context of IISs of variables for the k-colouring problem, COVER(I) returns the value "false" if and only if the subgraph induced by the set I of vertices is k-colourable. The output of COVER(I) can be computed as follows. Consider the weighting function ω that assigns a weight $\omega(i) = 1$ to each index $i \in I$, and a weight $\omega(i) = 0$ to the other indices, and let e be the output of MIN_WEIGHT(ω). Then COVER(I) returns the value "true" if and only if $F_I(e) \neq \emptyset$.

Now let I and J be two disjoint subsets of $\{1, \ldots, m\}$. We denote by MIN(I, J) the procedure that produces an element $e \in E$ that minimizes $M|F_I(e)| + |F_J(e)|$, where M is any number larger than $|J|$. The output OF MIN(I, J) can be obtained by applying procedure MIN_WEIGHT(ω) with the weighting function ω such that $\omega(i) = 0$ if $i \notin I \cup J$, $\omega(i) = 1$ if $i \in J$, and a $\omega(i) = M$ if $i \in I$. Notice that I is a cover if and only if the output e of MIN(I, J) is such that $F_I(e) \neq \emptyset$ for any J.

In the context of IISs of constraints for a CSP, MIN(I, J) determines a complete assignment that satisfies as many constraints as possible in I, and among such assignments, one that violates as few constraints in J as possible. Constraints in I are therefore considered as hard while those in J are soft ones. For example, when considering the k-colouring problem, let I be a subset of edges such that the partial subgraph containing only these edges is k-colourable, and let J be any subset of edges disjoint from I. Then MIN(I, J) determines a k-colouring that contains no conflicting edge from I, and among these k-colourings, one that has as few conflicting edges as possible from J.

Similarly, in the context of the search of an IIS of variables for a CSP, let I and J be two disjoint subsets of variables. The task of MIN(I, J) is to determine a partial legal assignment in which as many variables as possible in I are instantiated, and among such partial assignments, the algorithm produces one that contains as few non instantiated variables of J as possible. For the k-colouring problem, I and J correspond to two disjoint subsets of vertices. If the subgraph induced by I is k-colourable, then MIN(I, J) determines a partial k-colouring without conflicting edges such that all vertices in I are coloured, and among these partial k-colourings, the algorithm produces one with the smallest possible number of uncoloured vertices from J.

We now describe three exact algorithms for solving the (minimum) LSCP. The first two algorithms, called REMOVAL and INSERTION, generate minimal covers, while the third one, called HITTING-SET, determines minimum covers.

3.1. The REMOVAL algorithm

The REMOVAL algorithm is probably the most intuitive among the three proposed approaches. It has already been proposed by several authors, for example by Chinneck and Dravnieks [7] (where it is called *deletion filter*), for infeasible

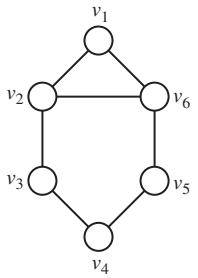

Fig. 2. A graph with no feasible 2-colouring.

linear programs and by Herrmann and Hertz [14] for graphs that are not k-colourable. The algorithm determines a minimal cover in m steps and works as follows:

Algorithm REMOVAL
Input: a cover $\{1, \ldots, m\}$
Output: a minimal cover $I \subseteq \{1, \ldots, m\}$
1. Set $I := \{1, \ldots, m\}$;
2. For $i = 1$ to m do
   If COVER$(I - \{i\}) =$ " true" then set $I := I - \{i\}$.

Property 1. *Algorithm* REMOVAL *produces a minimal cover.*

Proof. It follows from Step 2 that the output I is a cover. Now, let i be any element of the output I, and let $I_i = (I \cap \{1, \ldots, i\}) \cup \{i + 1, \ldots, m\}$. We know from Step 2 that $I_i - \{i\}$ is not a cover. Since $I \subseteq I_i$, we conclude that $I - \{i\}$ is not a cover. □

Notice that the output of this algorithm depends on the ordering of the sets E_i. Observe also that any existing minimal cover can be produced by this algorithm since if $I = \{i, i + 1, \ldots, m\}$ is a minimal cover, then the output of REMOVAL will be this subset I.

The REMOVAL algorithm is now illustrated on a 2-colouring problem for the graph represented in Fig. 2, with vertex set $V = \{v_1, \ldots, v_6\}$ and with constraint set $\{(1, 2), (1, 6), (2, 3), (2, 6), (3, 4), (4, 5), (5, 6)\}$ (we use the same notations as in Section 2). There are two IISs of constraints, $\{(1, 2), (1, 6), (2, 6)\}$ and $\{(2, 3), (2, 6), (3, 4), (4, 5), (5, 6)\}$ and two IISs of variables, $\{v_1, v_2, v_6\}$ and $\{v_2, v_3, v_4, v_5, v_6\}$.

In the context of an IIS of constraints, assume that REMOVAL considers the constraints in the lexicographical order $(1, 2)$, $(1, 6)$, $(2, 3)$, $(2, 6)$, $(3, 4)$, $(4, 5)$ and $(5, 6)$. Constraint $(1, 2)$ is first removed from I since the graph obtained by removing the edge (v_1, v_2) is still not 2-colourable. For the same reason, constraint (1,6) is removed from I. Then, no additional constraint can be removed from I since one would get a 2-colourable graph. The algorithm therefore produces the IIS of constraints $I = \{(2, 3), (2, 6), (3, 4), (4, 5), (5, 6)\}$. The output of the algorithm depends on the ordering of the constraints. For example, if the first constraint considered by REMOVAL belongs to $\{(2, 3), (3, 4), (4, 5), (5, 6)\}$, then it is easy to check that the output will be the IIS of constraints $I = \{(1, 2), (1, 6), (2, 6)\}$ which is minimum.

In the context of an IIS of variables, assume that the vertices are considered in the order $v_1, \ldots, v_6$. Vertex v_1 is first removed from I since the subgraph induced by vertices $v_2, \ldots, v_6$ is still not 2-colourable. Then, no additional vertex can be removed from I since one would get a 2-colourable graph. The algorithm therefore stops with the IIS of variables $I = \{v_2, v_3, v_4, v_5, v_6\}$. Again, the output depends on the ordering of the variables. For example, if the first vertex considered by REMOVAL belongs to $\{v_3, v_4, v_5\}$, then the output will be $I = \{v_1, v_2, v_6\}$ which is a minimum IIS of variables. Herrmann and Hertz [14] and Desrosiers et al. [9] have implemented the REMOVAL algorithm for the k-colouring problem in the context of IISs of variables. They have analysed the impact of the ordering of the variables on the size of the output.

3.2. The INSERTION algorithm

The INSERTION algorithm builds a minimal cover by adding exactly one element at each iteration. The algorithm works as follows:

Algorithm INSERTION
Input: a cover $\{1, \ldots, m\}$
Output: a minimal cover I_i
1. Set I_0:=$\emptyset$, $J_0 := \{1, \ldots, m\}$ and $i := 0$;
2. Set $e_i := \text{MIN}(I_i, J_i)$;
3. If $F_{Ii}(e_i) \neq \emptyset$ then STOP: I_i is a minimal cover;
4. Choose h_i in $F_{Ji}(e_i)$, set $I_{i+1} := I_i \cup \{h_i\}$, $J_{i+1} := J_i - F_{Ji}(e_i)$, $i := i + 1$ and go to Step 2.

Property 2. *Algorithm* INSERTION *produces a minimal cover.*

Proof. We first show that $I_i \cup J_i$ is a cover for each index i. This is trivially true for $i = 0$ since $I_0 \cup J_0 = \{1, \ldots, m\}$. So assume that $I_i \cup J_i$ is a cover. If I_i is a cover then the algorithm stops at Step 3. Otherwise, $\text{MIN}(I_i, J_i)$ produces an element e_i that minimizes $|F_{Ji}(e_i)|$ among those with $F_{Ii}(e_i) = \emptyset$. If $H = I_{i+1} \cup J_{i+1}$ is not a cover, then there exists an element $e \in E$ that does not belong to $\bigcup_{i \in H} E_i$. Since $I_i \subset H$ we know that $F_{Ii}(e) = \emptyset$. Moreover, since $H = I_i \cup J_i - (F_{Ji}(e_i) - \{h_i\})$ we know that $|F_{Ji}(e)| \leqslant |F_{Ji}(e_i) - \{h_i\}| = |F_{Ji}(e_i)| - 1$. This contradicts the optimality of e_i and we conclude that $H = I_{i+1} \cup J_{i+1}$ is also a cover.

Notice that J_{i+1} is strictly included in J_i, and that I_i is necessarily a cover when J_i is empty (because $I_i \cup J_i$ is a cover). This proves that the algorithm is finite, and it necessarily stops at Step 3 with cover I_i.

We now prove that the output I_i is a minimal cover. Consider any index $j < i$. By construction we know that $F_{Ij}(e_j) = \emptyset$. Moreover, since $h_{j+1}, \ldots, h_i$ belong to I_i, we know that they do not belong to $F_{Jj}(e_j)$ (else they would have been removed from J_j and could therefore not belong to I_i). Hence, e_j does not belong to any subset E_r with $r \in I_j \cup \{h_{j+1}, \ldots, h_i\} = I_i - \{h_j\}$, which means that $I_i - \{h_j\}$ is not a cover. □

The INSERTION algorithm has similarities with the *elastic filter* proposed by Chinneck and Dravnieks in [7]. Notice, however, that instead of using MIN_WEIGHT, the *elastic filter* uses the concept of "elastic programming" to determine an element $e_i \in E$ at Step 2. As a consequence, *all* elements (not only one) of $F_{Ji}(e_i)$ must be moved into I_i at Step 4 to ensure that the output contains an IIS. The output of the *elastic filter* is therefore a cover that is not necessarily minimal. The INSERTION algorithm looks much simpler since, by setting e_i equal to $\text{MIN}(I_i, J_i)$, we have shown in the above proof that it is sufficient to move only one element of $F_{Ji}(e_i)$ into I_i to guarantee that the output is a minimal cover. The INSERTION algorithm has similarities also with the *additive algorithm* introduced by Tamiz et al. [20]. Notice, however, that the *additive algorithm* produces an output of size k after $O(mk)$ calls to COVER while such an output is produced by INSERTION with only $O(k)$ calls to MIN_WEIGHT.

Here again, the output depends on the strategy used to choose h_i at Step 4. Algorithms INSERTION is now illustrated in the context of an IIS of constraints for the 2-colouring problem on the graph in Fig. 2. The MIN algorithm (which is an exact procedure) first determines a 2-colouring e_0 that violates constraint (2, 6). Hence, $I_1 = \{(2, 6)\}$ and $J_1 = \{(1, 2), (1, 6), (2, 3), (3, 4), (4, 5), (5, 6)\}$. Then, the best 2-colouring e_1 that does not violate constraint (2, 6) necessarily violates two constraints, one in $\{(1, 2), (1, 6)\}$ and one in $\{(2, 3), (3, 4), (4, 5), (5, 6)\}$, say (1, 2) and (2, 3). One of these violated constraints (denoted h_1) is added to I_1 to get I_2. If (2, 3) is added to I_1, it is not difficult to see that the algorithm ends with the IIS $I_5 = \{(2, 3), (2, 6), (3, 4), (4, 5), (5, 6)\}$. If the violated constraint added to I_1 is (1, 2), then the algorithm produces the IIS $I_3 = \{(1, 2), (1, 6), (2, 6)\}$ of constraints which is minimum.

In the context of an IIS of variables, algorithm INSERTION first determines a partial colouring e_0 where all vertices are coloured, except v_2 or v_6. Without loss of generality, we may assume $I_1 = \{v_2\}$. Since v_2 must now be coloured, procedure MIN produces a partial colouring e_1 where v_6 is not coloured. Hence, $I_2 = \{v_2, v_6\}$ and $J_2 = \{v_1, v_3, v_4, v_5\}$. Since v_2 and v_6 must now be coloured, procedure MIN produces a partial colouring e_2 where v_1 and one vertex in $A = \{v_3, v_4, v_5\}$ are not coloured. If a vertex of A is added to I_2, then the algorithm ends with the IIS $I_5 = \{v_2, v_3, v_4, v_5, v_6\}$, else it ends with the minimum IIS $I_3 = \{v_1, v_2, v_6\}$.

Notice that there may exist minimal covers that algorithm INSERTION cannot produce. To illustrate this, consider the 2-colouring problem on the graph in Fig. 3, in the context of an IIS of constraints. Algorithm MIN first determines a

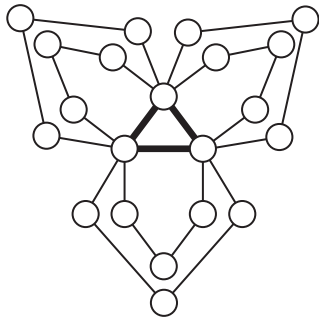

Fig. 3. A graph with no feasible 2-colouring and a unique minimum IIS of constraints.

2-colouring e_0 with three conflicting edges represented with bold lines. One of these edges defines I_1 while the others will not appear in any $I_i \cup J_i$ with $i \geqslant 1$. Therefore, the output of algorithm INSERTION cannot be the middle triangle which is the unique minimum IIS of constraints.

3.3. The HITTING-SET algorithm

The HITTING-SET algorithm determines a minimum cover and is based on the following idea. Assume as usual that $I = \{1, \ldots, m\}$ is a cover, and let $e_1, \ldots, e_p$ be p elements of E. Then all minimum covers must contain at least one element in each $F_I(e_i)$ since $I - F_I(e_i)$ is not a cover. Given p finite sets $A_1, \ldots, A_p$, we denote by BEST_HS$(A_1, \ldots, A_p)$ a procedure that determines a smallest possible subset of $A_1 \cup \cdots \cup A_p$ that intersects each A_i. Notice that BEST_HS solves the hitting set problem which is known to be NP-hard [12].

Algorithm HITTING-SET
Input: a cover $I = \{1, \ldots, m\}$
Output: a minimum cover
1. Set $I_0 := \emptyset$ and $i := 0$;
2. Set $e_i := \text{MIN}(I_i, I - I_i)$;
3. If $F_{Ii}(e_i) \neq \emptyset$ then STOP: I_i is a minimum cover;
4. Set $i := i + 1$, $I_i :=$ BEST_HS$(F_I(e_0), \ldots, F_I(e_{i-1}))$, and go to Step 2.

Property 3. *Algorithm* HITTING-SET *produces a minimum cover.*

Proof. Consider two sets I_r and I_s with $s > r$. Notice first that $F_{Ir}(e_r) = \emptyset$, else the algorithm would have stopped with the output I_r. Hence, we know that $F_I(e_r) \cap I_r = \emptyset$. Moreover, by construction, $F_I(e_r) \cap I_s \neq \emptyset$, which means that $I_r \neq I_s$. We can conclude that the algorithm is finite since there are a finite number of subsets of I. Since the algorithm can only stop at Step 3, we know that the output I_i is a cover. Notice finally that each cover necessarily intersects all $F_I(e_r)$ $(r = 1, \ldots, i-1)$ since $I - F_I(e_r)$ is not a cover. Hence, since I_i is the output of BEST_HS$(F_I(e_0), \ldots, F_I(e_{i-1}))$, it is a minimum cover. □

While HITTING-SET is a finite algorithm, its number of iterations can be exponential in m. We show in the next section that procedure HITTING-SET can be stopped at any time to produce a lower bound on the size of a minimum cover. It is now illustrated in the context of an IIS of constraints for the 2-colouring problem on the graph in Fig. 2. We first describe a kind of worst case scenario for HITTING-SET, where the minimum IIS of constraints is only obtained after having determined 7 different k-colourings. The first 2-colouring e_0 will only violate $(2, 6)$. Hence, $I_1 = \{(2, 6)\}$. Then, the best 2-colouring e_1 that does not violate constraint $(2, 6)$ necessarily violates two constraints, one in $\{(1, 2), (1, 6)\}$, say $(1, 2)$, and one in $\{(2, 3), (3, 4), (4, 5), (5, 6)\}$, say $(2, 3)$. The output I_2 of BEST_HS can therefore be $\{(1, 2), (2, 6)\}$. Then, the best 2-colouring e_2 that does not violate $(1, 2)$ and $(2, 6)$ necessarily violates $(1, 6)$ and one constraint in $\{(2, 3), (3, 4), (4, 5), (5, 6)\}$, say $(2, 3)$. In such a case, BEST_HS necessarily produces $I_3 = \{(2, 3), (2, 6)\}$. It may then happen that e_3 only violates $(1, 2)$ and $(3, 4)$, in which case I_4 is possibly equal to $\{(1, 2), (2, 3), (2, 6)\}$. The next 2-colouring e_4 can then violate $(1, 6)$ and $(3, 4)$, in which case BEST_HS can produce $I_5 = \{(2, 3), (2, 6), (3, 4)\}$.

Assume then that e_5 violates (1, 2) and (4, 5). The smallest hitting set I_6 produced by BEST_HS is then necessarily equal to {(1, 2), (1, 6), (2, 6)} which is an infeasible set of constraints. The best 2-colouring e_6 violates one constraint in I_6 and the algorithm therefore stops.

The same algorithm can also determine this minimum IIS of constraints with only 4 different assignments. Indeed, as observed above, e_0 necessarily violates constraint (2, 6). Assume that e_1 violates (1, 2) and (2, 3) and that BEST_HS produces the output $I_2 = \{(1, 2), (2, 6)\}$. Then e_2 can violate (1, 6) and (3, 4), in which case BEST_HS can produce the output $I_3 = \{(1, 2), (1, 6), (2, 6)\}$ which is a minimum IIS of constraints (and the next 2-colouring e_3 will violate one constraint in I_3).

In the context of an IIS of variables, algorithm HITTING-SET first determines a partial colouring e_0 where all vertices are coloured, except v_2 or v_6. Without loss of generality, we may assume that $I_1 = \{v_2\}$. Procedure MIN then necessarily determines a partial colouring e_1 where v_6 is not coloured. Hence, $I_2 = \{v_2, v_6\}$. Now, procedure MIN will find a partial colouring e_2 where v_1 and one vertex in $\{v_3, v_4, v_5\}$, say v_3, is not coloured. Procedure BEST_HS can then either produce the output $I_3 = \{v_1, v_2, v_6\}$ which is a minimum IIS of variables, or the set $I_3 = \{v_2, v_3, v_6\}$. In the later case, MIN will find a partial colouring e_3 where v_1 and one vertex in $\{v_4, v_5\}$ is not coloured, and BEST_HS will necessarily produce the minimum IIS of variables $I_4 = \{v_1, v_2, v_6\}$ as output.

4. Heuristic algorithms

Algorithms REMOVAL, INSERTION and HITTING-SET are typically difficult to implement in practice. Indeed, algorithm REMOVAL must determine in Step 2 whether a given subset I of $\{1, \ldots, m\}$ is a cover. This problem is NP-complete since a special case is to determine if a given CSP is consistent. Notice also that algorithms INSERTION and HITTING-SET call procedure MIN at Step 2, while this procedure solves an NP-hard problem such as Max-CSP. Notice finally that procedure BEST_HS called at Step 4 of HITTING-SET solves the hitting set problem which is also a famous NP-hard problem.

We now analyse the impact of using heuristic algorithms instead of exact ones for solving these NP-hard problems. Let HMIN_WEIGHT(ω) be a heuristic procedure that aims to determine an element $e \in E$ such that $\sum_{e \in E_i} \omega(i)$ is minimum. HMIN_WEIGHT can be implemented by using neighbourhood search techniques. For example, Galinier and Hao [11] have designed a tabu search algorithm that corresponds to HMIN_WEIGHT in the context of IISs of constraints for a CSP. Also, the algorithms developed by Hertz and de Werra [15] and by Morgenstern [18] correspond to HMIN_WEIGHT in the context of IISs of constraints and variables, respectively, for the k-colouring problem.

We denote by HCOVER and HMIN the heuristic versions of COVER and MIN obtained by replacing MIN_WEIGHT by HMIN_WEIGHT. More precisely, given two disjoint subsets I and J of $\{1, \ldots, m\}$, we denote by HMIN(I, J) the output of HMIN_WEIGHT(ω) where $\omega(i) = 0$ if $i \notin I \cup J$, $\omega(i) = 1$ if $i \in J$, and a $\omega(i) = M$ if $i \in I$ (where M is any number larger than $|J|$). Also, let I be a subset of $\{1, \ldots, m\}$. Consider the weighting function ω that assigns a weight $\omega(i) = 1$ to each index $i \in I$, and a weight $\omega(i) = 0$ to the other indices, and let e be the output of HMIN_WEIGHT(ω). We denote by HCOVER(I) the procedure that returns the value "true" if $F_I(e) \neq \emptyset$, and "false" otherwise. Observe that when the output of HCOVER(I) is "false" then we know that I is not a cover since HMIN_WEIGHT has exhibited an element that does not belong to $\bigcup_{i \in I} E_i$. However, when the output of HCOVER(I) is "true", we have no guarantee that I is a cover. The heuristic version HREMOVAL of REMOVAL is simply obtained by replacing COVER by HCOVER at Step 2.

Algorithm HREMOVAL
Input: a cover $\{1, \ldots, m\}$
Output: a set I that is possibly a cover
1. Set $I := \{1, \ldots, m\}$;
2. For $i = 1$ to m do
   If HCOVER($I - \{i\}$) = " true" then set $I := I - \{i\}$.

Property 4. *If the output of* HREMOVAL *is a cover, then it is a minimal cover.*

Proof. Assume that the output I is a cover. Consider any $i \in I$, and let $I_i = (I \cap \{1, \ldots, i\}) \cup \{i + 1, \ldots, m\}$. We know from Step 2 that $I_i - \{i\}$ is not a cover since i was not removed from I and HCOVER($I - \{i\}$) returns value "false"

only if $I - \{i\}$ is not a cover. Since $I \subseteq I_i$, we conclude that $I - \{i\}$ is not a cover, which means that I is a minimal cover. □

Notice that if HCOVER does not always recognize a cover, then it may happen that the output of HREMOVAL is not a cover. For example, consider again the 2-colouring problem for the graph in Fig. 2, in the context of an IIS of constraints. Assume that we first try to remove constraint (2, 6). If HCOVER is not able to recognize that $I - \{(2, 6)\}$ is 2-colourable, then HREMOVAL will remove the edge linking v_2 to v_6, and the remaining graph is 2-colourable, which means that the output will be a feasible set of constraints.

The situation is different for algorithm HINSERTION which cannot be obtained by simply replacing MIN by its heuristic version HMIN. Indeed, while we have proved for algorithm INSERTION that $I_i \cup J_i$ is a cover for each index i, the use of HMIN may lead to a situation where $I_i \cup J_i$ is not a cover, even if the algorithm always recognizes covers. For illustration, consider again the example of Fig. 2 for an IIS of constraints, and assume that HMIN produces as output a 2-colouring where vertices v_1, v_3 and v_4 have colour 1 and vertices v_2, v_5 and v_6 have colour 2. This 2-coloring violates constraints (2, 6), (3, 4) and (5, 6). If (3, 4) or (5, 6) is chosen to be included in I_1, then constraint (2, 6) will not belong to any I_i ($i > 1$). Since $I - \{(2, 6)\}$ is feasible, the algorithm cannot produce an infeasible set of constraints (i.e., a cover). When the output e_i of HMIN(I_i, J_i) is such that $F_{Ii \cup Ji}(e_i) = \emptyset$, we know that an error occurred and we therefore have to stop the algorithm. A repair process is described at the end of this section. The heuristic version HINSERTION of INSERTION is described below.

Algorithm HINSERTION
Input: a cover $\{1, \ldots, m\}$
Output: a subset of $\{1, \ldots, m\}$ or an error message
1. Set $I_0 := \emptyset$, $J_0 := \{1, \ldots, m\}$ and $i := 0$;
2. Set $e_i :=$ HMIN(I_i, J_i); if $F_{Ii}(e_i) \neq \emptyset$ then STOP: I_i is possibly a cover;
3. If $F_{Ji}(e_i) = \emptyset$ then STOP: an error occurred since $I_i \cup J_i$ is not a cover;
4. Choose h_i in $F_{Ji}(e_i)$, set $I_{i+1} := I_i \cup \{h_i\}$, $J_{i+1} := J_i - F_{Ji}(e_i)$, $i := i + 1$ and go to Step 2.

Property 5. *If the output of* HINSERTION *is a cover, then it is a minimal cover.*

Proof. Assume that the output I_i of HINSERTION is a cover and consider any index $j < i$. By construction we known that $F_{Ij}(e_j) = \emptyset$. Moreover, since $h_{j+1}, \ldots, h_i$ belong to I_i, we know that they do not belong to $F_{Jj}(e_j)$ (else they would have been removed form J_j and could therefore not belong to I_i). Hence, e_j does not belong to any subset E_r with $r \in I_j \cup \{h_{j+1}, \ldots, h_i\} = I_i - \{h_j\}$, and this means that $I_i - \{h_j\}$ is not a cover. □

In order to get a heuristic version of algorithm HITTING-SET, we use a heuristic procedure called HBEST_HS $(A_1, \ldots, A_p)$ that produces a minimal (inclusion wise) subset of $A_1 \cup \cdots \cup A_p$ that intersects each A_i.

Algorithm HHITTING-SET
Input: a cover $I = \{1, \ldots, m\}$
Output: a set I_i that is possibly a cover
1. Set $I_0 := \emptyset$ and $i := 0$;
2. Set $e_i :=$ HMIN$(I_i, I - I_i)$;);
3. If $F_{Ii}(e_i) \neq \emptyset$ then STOP: I_i is possibly a cover;
4. Set $i := i + 1$, $I_i :=$ HBEST_HS$(F_I(e_0), \ldots, F_I(e_{i-1}))$, and go to Step 2.

Notice that HHITTING-SET is a finite algorithm, since the first part of the proof of Property 3 is still valid. If the output I_i is not a cover, this means that HMIN has not been able to produce an element e_i with $F_{Ii}(e_i) = \emptyset$, while such an element exists. We now prove that if I_i is a cover, then it is minimal.

Property 6. *If the output of* HHITTING-SET *is a cover, then it is a minimal cover.*

Proof. Assume that the algorithm stops with a cover I_i, and consider any $j \in I_i$. Since HBEST_HS produces a minimal (inclusion wise) solution, we know that $I_i - \{j\}$ does not intersect some $F_I(e_r)$ ($r \in \{0, 1, \ldots, i - 1\}$). Hence, $I_i - \{j\}$ is a subset of $I - F_I(e_r)$ which is not a cover. □

Notice that if HHITTING-SET uses the exact algorithm BEST_HS instead of the heuristic version HBEST_HS, then it either produces a subset of $\{1, \ldots, m\}$ that is not a cover (which means that HMIN has not been able to detect that some I_i was not a cover), or a minimum cover (i.e., an optimal solution to the minimum LSCP). In summary, algorithms HREMOVAL, HINSERTION and HHITTING-SET produce a subset of $\{1, \ldots, m\}$ that is either not a cover or a minimal cover. When the output of one of the above algorithms is not a cover, one can use the following repair procedure.

Algorithm REPAIR
Input: a subset $I \subseteq \{1, \ldots, m\}$ that is not a cover
Output: a minimal cover
1. If COVER$(I) =$ " true" then go to Step 3;
2. Choose an element i in $\{1, \ldots, m\} - I$, add it to I and go to Step 1;
3. Call REMOVAL INSERTION or HITTING-SET with I as input.

Step 1 of the above algorithm calls procedure COVER. If the output I of HREMOVAL, HINSERTION or HHITTING-SET has a smaller size than the original set $\{1, \ldots, m\}$, then it is typically easier to solve COVER(I) than COVER$(\{1, \ldots, m\})$. Similarly, the call to REMOVAL, INSERTION or HITTING-SET at Step 3 is typically easier with input I than with $\{1, \ldots, m\}$. Otherwise, one can replace the calls to COVER, REMOVAL, INSERTION or HITTING-SET by calls to HCOVER, HREMOVAL, HINSERTION or HHITTING-SET (but there is then no guarantee that the output is a cover). Notice also that we do not indicate how to choose i at Step 2. Experiments reported in [9] demonstrate that the strategy used when doing such a choice may have a strong influence on the performance of the detection algorithms.

5. Lower bounds

In this section, we describe several procedures for the computation of a lower bound on the size of a minimum cover. A first bound can be simply obtained by stopping algorithm HITTING-SET before the end. The size of the current I_i (when the algorithm is stopped) is a lower bound on the size of a minimum cover. Indeed, a minimum cover necessarily intersects all $F_I(e_r)$ $(r = 1, \ldots, i-1)$ since $I - F_I(e_r)$ is not a cover, while I_i is the smallest possible set that intersects all these $F_I(e_r)$.

Another lower bound can be obtained by slightly modifying algorithm INSERTION. More precisely, instead of adding only one element $h_i \in F_{Ji}(e_i)$ to I_i at Step 4, we add all of them (as in the *elastic filter* [7]). As established below, this ensures that at least one element of each cover is added to I_i at each iteration.

Procedure LOWER_BOUND_1
Input: a cover $\{1, \ldots, m\}$
Output: a lower bound on the size of a minimum cover
1. Set $I_0 := \emptyset$, $J_0 := \{1, \ldots, m\}$ and $i := 0$;
2. Set $e_i :=$ MIN(I_i, J_i);
3. If $F_{Ii}(e_i) \neq \emptyset$ then STOP: i is a lower bound on the size of a minimum cover;
4. Set $I_{i+1} := I_i \cup F_{Ji}(e_i)$, $J_{i+1} := J_i - F_{Ji}(e_i)$, $i := i + 1$ and go to Step 2.

Property 7. *Procedure* LOWER_BOUND_1 *produces a lower bound on the size of a minimum cover.*

Proof (*See also [5, Theorem 14.6]*). Notice first that for all $r = 1, \ldots, i$ we know that $\{1, \ldots, m\} - F_{Jr}(e_r)$ is not a cover. Hence, any given cover must intersect all sets $F_{Jr}(e_r)$ $(r = 1, \ldots, i)$. The property follows from the fact that the sets $F_{Jr}(e_r)$ are all disjoint. □

Procedure LOWER_BOUND_1 is first illustrated in the context of an IIS of constraints for the 2-colouring problem on the graph in Fig. 2. The algorithm starts with a 2-colouring e_0 that violates (2, 6), which gives $I_1 = \{(2, 6)\}$. Then, the best 2-colouring e_1 that does not violate (2, 6) necessarily violates two constraints, one in $\{(1, 2), (1, 6)\}$ and one in $\{(2, 3), (3, 4), (4, 5), (5, 6)\}$, say (1, 2) and (2, 3). One therefore gets $I_2 = \{(1, 2), (2, 3), (2, 6)\}$. The next 2-coloring necessarily violates (1, 6) and a constraint in $\{(3, 4), (4, 5), (5, 6)\}$, say (3, 4). We get $I_3 = \{(1, 2), (1, 6), (2, 3), (2, 6), (3, 4)\}$ which is not feasible. The lower bound is therefore equal to 3 which is indeed the size of a minimum IIS. Notice that the above proof is still valid if one replaces MIN by its heuristic version HMIN at Step 2. The advantage of using MIN instead of HMIN is twofold. First, if the output e_i of HMIN(I_i, J_i) is such that $F_{Ii}(e_i) \neq \emptyset$ while I_i is not a cover,

then the algorithm stops prematurely with a too small value for the lower bound. Secondly, if $F_{Ji}(e_i)$ is not of minimum size, then too many elements are added to I_i, and this also tends to reduce the value of the lower bound.

Procedure LOWER_BOUND_1 is not always as successful as for the example in Fig. 2. For illustration, consider the 2-coloring problem for the graph in Fig. 3. The procedure first determines a 2-colouring e_0 with three conflicting edges represented with bold lines. These three edges define I_1 which is not feasible. Hence, the procedure produces a lower bound equal to 1 while the minimum size of an IIS is 3. It is in fact easy to build examples where the lower bound is equal to 1 while the minimum size of an IIS is any given number k.

A better lower bound can be computed using procedure LOWER_BOUND_2 described here below, that does not stop when a cover is detected. This procedure uses a function which, given $m+1$ integers $a_1, \ldots, a_m$ and b returns the value $f(a_1, \ldots, a_m; b)$ computed as follows: the integers $a_1, \ldots, a_m$ are first ordered in a non-increasing order, say $a_{j1} \geqslant \cdots \geqslant a_{jm}$ and $f(a_1, \ldots, a_m; b)$ is then set equal to the smallest index r such that $\sum_{i=1}^{r} a_{ji} \geqslant b$. For example, $f(2, 1, 3, 2, 5; 11)$ is equal to 4 since the integers 2, 1, 3, 2, 5 are first ordered so that $5 \geqslant 3 \geqslant 2 \geqslant 2 \geqslant 1$ and $5+3+2 < 11$ while $5+3+2+2 \geqslant 11$.

Procedure LOWER_BOUND_2
Input: a cover $\{1, \ldots, m\}$; a given number $q \geqslant 1$
Output: a lower bound on the size of a minimum cover
1. Set $i := 0$ and $L := 0$; set $n(r) := 0$ for all $r = 1, \ldots, m$;
2. Set $e_i :=$ MIN-WEIGHT(ω) with $\omega(r) = M^{n(r)}$ for all $r = 1, \ldots, m$ (where M is a number $> m$);
3. Set $n(r) := n(r) + 1$ for all $r \in F_{\{1,\ldots,m\}}(e_i)$;
4. If $n(r) = q$ for some element r, then STOP: L is a lower bound on the size of a minimum cover;
5. Set $i := i + 1$, $L := \max\{L, f(n(1), \ldots, n(m); i)\}$ and go to Step 2.

Procedure LOWER_BOUND_2 is first illustrated in the context of an IIS of constraints for the 2-colouring problem on the graph in Fig. 3. We assume that $q = 4$. All weights $\omega(r)$ are initially set equal to 1 since $n(r) = 0$ for all $r = 1, \ldots, m$. Procedure MIN-WEIGHT(ω) then determines a 2-colouring e_0 with the three conflicting edges represented with bold lines. These three edges get a new weight equal to M since $n(r)$ is now equal to 1 for these edges. Moreover, the lower bound L is set equal to 1. Then, a 2-colouring e_1 is generated with one conflicting edge c in the middle triangle and one conflicting edge in each one of the four pentagons that do not go through c. Edge c gets a new weight equal to M^2 while the other four conflicting edges get a new weight equal to M. The lower bound is still equal to 1 since $n(c) = i = 2$. The third 2-coloring e_2 has one conflicting edge c' of weight $\omega(c') = M$ in the middle triangle, and one conflicting edge of weight 1 in each one of the four pentagons that do not go through c'. The bound is now set equal to 2 since $n(c) < 3$ while $n(c) + n(c') = 4 \geqslant 3$. By repeating this process, one still have a lower bound of 2 when $i = 4$, 5 and 6, while the lower bound is set equal to 3 when $i = 7$. The complete procedure is illustrated in Fig. 4.

The procedure is, however, not always so successful. Indeed, if we apply procedure LOWER_BOUND_2 in the context of an IIS of variables for the 4-colouring problem on the left graph of Fig. 5, one can verify that the procedure produces a lower bound of 5 while a minimum IIS has 7 vertices, as illustrated on the right graph of Fig. 5.

Observe that procedure LOWER_BOUND_2 works as LOWER_BOUND_1 if q is equal to 1. However, if q is larger than 1, it does not stop when a cover has been detected. Any strictly positive integer value can be chosen for q, but since the output of LOWER_BOUND_2 is non-decreasing when q increases, the choice for q mainly depends on the CPU-time available to compute the bound.

Property 8. *Procedure* LOWER_BOUND_2 *produces a lower bound on the size of a minimum cover.*

Proof. Notice first that a cover necessarily intersects $F_{\{1,\ldots,m\}}(e)$ for each $e \in E$. For an element $r \in \{1, \ldots, m\}$, let $n(r)$ denote the number of times that an element e_i has been generated in LOWER_BOUND_2 with $r \in F_{\{1,\ldots,m\}}(e_i)$. If a cover has fewer elements than $f(n(1), \ldots, n(m); i)$, this means that at least one set $F_{\{1,\ldots,m\}}(e_j)$ $(j = 1, \ldots, i)$ has no intersection with this cover, a contradiction. □

The output of function f is large when the input integers $a_1, \ldots, a_m$ are small numbers. This justifies the fact that we give a larger weight to elements of $\{1, \ldots, m\}$ with a large value $n(r)$. Indeed, procedure MIN-WEIGHT(ω) generates an element $e_i \in E$ that minimizes $\sum_{e_i \in Er} \omega(r)$, which means that $F_{\{1,\ldots,m\}}(e_i)$ preferably contains elements r with a small value $n(r)$.

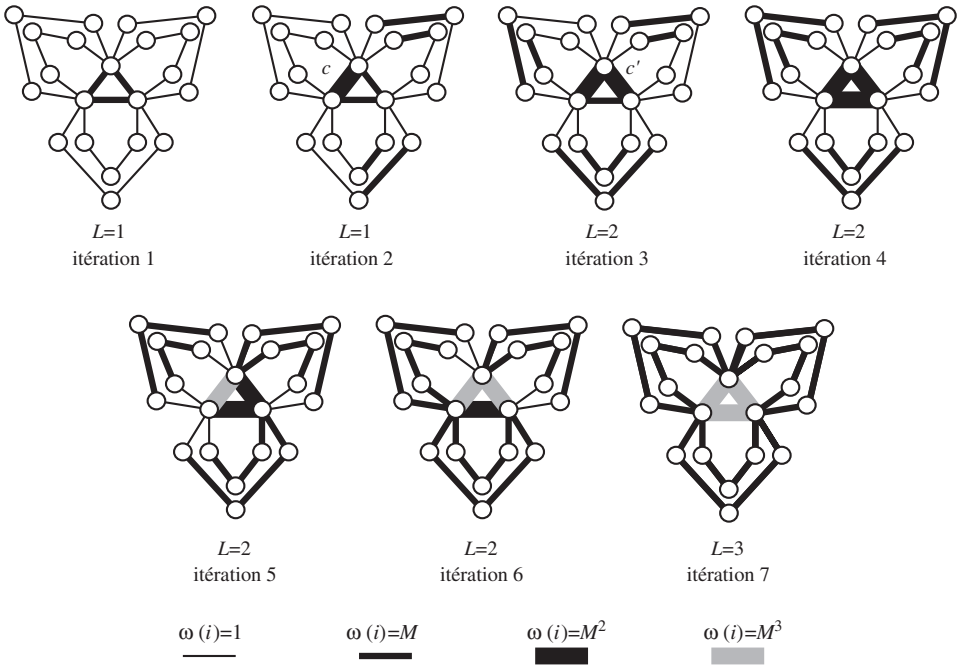

Fig. 4. An illustration of procedure LOWER_BOUND_2.

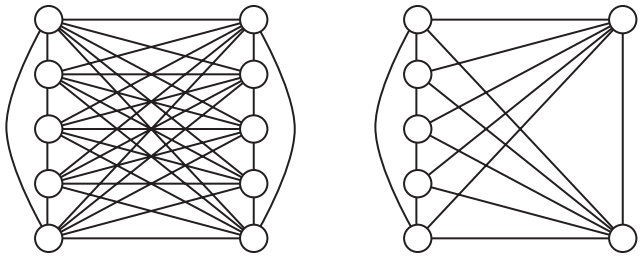

Fig. 5. A graph that is not 4-colourable, and the subgraph induced by a minimum IIS of variables.

6. Finding a maximum subset that is not a cover

In this section we study a problem that is related to the LSCP. It can be formulated as follows. Let E be a set, and let $E_1, \ldots, E_m$ be m subsets of E such that $\bigcup_{i=1}^{m} E_i = E$. Suppose that E and the subsets E_i are very large sets that are possibly infinite. We consider the problem of finding the largest possible subset $I \subseteq \{1, \ldots, m\}$ such that $\bigcup_{i \in I} E_i \neq E$. Such a subset is called a maximum *non-cover*.

Algorithms have been proposed to solve this problem in the context of IISs of constraints in linear programs [3,6,19]. In the context of a CSP, the above problem has two different meanings. On the one hand, if each E_i represents the set of complete assignments that violate constraint c_i, then the above problem is to find a maximum feasible set of constraints, and this is the well known Max-CSP [10]. On the other hand, if each E_i represents the set of legal partial assignments where variable v_i is not instantiated, then the above problem is to instantiate a maximum number of variables without violating any constraint.

The problem of finding a maximum non-cover can easily be solved using procedure MIN-WEIGHT. Indeed, let e be the output of MIN-WEIGHT(ω) where $\omega(i) = 1$ for all $i = 1, \ldots, m$. A maximum non-cover is obtained by setting I equal to the set of elements i such that $e \notin E_i$.

We consider the context where the exact procedure COVER can be used to determine whether a given subset of $\{1, \ldots, m\}$ is a cover while procedure MIN_WEIGHT cannot be used to produce an element e that minimizes $\sum_{e \in E_i} \omega(i)$. This is typically the case for Max-CSP where the problem of finding a maximum feasible solution (i.e., a maximum non-cover) is much harder than the problem of determining whether a given CSP is consistent [10,11]. In the algorithm MAX-NON-COVER presented below, we use the heuristic procedure HMIN in order to determine an element $e \in E$ such that $F_{\{1,\ldots,m\}}(e)$ is as small as possible. The set $\{1, \ldots, m\} - F_{\{1,\ldots,m\}}(e)$ is a non-cover that is possibly maximum, and its size is in every case a lower bound on the size of a maximum non-cover. We then use one of the heuristic procedures HREMOVAL, HINSERTION or HHITTING-SET of Section 4 in order to produce sets that are possibly minimal covers. Procedure COVER is used to confirm that these sets are indeed minimal covers. Since a non-cover cannot contain all elements of a cover, we solve a hitting set problem to remove at least one element in each minimal cover and to obtain an upper bound on the size of a maximum non-cover.

Algorithm MAX-NON-COVER
Input: a cover $I = \{1, \ldots, m\}$
Output: a subset of $\{1, \ldots, m\}$ that is not a cover or an error message
1. Set $e :=$ HMIN$(I, \emptyset)$ and LB $:= m - |F_I(e)|$;
2. Set $i := 1$ and $H_0 := \emptyset$;
3. Call HREMOVAL, HINSERTION or HHITTING-SET with input $I - H_{i-1}$;
   If the output is an error message (see Section 4) then STOP: an error occurred;
   Else let I_i denote this output;
4. If COVER$(I_i) =$ " false" then STOP: an error occurred;
5. Set $H_i :=$ BEST_HS$(I_1, \ldots, I_i)$; UB $:= m - |H_i|$;
6. If LB = UB then STOP: $I - F_I(e)$ is a maximum non-cover;
7. If COVER$(I - H_i) =$ " false" then STOP: $I - H_i$ is a maximum non-cover;
8. Set $i := i + 1$ and go to Step 3.

Property 9. *Algorithm* MAX-NON-COVER *is finite and it either stops with an error message, or it produces a maximum non-cover.*

Proof. Consider two covers I_r and I_s with $s > r$. We know by construction that $H_{s-1} \cap I_r \neq \emptyset$ while $H_{s-1} \cap I_s = \emptyset$. Hence $I_r \neq I_s$ and we can conclude that the algorithm is finite since there are a finite number of subsets of $I = \{1, \ldots, m\}$.

In order to obtain a non-cover, it is necessary to remove at least one element in each cover, in particular in $I_1, \ldots, I_i$. Hence, since H_i is the output of BEST_HS$(I_1, \ldots, I_i)$, we know that $m - |H_i|$ is an upper bound on the size of a maximum non-cover. This also proves that if $I - H_i$ is a non-cover, then it is of maximum size and we can stop at Step 7.

Now, since $I - F_I(e)$ is a non-cover for any $e \in$ E, the value LB $= m - |F_I(e)|$ computed at Step 1 is a lower bound on the size of a maximum non-cover. If LB = UB, one can stop at Step 6 with a guarantee that $I - F_I(e)$ is an optimal solution. □

Notice that algorithm MAX-NON-COVER can be stopped at any time, in which case LB and UB correspond, respectively, to a lower and to an upper bound on the size of a maximum non-cover.

7. Conclusion

In this paper, we have introduced a new problem, called the Large Set Covering Problem (LSCP) which can be seen as a variant of the well known unicost set covering problem. The specificity of the LSCP is that the set E to be covered and the subsets E_i $(i = 1, \ldots, m)$ are not given in extension, while we are given a procedure, called MIN_WEIGHT, that can extract elements from E. An important motivation of the LSCP is to find IISs of variables and IISs of constraints in infeasible CSPs.

We have presented two algorithms REMOVAL and INSERTION for finding minimal covers and a third algorithm, called HITTING-SET, that always produces a minimum cover but with a number of iterations that can be exponential in m. We have noticed that procedure MIN_WEIGHT typically has to solve an NP-hard problem—in the case of IISs of constraints, it is a weighted version of Max-CSP. The exact methods presented in Section 3 can therefore only be applied to small

instances or in particular cases (see for example the CSP studied by Amilhastre et al. [1]). For larger instances, we propose to replace MIN_WEIGHT by a heuristic version, called HMIN_WEIGHT. We have analysed in Section 4 the impact of using such a heuristic which can be implemented using a neighbourhood search technique [11,15,18] (see Section 4). We have shown that the heuristic versions of REMOVAL, INSERTION and HITTING-SET produce either a non-cover, or a minimal cover (i.e., an IIS). Our approach is the opposite of the one proposed in [6,13] for mixed-integer and integer linear programs, where the output is always a cover, but not necessarily a minimal one. We have then proposed in Section 5 several techniques for the computation of lower bounds on the size of a minimum cover. The related problem to find a maximum non-cover was addressed in Section 6.

Most of the algorithms proposed for solving the LSCP can be classified in two different categories. The first category contains algorithms based on procedure COVER that is not as powerful as MIN_WEIGHT—recall that, in the context of IISs of constraints, COVER is a procedure that determines whether a given subset of constraints is feasible, or not. The REMOVAL algorithm (also called *deletion filter* in the context of linear programs [7]) belongs to this first category. The second category contains algorithms that necessitate the use of MIN_WEIGHT. This is the case of the INSERTION and HITTING-SET algorithms proposed in this paper. Several algorithms for finding IISs have already been proposed in the field of continuous optimization (see, e.g., [3–7,20]). Although these algorithms were designed for finding IISs of constraints, they can also be seen as solution techniques for the LSCP. Notice, however, that most of these algorithms belong to the first category. The use of MIN_WEIGHT provides interesting advantages to the algorithms in the second category. For example, the INSERTION algorithm typically requires a much smaller number of steps when compared to the REMOVAL algorithm or the *additive algorithm* [20]. Also, the output of INSERTION is necessarily a minimal cover, while the *elastic filter* [7] requires the use of REMOVAL on its output to isolate a minimal cover.

We have shown in Section 2 that the problems of finding irreducible inconsistent sets (IISs) of constraints and variables in an inconsistent constraint satisfaction problem (CSP) are two particular cases of the LSCP. The algorithms proposed for the solution of the LSCP may therefore be very helpful for detecting IISs. There are at least two motivations for searching IISs of constraints or variables—and, in both cases, preferably IISs of reduced size. First, in many real-life applications, an IIS of constraints makes it easier to understand the cause of infeasibility and, eventually, may help the user to decide which constraints he should relax in his problem. Secondly, the identification of an IIS can be a powerful tool when no exact algorithm is able to prove the infeasibility of a CSP. Indeed, one can use the heuristic versions of REMOVAL, INSERTION or HITTING-SET in order to detect candidate IISs. Then, a proof of infeasibility of one of these IISs is sufficient to prove that the original problem is infeasible.

In order to demonstrate that the proposed algorithms are applicable and useful in practice, it is important to evaluate the computational effort required for detecting IISs, and to show that the proposed techniques may help proving infeasibility. Partial answers are given in [9] where experiments have been conducted on famous benchmarks problems for the k-coloring problem. These experiments demonstrate that IISs can sometimes be found in very short computing times. Also, the detection of IISs has helped proving for the first time the infeasibility of some k-coloring problems—in other words, the best known lower bound on the chromatic number of theses graphs has been strictly increased. A limited number of graphs were even colored optimally for the first time. The approach proposed in this paper should therefore be considered as a very promising avenue in order to prove the infeasibility of various CSP instances.

Acknowledgements

The authors wish to thank the anonymous referees for their useful comments and suggestions.

References

[1] J. Amilhastre, H. Fargier, P. Marquis, Consistency restoration and explanations in dynamic CSPs—application to configuration, Artificial Intelligence 135 (2002) 199–234.
[2] W.B. Carver, Systems of linear inequalities, Ann. Math. 23 (1921) 212–220.
[3] J.W. Chinneck, An effective polynomial-time heuristic for the minimum-cardinality IIS set-covering problem, Ann. Math. Artificial Intelligence 17 (1996) 127–154.
[4] J.W. Chinneck, Finding a useful subset of constraints for analysis in an infeasible linear program, INFORMS J. Comput. 9 (2) (1997) 164–174.
[5] J.W. Chinneck, Feasibility and viability, in: T. Gal, H.J. Greenberg (Eds.), Recent Advances in Sensitivity Analysis and Parametric Programming, Kluwer Academic Publishers, Boston, 1997, pp. 14:2–14:41.
[6] J.W. Chinneck, Fast heuristics for the maximum feasible subsystem problem, INFORMS J. Comput. 13 (3) (2001) 210–223.
[7] J.W. Chinneck, E.W. Dravnieks, Locating minimal infeasible constraint sets in linear programs, ORSA J. Comput. 3 (1991) 157–168.

[8] J.M. Crawford, Solving satisfiability problems using a combination of systematic and local search, in: Working Notes of the DIMACS Workshop on Maximum Clique, Graph Coloring, and Satisfiability, Rutgers University, NJ, 1993.
[9] C. Desrosiers, P. Galinier, A. Hertz, Efficient algorithms for finding critical subgraphs, Discrete Appl. Math., 2006, to appear.
[10] E.C. Freuder, R.J. Wallace, Partial constraint satisfaction, Artificial Intelligence 58 (1992) 21–70.
[11] P. Galinier, J.K. Hao, Tabu search for maximal constraint satisfaction problems, in: Proceedings of Third Internat. Conf. on Principles and Practice of Constraint Programming (CP'97), Lecture Notes in Computer Science, vol. 1330, Springer, Berlin, 1997, pp. 196–208.
[12] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, San Francisco, 1979.
[13] O. Guieu, J.W. Chinneck, Analyzing infeasible mixed-integer and integer linear programs, INFORMS J. Comput. 11 (1) (1999) 63–77.
[14] F. Herrmann, A. Hertz, Finding the chromatic number by means of critical graphs, ACM J. Experimental Algorithmics 7 (10) (2002) 1–9.
[15] A. Hertz, D. de Werra, Using Tabu search techniques for graph coloring, Computing 39 (1987) 345–351.
[16] A.K. Mackworth, Constraint satisfaction, in: S.C. Shapiro (Ed.), Encyclopedia on Artificial Intelligence, Wiley, New York, 1987, pp. 205–211.
[17] B. Mazure, L. Saı̈s, É. Grégoire, Boosting complete techniques thanks to local search methods, Ann. Math. Artificial Intelligence 22 (1998) 319–331.
[18] C. Morgenstern, Distributed coloration neighborhood search, in: D.S. Johnson, M.A. Trick (Eds.), Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, DIMACS Ser. Discrete Math. Theoret. Comput. Sci., vol. 26, American Mathematical Society, Providence, 1996, pp. 335–357.
[19] M. Parker, J. Ryan, Finding the minimum weight IIS cover of an infeasible system of linear inequalities, Ann. Math. Artificial Intelligence 17 (1996) 107–126.
[20] M. Tamiz, S.J. Mardle, D.F. Jones, Detecting IIS in infeasible linear programmes using techniques from goal programming, Comput. Oper. Res. 23 (2) (1996) 113–119.
[21] E.P.K. Tsang, Foundations of Constraint Satisfaction, Academic Press, London, 1993.
[22] J. Van Loon, Irreducibly inconsistent systems of linear inequalities, European J. Oper. Res. 8 (1981) 283–288.