Finding irreducible infeasible sets in inconsistent constraint satisfaction problems

Alain Hertz
GERAD – École Polytechnique
Montréal, Canada

This is a joint work with
Philippe Galinier and Christian Desrosiers
Ecole Polytechnique, Montréal
I  The main idea
II  An example: the k-coloring problem
III Problem formulation
IV  A crazy assumption
V   Solution techniques
VI  A more realistic assumption
VII A related problem
VIII  What to do in practice
IX  Computational results
I  The main idea

Let P be a problem for which we are not able to determine any feasible solution.

How can we prove that P has no feasible solution?

1. Determine a subproblem P' for which we are also not able to determine any feasible solution
2. Prove that P' has no feasible solution
   ⇒ this proves that P also has no feasible solution
II An example: the k-coloring problem

Input: a graph G, a number k

Problem: is it possible to color the vertices of G with at most k colors such that no two adjacent vertices get the same color?

If the answer is yes, then G is k-colorable

Example

This graph is 3-colorable but not 2-colorable

The smallest k such that G is k-colorable is called the chromatic number of G.

How can we prove that G is not 2-colorable?

Idea: and are also not 2-colorable
First formulation

A solution is any assignment of a color to each vertex. An edge having its endpoints of the same color is called a conflicting edge.

Problem : find a solution with as few conflicting edges as possible.

No-conflicting edge ⇔ G is k-colorable

a 2-coloring with one conflicting edge
(optimal solution for k=2)
Second formulation

A solution is any legal partial coloring (without conflicting edge).

**Problem**: find a solution with as few non-colored vertices as possible.

All vertices colored $\iff G$ is k-colorable

a 2-coloring with one non-colored vertex (the grey one)
(optimal solution for $k=2$)
Up to a permutation there are 16 different (non necessarily legal) 2-colorings of G

G is not 2-colorable because all these 16 2-colorings have at least one conflicting edge
Let $E_i$ denote the set of 2-colorings in which edge $i$ is conflicting.

Example

G is not 2-colorable because the sets $E_i$ cover the set $E$ of possible 2-colorings.
These 3 sets $E_i$ also cover $E$
They also explain why $G$ is not 2-colorable

$G$ is not 2-colorable because of

This is an irreducible infeasible set
The graph has a second irreducible infeasible set (for $k=2$). Indeed, it is not 2-colorable because of the existence of
For the second formulation

Let \( E = \) set of legal partial \( k \)-colorings
(i.e., there are no conflicting edges, but there may exist non-colored vertices)

**Question**: does \( E \) contain a solution where all vertices are colored?

Let \( E_i \) denote the set of solutions in \( E \) where vertex \( i \) is non-colored.

The sets \( E_i \) cover \( E \) \iff \( G \) is not 2-colorable
Example

This is not 2-colorable

This is the set of legal partial 2-colorings (up to a permutation of the colors)
This is the subset $E_i$ where the middle vertex is non-colored
These three subsets already cover $E$
They explain why $G$ is not 2-colorable
$G$ is not 2-colorable because of

This is an **irreducible infeasible set**
The constraint satisfaction problem (CSP)

- a finite set of *variables*
- each variable is associated with a *domain* of values
- 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 overconstrained, or infeasible).
Example: the k-coloring problem

- the vertices of the graph are the variables
- the set of possible colors \{1,\ldots,k\} is the domain of each variable
- the inequality constraints between adjacent vertices are the constraints
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 is *infeasible* if no complete assignment satisfies all constraints in $S$ simultaneously.

A subset of variables is *infeasible* if there is no legal partial assignment of these variables.
• 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.

• A subset of variables is called an irreducible infeasible set (IIS) of variables if it is infeasible, but becomes feasible when any one variable is removed.
Illustration

with $k=2$

has 2 IISs of constraints

and 1 IIS of variables
Why is it interesting to detect IISs?

- An IIS represents a part of the problem that gives a partial explanation to its infeasibility.

- Determining IISs of constraints or variables can be very helpful to prove the inconsistency of a CSP.

  Indeed, IISs contain typically a small amount of constraints and variables when compared to the original problem

  ⇒ a proof of inconsistency is possibly easier to obtain on an IIS rather than on the original problem.
A typical example

Question: how can we prove the optimality of a solution delivered by a heuristic method?

1. Determine an IIS of constraints or variables
2. Prove that this IIS is not feasible

Example: which is the chromatic number of ?

A heuristic method was able to color this graph with 3-colors, but not with 2. How can we prove that 3 is the chromatic number of this graph?

1. Determine an IIS of constraints or variables:
   \[ \Rightarrow \text{or} \]

2. Prove that this IIS is not 2-colorable
The General Procedure

1. use a heuristic algorithm do determine a k-coloring of a given graph G with the smallest possible number k of colors
2. consider the problem P of determining a (k-1)-coloring of G. This problem has probably no solution
3. generate a subgraph of G that has no solution for P
4. prove that this subgraph is indeed not (k-1)-colorable
   ⇒ this proves that P has no solution
   ⇒ this proves that the chromatic number is k

Illustration for G:

1. determine a 3-coloring
2. consider the 2-coloring problem on G
3. generate the subgraph G’=
4. prove that G’ is not 2-colorable
   ⇒ this proves that G is not 2-colorable
   ⇒ this proves that the chromatic number of G is equal to 3.
III Problem formulation

A more general problem: the Large Set Covering Problem (LSCP)

The Unicost set covering problem (USCP).
Given a finite set $E$ and a family $F={E_1, \ldots, E_m}$ of subsets of $E$ such that $F$ covers $E$, the USCP is to determine the smallest possible subset of $F$ that also covers $E$.

The LSCP is a variant of the USCP 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 are possibly infinite sets.
Definition

A subset $I$ of $\{1, \ldots, m\}$ such that $\bigcup_{i \in I} E_i = E$ is called a cover.

The LCSP is to determine a minimal (inclusion wise) cover.

We also consider the problem, called minimum LSCP which is to determine a minimum cover.
The problems of finding IISs of constraints and variables in an inconsistent CSP are two special cases of the LSCP.

Given an inconsistent CSP,
1. Define as an element of $E$ as any complete assignment.
2. To each constraint $c_i$ ($i=1,…,m$) of the CSP, associate the subset $E_i$ of $E$ containing all assignments that violate $c_i$.

$\Rightarrow$ A set of constraints is infeasible if and only if it covers $E$.
$\Rightarrow$ finding an IIS of constraints is equivalent to solving the LSCP.
Given an inconsistent CSP, 
1. Define as an element of $E$ any legal partial assignment. 
2. To each variable $v_i$ of the CSP, associate the subset $E_i$ of $E$ containing all legal partial assignments in which $v_i$ has no value. 

$\Rightarrow$ A subset of variables is infeasible if and only if it covers $E$. 
$\Rightarrow$ finding an IIS of variables is equivalent to solving the LSCP. 

**Conclusion**

**IISs in an inconsistent CSP are minimal covers for the LSCP.**
IV A crazy assumption
We are given

- a procedure 
  \textbf{Is-Element}(e,i) that returns value “true” if and only if \( e \in E_i \).

\textbf{First model} : given any k-coloring, the procedure tells you whether a given edge
is conflicting or not

\textbf{Second model} : given any partial legal k-coloring, the procedure tells you
whether a given vertex is colored or not
We are given (continued)

- given any weighting function \( \omega \) that assigns a weight \( \omega(i) \) to each set \( E_i \), we can use a procedure \( \text{Min\_Weight}(\omega) \) that returns an element \( e \in E \) such that \( \sum_{e \in E_i} \omega(i) \) is minimum.

First model: if all edges have a weight \( \omega(i)=1 \), then \( \text{Min\_Weight}(\omega) \) returns a k-coloring with as few conflicting edges as possible.

Second model: if all vertices have a weight \( \omega(i)=1 \), then \( \text{Min\_Weight}(\omega) \) returns a legal partial k-coloring with as few non-colored vertices as possible.
V Solution techniques

Notations

• $F_f(e) = \text{subset of elements } i \in I \text{ such that } e \in E_i$.
  Such a subset can easily be generated by means of Is-Element.

  (it is  
  - the subset of constraints violated by solution $e$, or 
  - the subset of non-instantiated variables in solution $e$)
Given a subset $I \subseteq \{1, \ldots, m\}$, procedure $\text{Cover}(I)$ returns the value “true” if $\bigcup_{i \in I} E_i = E$, and “false” otherwise.

Consider the weighting function $\omega$ that assigns a weight

- $\omega(i) = 1$ to each index $i \in I$
- $\omega(i) = 0$ to the other indices,

Let $e$ be the output of $\text{Min\_Weight}(\omega)$. Then $\text{Cover}(I)$ returns the value “true” if and only if $F_I(e) \neq \emptyset$. 
We denote \( \text{Min}(I,J) \) 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 \( |J| \).
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 \(\text{Cover}(I \setminus \{-i\}) = \text{"true"}\) then set \(I := I \setminus \{-i\}\).

**Property** Algorithm Removal produces a minimal cover.
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 \( F_{I_i}(e_i)\neq \emptyset \) then STOP: \( I_i \) is a minimal cover;
4. Choose \( h_i \) in \( F_{J_i}(e_i) \), set \( I_{i+1}:=I_i \cup \{h_i\}, J_{i+1}:=J_i - F_{J_i}(e_i) \), \( i:=i+1 \) and go to Step 2.

Property Algorithm Insertion produces a minimal cover.
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 \( F_{xi}(e_i) \neq \emptyset \) then STOP: \( I_i \) is a minimum cover;
4. Set \( i := i + 1, I_i := \text{Best_HS}(F_{i}(e_0), \ldots, F_{i}(e_{i-1})) \), and go to Step 2.

Property Algorithm Hitting-Set produces a minimum cover.
Comparisons

- Algorithm **Removal** and **Insertion** always produce a minimal cover.
- Algorithm **Hitting-Set** always produces a minimum cover.
- **Removal** finds a minimal cover in $m$ steps.
- **Insertion** finds a minimal cover in $|I|$ steps.
- **Hitting-Set** finds a minimum cover in an exponential number of steps.
- But **Hitting-Set** can be stopped at any time to produce a lower bound on the size of a minimum cover.
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 \(e_i:=\text{Min}(I_i,J_i)\);

3. If \(F_{I_i}(e_i)\neq\emptyset\) then STOP: \(i\) is a lower bound on the size of a minimum cover;

4. Set \(I_{i+1}:=I_i \cup F_{J_i}(e_i), J_{i+1}:=J_i - F_{J_i}(e_i), i:=i+1\) and go to Step 2.

Property Procedure Lower_Bound_1 produces a lower bound on the size of a minimum cover
Given \( m+1 \) integers \( a_1, \ldots, a_m \) and \( b \) we compute \( f(a_1, \ldots, a_m; b) \) computed as follows:

1. the integers \( a_1, \ldots, a_m \) are first ordered in a non-increasing order, say \( a_{j_1} \geq \ldots \geq a_{j_m} \)

2. \( f(a_1, \ldots, a_m; b) \) is then set equal to the smallest index \( r \) such that \( \sum_{i=1}^{r} a_{j_i} \geq b \).

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 \( 5 \geq 3 \geq 2 \geq 2 \geq 1 \) and \( 5+3+2<11 \) while \( 5+3+2+2 \geq 11 \).
Procedure Lower_Bound_2

**Input:** a cover \{1,\ldots,m\}; a given number \(q \geq 1\)

**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 \(e_i:=\text{Min-Weight}(\omega)\) 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.

**Property** Procedure Lower_Bound_2 produces a lower bound on the size of a minimum cover
GERAD

1 1,1
1+1 2,2
1+1+1 3,3
1+1+1+1 4,4
1+1+1+1 5,5
2+2+1+1 4,6

2+2+2+2 4,7
2+2+2+2 4,8
2+2+2+2+2 5,9
2+2+2+2+2 5,10
3+3+2+2+2 5,11
3+3+3+3 4,12
3+3+3+3+3 5,13
VI  A more realistic assumption

Procedure Min_Weight(\(\omega\)) solves an NP-hard problem

For the k-coloring with \(\omega(i)=1\) for all \(i\), \(\text{Min\_Weight}(\omega)\) determines a k-coloring with as few conflicting edges as possible

\[\Rightarrow\] the output of \(\text{Min\_Weight}(\omega)\) has no conflicting edge if and only if the graph is k-colorable

\[\Rightarrow\] one can determine the chromatic number by using \(\text{Min\_Weight}(\omega)\)
What happens if one replaces Min.Weight($\omega$) by a heuristic algorithm?

Property

If the output of Removal is a cover, then it is a minimal cover.
If the output of Insertion is a cover, then it is a minimal cover.
If the output of Hitting-Set is a cover, then it is a minimum cover.
Let P be a problem for which we are not able to determine any feasible solution. How can we prove that P has no feasible solution?

1. Determine a subproblem P’ for which we are also not able to determine any feasible solution.
2. Prove that P’ has no feasible solution
   \[\Rightarrow\] this proves that P also has no feasible solution.

In other words,

1. Detect a possible cover (IIS) by means of the heuristic versions of Removal, Insertion or Hitting-Set.
2. Prove that this possible cover is indeed a cover (IIS).
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.
Example

given a graph that is not k-colorable, find the largest possible subgraph that is k-colorable

Input

Ouput:

for the “constraint” formulation

for the “variable” formulation
This problem can easily be solved using procedure Min-Weight(\(\omega\)).

Indeed, let \(e\) be the output of Min-Weight(\(\omega\)) 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 \(e \notin E_i\).

If we have to use a heuristic version of Min-Weight(\(\omega\)), then we can use the following algorithm
Algorithm Max-non-cover

**Input:** a cover $I=\{1,\ldots,m\}$

**Output:** a subset of $\{1,\ldots,m\}$ that is not a cover or a message of error

1. Set $e:=\text{Min-}(I,\emptyset)$ and $\text{LB}:=m-|F_1(e)|$;
2. Set $i:=1$ and $H_0:=\emptyset$;
3. Call Removal, Insertion or Hitting-Set with input $I-H_{i-1}$;
   If the output is a message of error then STOP: an error occurred;
   Else let $I_i$ denote this output;
4. If $\text{Cover}(I_i)=\text{false}$ then STOP: an error occurred;
5. Set $H_i:=\text{Best_HS}(I_1,\ldots,I_i)$; $\text{UB}:=m-|H_i|$;
6. If $\text{LB}=? \text{UB}$ then STOP: $I-F_1(e)$ is a maximum non-cover;
7. If $\text{Cover}(I-H_i)=\text{false}$ then STOP: $I-H_i$ is a maximum non-cover;
8. Set $i:=i+1$ and go to Step 3.

**Property** This algorithm is finite and it either stops with a message of error, or it produces a maximum non-cover
VIII  What to do in practice

Heuristic algorithms for *Min-Weight*(ω)

1. Minimize the number the total weight of the conflicting edges

   Tabu search algorithm with
   - search space = set of k-colorings
   - objective function = total weight of the conflicting edges
   - neighborhood = move a vertex from one color class to another
Heuristic algorithms for Min-Weight(\(\omega\))

(continued)

2. Minimize the total weight of the non-colored vertices

Tabu search algorithm with
- search space = set of partial k-colorings
- objective function = total weight of the non-colored vertices
- neighborhood = assign a color \(i\) to a non-colored vertex \(v\) and remove the color from each vertex adjacent to \(v\) having color \(i\).

\((i\)-swaps defined by Morgenstern, 1996\)
What else can we do in practice?

Procedure Pre-filtering

**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 \( e_i := \text{Min}(I_i, J_i) \);
3. If \( F_{I_i}(e_i) \neq \emptyset \) then STOP: Set \( G := G[I_i] \);
4. Set \( I_{i+1} := I_i \cup F_{J_i}(e_i), J_{i+1} := J_i - F_{I_i}(e_i), i := i + 1 \) and go to Step 2.
Speed up procedure

When the procedure that solves $\text{Min} \ (I,J)$ detects a solution $e$ with $|F_1(e)|=1$ then insert the unique element of $F_1(e)$ into $I$
Heuristic for choosing $h_i$ in the Insertion algorithm

1. Partial $k$-colorings
   Choose the vertex $v$ with maximum total weight in its neighborhood
   
   $$(\text{minimise } \sum_{u \in N(v)} \omega(u))$$

2. Non-legal $k$-colorings
   Choose the edge $e$ with maximum total weight in its neighborhood
   
   $$(\text{minimise } \sum_{e' \in N(e)} \omega(e'))$$
\begin{align*}
4 + 9 + 9 &= 22 \\
2 + 4 + 9 &= 15
\end{align*}
\[ 6 + 1 = 7 \]
IX Computational results

Example

Original graph $G_{180,0.1}$:
- 180 vertices,
- 1603 edges
- Exact method: more than 250 000 000 backtracks

IIS:
- 78 vertices
- 467 edges
- Exact method: 10347 backtracks
Typical behavior

Number of backtracks for the original graph

Number of backtracks for the IIS

Chromatic number
Questions