Stepwise Explanations of Unsatisfiable Constraint Programs
(Extended abstract)

Ignace Bleukx[1], Emilio Gamba[2], Bart Bogaerts[2], and Tias Guns[1]

[1] KU Leuven, Belgium *firstname.lastname*`@kuleuven.be`
[2] Vrije Universiteit Brussel, Belgium *firstname.lastname*`@vub.be`

1 Introduction

Formulating combinatorial problems as a constraint satisfaction problems (CSP) or optimization problems (COP) are popular ways for modelling and solving a multitude of problems including routing, scheduling, configuration,. . . [1, 3, 9].

Unsatisfiable constraint models are models to which no satisfying assignment exist. Many developers starting with constraint programming have encountered unsatisfiable models while testing CSP implementations. This can either be because the problem at hand does not have any solutions, or be the result of a modelling error. In any case, the user can be interested in an explanation why his model is unsatisfiable and potentially use it to debug his model.

Unfortunately, only a handful tools are available for debugging such unsatisfiable models. Many, if not all, of these tools consist of finding a minimal unsatisfiable subset (MUS) of constraints which is presented to the user as an explanation for the unsatisfiability [7, 8]. Although some obvious bugs in the model can be detected easily using such techniques, in other cases the extracted MUS involves too many constraints to understand their dynamics and the cause of unsatisfiability may not be made clear.

Indirectly, an explanation of why a CSP is unsatisfiable can be useful in other contexts as well. An example is explaining the optimality of a COP. This can be done asking the question: *'why is there no better solution possible?'*, that is, by adding a constraint that forces the objective function to have a better-than-optimal value. If the original solution was truly optimal, adding this new constraint to the model will result in an unsatisfiable problem. By explaining the unsatisfiability of this new model, we can explain to the user why no better solutions exist for the COP.

Recent work on explanations for *satisfiable* CSPs has introduced the concept of stepwise explanations [2]. In that setting the goal is to explain facts which are true in the solution of the problem. This is done using a sequence of small steps each involving as few constraints as possible to explain an additional fact.

In this paper, we aim to generate an explanation sequence explaining why a CSP is *unsatisfiable*. This sequence will explain facts until an inconsistency such as $a \wedge \neg a$ is discovered, or in CSP terms: it will explain domain reductions until an empty domain is obtained. For a user not to be overwhelmed by such an explanation sequence, it must involve as few steps as possible, each consisting of only a small set of constraints. This ensures every step is interpretable. We prioritize the size of each step over the total length of the sequence.

Our contributions are as follows:

- First, we extend the notion of stepwise explanations to problems considering variables in the integer domain;
- Second, we motivate the partitioning of the constraints in a CSP into simple and non-simple constraints; and
- Finally, we propose an algorithm for deriving the unsatisfiability of a CSP using stepwise explanations.

2 Stepwise explanations

Stepwise explanations of a Boolean satisfiability problem are introduced by Bogaerts, Gamba, and Guns [2]. Their goal is to explain a fact (the truth value of a Boolean variable) in the solution of a SAT problem. This is done by building a sequence of small explanation steps ultimately deriving the set of facts we wish to explain. Starting from a given (possibly empty) set of facts, the sequence iteratively derives new facts which are true in the solution using small steps to ease interpretability. The assumption is made that the model is satisfiable and there exists a unique solution.

In particular, an explanation step consists of a 3-tuple (E, S, N) where each component has the following semantics:

- E the set of facts used to propagate the constraints in S
- S the set of constraints propagated in this explanation step
- N the set of new facts derived from the propagation of S and E

Formally, it must hold that $E \wedge S \models N$, i.e. any new information derived in the step must be implied by the set of facts and constraints used in that step [2].

To calculate an explanation step, first a set of possible facts is constructed. In all settings proposed by Bogaerts, Gamba, and Guns [2], the possible facts are those that are true in the unique solution of the problem.

Next, their algorithm computes set of constraints that cause any one of these facts to be true. This is done by iteratively calculating MUSes from the set of constraints, the already derived facts and the negation of the possible new facts that could be derived. This process is repeated until an optimal (smallest or cost-optimal) MUS is found. An application of this explanation approach can be to show the steps needed to fill in an unfinished sudoku puzzle.

2.1 Extension to integer variables

As a first contribution, we extend the notion of stepwise explanations from Boolean to integer domains. Because a fact is denoted as the truth value of a Boolean variable, this extension is not trivial. A naive solution would be to compile the CSP to a SAT problem by representing integers as Boolean variables [10]. Other than introducing a lot of new variables, such an approach would require post-processing the clausal explanations to generate meaningful problem-level explanations. After all, the user is reasoning in the domain of the CSP itself rather than the Boolean domain of the compiled CSP.

Instead, we propose to represent information about variables (facts) using their allowed domain. The 3-tuple (E, S, N) representing an explanation step in a CSP context now has the following semantics:

- E is a set of allowed domain values for a set of variables;
- S a set of constraints involving only variables occurring in E
- N are the shrunken domains which result from filtering E using S.

We now explain how to derive the set of filtered domains N from E and S. Given a CSP$(\mathcal{X}, \mathcal{D}, \mathcal{C})$ with $\mathcal{X}$ a set of variables, $\mathcal{D}$ a corresponding set of domains and $\mathcal{C}$ a set of constraints. Filtering the domain $\mathcal{D}(x)$ of variables $x \in \mathcal{X}$ using a set of constraints C is done by propagating information in the constraints. We derive N by computing all values in S which have a solution satisfying constraints in S. Let *scope*(S) be the set of variables occurring in the set of constraints S, then the shrunken domains are $N = \bigcup_{sol(scope(S),E,S)}$ where $sol(\mathcal{X}, \mathcal{D}, \mathcal{C})$ is the set of all solutions of the CSP$(\mathcal{X}, \mathcal{D}, \mathcal{C})$. Using this formalism, we are able to represent newly derived information at every step directly in the domain of the original CSP.

In any explanation step, the goal is now to find a smallest set of constraints which eliminate a value for at least one variable when propagated.

Example Suppose we have some CSP with integer variables a and b. Both variables have a domain from 0 to 5. If there exists some constraint $a < b$, then the following tuple (E, S, N) represents an explanation step:

$$(\{a \in [0, 5], b \in [0, 5]\}, \{a < b\}, \{a \in [0, 4], b \in [1, 5]\})$$

Or informally, *Given that* a *and* b *can both take values from 0 to 5 and* a *must be strictly smaller than* b*, we can derive* a *takes values from 0 to 4 and the value of* b *lies between 1 and 5.*

3 Skipping simple explanations

Another difference between CSP's and Boolean satisfaction problems is the form of constraints. In a SAT problem, constraints are clauses. In constraint programming, a wide variety of high level constraints are available such as global constraints with specific semantics [5].

Some constraints and their effects are easier for a user to understand than others. Examples of these are straightforward relations between variables $(x = y)$ or bounds on the domain of variable $(x \leq c)$. We argue that explanations for sets of such straightforward constraints can be hidden and do not need to be shown to the user. We refer to this subset as the *simple* constraints $(\mathcal{C}_s \subset \mathcal{C})$.

When constructing explanation sequences for CSPs, the consequence of propagating only simple constraints should not be explained in a dedicated step. Instead, we can propagate the entire set of simple constraints at once and present this as a single *simple* step, thereby shortening the sequence as a whole.

The remaining *non-simple* constraints, denoted by the set $\mathcal{C}_{ns} = \mathcal{C} \setminus \mathcal{C}_s$, are the constraints that make up the core of the problem and that may interact in complex ways. These are the constraints an explanation is needed for.

In any explanation setting, the *non-simple* constraint set cannot be empty. This is logical as a user does not need an explanation of constraints he can understand all consequences of.

Example Suppose we have a CSP containing integer variables x and y with domain $[0, 10]$ and constraints $\mathcal{C}_s = \{x \geq 5, x = y\}$. Instead of adding two steps $(\{x \in [0, 10]\}, \{x \geq 5\}, \{x \in [5, 10]\})$ and $(\{x \in [5, 10], y \in [0, 10]\}, \{x = y\}, \{y \in [5, 10]\}$ to the explanation sequence, we derive this information in one trivial step: $(\{x \in [0, 10], y \in [0, 10]\}, \mathcal{C}_s, \{x \in [5, 10], y \in [5, 10]\})$. Notice that changes in the domain of y directly affect the domain of x. As this relation is part of the simple constraints, such effects will not be explained separately.

4 Explanation of UNSAT

The method of Bogaerts, Gamba, and Guns [2] provides stepwise explanations of SAT and iteratively searches for an explanation for each of the derivable facts. These are the facts which are true in the solution of the problem. Our goal instead is to explain UNSAT, that is, an empty domain for one variable, though we do not know which variable in advance.

In every explanation generation step, by lack of knowledge of facts that can (not) be derived, all theoretically possible facts should be checked why they cannot be true. Then, the fact which can be ruled out using the least amount of constraints is chosen. This set of constraints will always exist as the set of all constraints is incompatible with any value assignment for any variable.

To compute this set of constraints, we use the concept of an *optimal constrained unsatisfiable subset* (OCUS) proposed by Gamba, Bogaerts, and Guns [4]. As the name suggests, this is an unsatisfiable subset which additionally satisfies some constraint. The authors provide an algorithm for calculating this OCUS as an extension to the well-known hitting set-based MUS algorithm [6] and show how it can be used in an explanation generation setting.

We use the algorithm to find a smallest set of constraints which is incompatible with at least one value in the domain of some variable in the CSP. This set of constraints is then propagated to filter domains of all variables in its scope.

By filtering the domain of at least one variable at each step in the explanations sequence, eventually the domain of a variable will become empty and the algorithm terminates. A formal description is given in algorithm 1

The sequences of explanation steps now serves as an explanation why there can be no solutions, and thus why the CSP is unsatisfiable.

4.1 Example

Consider the following unsatisfiable CSP $(\mathcal{X}, \mathcal{D}, \mathcal{C}_s \cup \mathcal{C}_{ns})$:

- $\mathcal{X} = \{i, j, k, l, m\}$
- $\mathcal{D}(i) = \mathcal{D}(j) = \mathcal{D}(k) = \mathcal{D}(l) = \mathcal{D}(m) = [0, 20]$
- $\mathcal{C}_s = \{k \geq 7, m \leq 10\}$
- $\mathcal{C}_{ns} = \{i = j, m = max(i, j, k, l), 3 * i = l, i + j = k\}$

If a MUS is extracted from the constraints of this CSP to explain the user why there are no solutions, every constraint would be present in this MUS. Naturally this is not a good explanation for why the CSP is UNSAT.

The separation of the constraints means the user understands all ramifications of the constraints $k \geq 7$ and $m \leq 10$, i.e. any solution of the CSP has some value for k in the interval $[7, 20]$ and a value for m in the interval $[0, 10]$. A possible sequence explaining the unsatisfiability of this CSP is shown in table 1.

Table 1. An explanation sequence deriving the unsatisfiability of a CSP

step	E	S	N
trivial	$\mathcal{D}(k) = [0, 20], \mathcal{D}(m) = [0, 10]$	$\mathcal{C}_s$	$\mathcal{D}(k) = [7, 20], \mathcal{D}(m) = [0, 10]$
1	$\mathcal{D}(i) = [0, 20], \mathcal{D}(j) = [0, 20]$ $\mathcal{D}(k) = [7, 20], \mathcal{D}(l) = [0, 20],$ $\mathcal{D}(m) = [0, 10]$	$m =$ $max(i, j, k, l)$	$\mathcal{D}(i) = [0, 10], \mathcal{D}(j) = [0, 10]$ $\mathcal{D}(k) = [7, 10], \mathcal{D}(l) = [0, 10]$
2	$\mathcal{D}(i) = [0, 10], \mathcal{D}(l) = [0, 10]$	$3 * i = l$	$\mathcal{D}(i) = [0, 3], \mathcal{D}(l) = \{0, 3, 6, 9\}$
3	$\mathcal{D}(i) = [0, 3], \mathcal{D}(j) = [0, 10]$ $\mathcal{D}(k) = [7, 10]$	$i + j = k$	$\mathcal{D}(j) = [4, 10]$
4	$\mathcal{D}(i) = [0, 3], \mathcal{D}(j) = [4, 10]$	$i = j$	$\mathcal{D}(i) = \emptyset, \mathcal{D}(j) = \emptyset$

5 Open questions

Preliminary testing of our algorithm shows promising results. However, a lot of improvements and open questions still remain.

- How can the search of explanations be guided so the sequence is as small as possible? For example by implementing heuristics as weights on constraints?
- Can we restrict the domain of variables in the explanation sequence to the parts that matter for the conflict? Sometimes only the lower or upper bound of a domain interval is needed for deriving an *OCUS*.
- Are there better methods than partitioning constraints in $\mathcal{C}_s$ and $\mathcal{C}_{ns}$ to eliminate simple explanation steps, or more generally show only the most relevant parts of a sequence?
- Other than the size of the *OCUS* found in the algorithm, can we use other metrics to evaluate an explanation step? Examples include the scope of an explanation or the number of filtered values.
- Computing OCUS'es for every step are expensive operations. How can we ensure the runtime of explanation generation becomes tractable?

References

[1] Roman Barták. "Constraint programming: In pursuit of the holy grail". In: *Proceedings of the Week of Doctoral Students (WDS99)*. Vol. 4. MatFyz-Press Prague. 1999, pp. 555–564.

[2] Bart Bogaerts, Emilio Gamba, and Tias Guns. "A framework for step-wise explaining how to solve constraint satisfaction problems". In: *Artificial Intelligence* 300 (2021), p. 103550.

[3] Edward H Bowman. "The schedule-sequencing problem". In: *Operations research* 7.5 (1959), pp. 621–624.

[4] Emilio Gamba, Bart Bogaerts, and Tias Guns. "Efficiently explaining csps with unsatisfiable subset optimization". In: *arXiv preprint arXiv:2105.11763* (2021).

[5] Willem-Jan van Hoeve and Irit Katriel. "Global constraints". In: *Foundations of Artificial Intelligence*. Vol. 2. Elsevier, 2006, pp. 169–208.

[6] Alexey Ignatiev et al. "Smallest MUS extraction with minimal hitting set dualization". In: *International Conference on Principles and Practice of Constraint Programming*. Springer. 2015, pp. 173–182.

[7] Kevin Leo and Guido Tack. "Debugging unsatisfiable constraint models". In: *International conference on AI and OR techniques in constraint programming for combinatorial optimization problems*. Springer. 2017, pp. 77–93.

[8] Mark H Liffiton and Karem A Sakallah. "Algorithms for computing minimal unsatisfiable subsets of constraints". In: *Journal of Automated Reasoning* 40.1 (2008), pp. 1–33.

[9] Paul Shaw. "Using constraint programming and local search methods to solve vehicle routing problems". In: *International conference on principles and practice of constraint programming*. Springer. 1998, pp. 417–431.

[10] Naoyuki Tamura et al. "Compiling finite linear CSP into SAT". In: *International Conference on Principles and Practice of Constraint Programming*. Springer. 2006, pp. 590–603.

A Appendix

Algorithm 1 Stepwise explanation of unsatisfiability

```
Input: $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{C}_s \cup \mathcal{C}_u)$
Output: A sequence of explanations steps leading to a contradiction
 1: $\mathcal{D}' \leftarrow \mathcal{D}$
 2: $seq \leftarrow <>$
 3: while $\mathcal{D}'$ is consistent do
 4:     $\mathcal{D}' \leftarrow \mathcal{D}' \cap filter(\mathcal{D}', \mathcal{C}_s)$
 5:     $\mathcal{D}_t \leftarrow filter(\mathcal{D}', \mathcal{C}_s)$
 6:     $\mathcal{C}_a = \{x_i = a_j \mid x_i \in X, a_j \in \mathcal{D}'(x_i)\}$
 7:     $MUS$ = OCUS($\mathcal{X}, \mathcal{D}', \mathcal{C}_s \cup \mathcal{C}_{ns} \cup \mathcal{C}_a$, at_most_one_of($\mathcal{C}_a$))
 8:
 9:     $E \leftarrow \{\mathcal{D}'(x_i) \mid x_i \in scope(MUS)\}$
10:     $S \leftarrow MUS \setminus \mathcal{C}_a$
11:     $\mathcal{D}' \leftarrow \mathcal{D}' \cap filter(\mathcal{D}', S)$
12:     $N \leftarrow \{\mathcal{D}'(x_i) \mid x_i \in scope(MUS)\}$
13:
14:     append $(E, S, N)$ to $seq$
15: end while
16: return $seq$
```