Efficiently Explaining CSPs with Unsatisfiable Subset Optimization

Emilio Gamba[1] , **Bart Bogaerts**[1] and **Tias Guns**[1,2]
[1]Vrije Universiteit Brussel, Belgium
[2]KU Leuven, Belgium
emilio.gamba@vub.be, bart.bogaerts@vub.be, tias.guns@kuleuven.be

Abstract

We build on a recently proposed method for explaining solutions of constraint satisfaction problems. An explanation here is a *sequence* of simple inference steps, where the simplicity of an inference step is measured by the number and types of constraints and facts used, and where the sequence explains all logical consequences of the problem. We build on these formal foundations and tackle two emerging questions, namely how to generate explanations that are provably optimal (with respect to the given cost metric) and how to generate them efficiently. To answer these questions, we develop 1) an implicit hitting set algorithm for finding *optimal* unsatisfiable subsets; 2) a method to reduce multiple calls for (optimal) unsatisfiable subsets to a single call that takes *constraints* on the subset into account, and 3) a method for re-using relevant information over multiple calls to these algorithms. The method is also applicable to other problems that require finding cost-optimal unsatisfiable subsets. We specifically show that this approach can be used to effectively find sequences of *optimal* explanation steps for constraint satisfaction problems like logic grid puzzles.

1 Introduction

Building on old ideas to explain domain-specific propagations performed by constraint solvers [Sqalli and Freuder, 1996; Freuder *et al.*, 2001], we recently introduced a method that takes as input a satisfiable constraint program and explains the solution-finding process in a human-understandable way [Bogaerts *et al.*, 2020]. Explanations in that work are sequences of simple inference steps, involving as few constraints and facts as possible. The explanation-generation algorithms presented in that work rely heavily on calls for *Minimal Unsatisfiable Subsets* (MUS) [Marques-Silva, 2010] of a derived program, exploiting a one-to-one correspondence between so-called *non-redundant explanations* and MUSs. The explanation steps in the seminal work are heuristically optimized with respect to a given cost function that should approximate human-understandability, e.g., taking the number of constraints and facts into account, as well as a valuation of their complexity (or priority). The algorithm developed in that work has two main weaknesses: first, it provides no guarantees on the quality of the produced explanations due to internally relying on the computation of $\subseteq$-minimal unsatisfiable subsets, which are often suboptimal with respect to the given cost function. Secondly, it suffers from performance problems: the lack of optimality is partly overcome by calling a MUS algorithm on increasingly larger subsets of constraints for each candidate implied fact. However, using multiple MUS calls per literal in each iterations quickly causes efficiency problems, causing the explanation generation process to take several hours.

Motivated by these observations, we develop algorithms that aid explaining CSPs and improve the state-of-the-art in the following ways:

- We develop algorithms that compute (cost-)**Optimal** Unsatisfiable Subsets (from now on called OUSs) based on the well-known hitting-set duality that is also used for computing cardinality-minimal MUSs [Ignatiev *et al.*, 2015; Saikko *et al.*, 2016].
- We observe that many of the individual calls for MUSs (or OUSs) can actually be replaced by a single call that searches for an optimal unsatisfiable subset **among subsets satisfying certain structural constraints**. In other words, we introduce the *Optimal **Constrained** Unsatisfiable Subsets (OCUS)* problem and we show how $O(n^2)$ calls to MUS/OUS can be replaced by $O(n)$ calls to an OCUS oracle, where n denotes the number of facts to explain.
- Finally, we develop techniques for **optimizing** the O(C)US algorithms further, exploiting domain-specific information coming from the fact that we are in the *explanation-generation context*. One such optimization is the development of methods for **information re-use** between consecutive OCUS calls.

In this paper, we apply our OCUS algorithms to generate *step-wise* explanations of satisfaction problems. However, MUSs have been used in a variety of contexts, and in particular lie at the foundations of several explanation techniques [Junker, 2001; Ignatiev *et al.*, 2019; Espasa *et al.*, 2021]. We conjecture that OCUS can also prove useful in those settings, to take more fine-grained control over which MUSs, and eventually, which explanations are produced.

The rest of this paper is structured as follows. We discuss background on the hitting-set duality in Section 2. Section 3 motivates our work, while Section 4 introduces the OCUS problem and a generic hitting set–based algorithm for computing OCUSs. In Section 5 we show how to optimize this computation in the context of explanations and in Section 6 we experimentally validate the approach. We discuss related work in Section 7 and conclude in Section 8.

2 Background

We present all methods using propositional logic but our results easily generalize to richer languages, such as constraint languages, as long as the semantics is given in terms of a satisfaction relation between expressions in the language and possible states of affairs (assignments of values to variables).

Let Σ be a set of propositional symbols, also called *atoms*; this set is implicit in the rest of the paper. A *literal* is an atom p or its negation $\neg p$. A clause is a disjunction of literals. A formula $\mathcal{F}$ is a conjunction of clauses. Slightly abusing notation, a clause is also viewed as a set of literals and a formula as a set of clauses. We use the term clause and constraint interchangeably. A (partial) interpretation is a consistent (not containing both p and $\neg p$) set of literals. Satisfaction of a formula $\mathcal{F}$ by an interpretation is defined as usual [Biere *et al.*, 2009]. A *model* of $\mathcal{F}$ is an interpretation that satisfies $\mathcal{F}$; $\mathcal{F}$ is said to be *unsatisfiable* if it has no models. A literal l is a *consequence* of a formula $\mathcal{F}$ if l holds in all $\mathcal{F}$'s models. If I is a set of literals, we write $\overline{I}$ for the set of literals $\{\neg l \mid l \in I\}$.

Definition 1. *A* Minimal Unsatisfiable Subset *(MUS) of $\mathcal{F}$ is an unsatisfiable subset $\mathcal{S}$ of $\mathcal{F}$ for which every strict subset of $\mathcal{S}$ is satisfiable. $MUSs(\mathcal{F})$ denotes the set of MUSs of $\mathcal{F}$.*

Definition 2. *A set $\mathcal{S} \subseteq \mathcal{F}$ is a* Maximal Satisfiable Subset *(MSS) of $\mathcal{F}$ if $\mathcal{S}$ is satisfiable and for all $\mathcal{S}'$ with $\mathcal{S} \subsetneq \mathcal{S}' \subseteq \mathcal{F}$, $\mathcal{S}'$ is unsatisfiable.*

Definition 3. *A set $\mathcal{S} \subseteq \mathcal{F}$ is a* correction subset *of $\mathcal{F}$ if $\mathcal{F} \setminus \mathcal{S}$ is satisfiable. Such a $\mathcal{S}$ is a* minimal correction subset *(MCS) of $\mathcal{F}$ if no strict subset of $\mathcal{S}$ is also a correction subset. $MCSs(\mathcal{F})$ denotes the set of MCSs of $\mathcal{F}$.*

Each MCS of $\mathcal{F}$ is the complement of an MSS of $\mathcal{F}$ and vice versa.

Definition 4. *Given a collection of sets $\mathcal{H}$, a hitting set of $\mathcal{H}$ is a set h such that $h \cap C \neq \emptyset$ for every $C \in \mathcal{H}$. A hitting set is* minimal *if no strict subset of it is also a hitting set.*

The next proposition is the well-known hitting set duality [Liffiton and Sakallah, 2008; Reiter, 1987] between MCSs and MUSs that forms the basis of our algorithms, as well as algorithms to compute MSSs [Davies and Bacchus, 2013] and *cardinality-minimal* MUSs [Ignatiev *et al.*, 2015].

Proposition 5. *A set $\mathcal{S} \subseteq \mathcal{F}$ is an MCS of $\mathcal{F}$ iff it is a* minimal hitting set *of $MUSs(\mathcal{F})$. A set $\mathcal{S} \subseteq \mathcal{F}$ is a MUS of $\mathcal{F}$ iff it is a* minimal hitting set *of $MCSs(\mathcal{F})$.*

3 Motivation

Our work is motivated by the problem of explaining satisfaction problems through a sequence of simple explanation steps. This can be used to teach people problem-solving skills, to compare the difficulty of related satisfaction problems (through the number and complexity of steps needed), and in human-computer solving assistants.

Algorithm 1: EXPLAIN-ONE-STEP$(\mathcal{C}, f, I, I_{end})$

```
1 $X_{best} \leftarrow nil$
2 for $l \in \{I_{end} \setminus I\}$ do
3     $X \leftarrow \text{MUS}(\mathcal{C} \wedge I \wedge \neg l)$
4     if $f(X) < f(X_{best})$ then
5         $X_{best} \leftarrow X$
6     end
7 end
8 return $X_{best}$
```

Our original explanation generation algorithm [Bogaerts *et al.*, 2020] starts from a formula $\mathcal{C}$ (in the application coming from a high level CSP), a partial interpretation I (here also viewed as a conjunction of literals) and a cost function f quantifying the difficulty of an explanation step, by means of a weight for every clause and literal in $\mathcal{F}$.

The goal is to find a sequence of *simple* explanation steps, where the simplicity of a step is measured by the total cost of the elements used in the explanation. An explanation step is an implication $I' \wedge \mathcal{C}' \implies N$ where I' is a subset of already derived literals, $\mathcal{C}'$ is a subset of constraints of the input formula $\mathcal{C}$, and N is a set of literals entailed by I' and $\mathcal{C}'$ which are not yet explained.

The key part of the algorithm is the search for the next best explanation, given an interpretation I derived so far. Algorithm 1 shows the gist of how this was done. It takes as input the formula $\mathcal{C}$, a cost function f quantifying the quality of explanations, an interpretation I containing all already derived literals in the sequence so far, and the interpretation-to-explain I_{end}. To compute an explanation, this procedure iterates over the literals that are still to explain, computes for each of them an associated MUS and subsequently selects the lowest cost one from found MUSs. The reason this works is because there is a one-to-one correspondence between MUSs of $\mathcal{C} \wedge I \wedge \neg l$ and so-called *non-redundant explanation* of l in terms of (subsets of) $\mathcal{C}$ and I [Bogaerts *et al.*, 2020].

Experiments have shown that such a MUS-based approach can easily take hours, especially when multiple MUS calls are performed to increase the chance of finding a good MUS, and hence that algorithmic improvements are needed to make it more practical. We see three main points of improvement, all of which will be tackled by our generic OCUS algorithm presented in the next section.

- First of all, since the algorithm is based on MUS calls, there is no guarantee that the explanation found is indeed optimal (with respect to the given cost function). Performing multiple MUS calls is only a heuristic that is used to circumvent the restriction that *there are no algorithms for cost-based unsatisfiable subset* ***optimization***.
- Second, this algorithm uses MUS calls for every literal to explain separately. The goal of all these calls is to find a single unsatisfiable subset of $\mathcal{C} \wedge I \wedge \overline{(I_{end} \setminus I)}$ that contains exactly one literal from $\overline{(I_{end} \setminus I)}$. This begs the

questions whether it is possible *to compute a single (optimal) unsatisfiable subset **subject to constraints***, where in our case, the constraint is to include exactly one literal from $\overline{(I_{end} \setminus I)}$.

- Finally, the algorithm that computes an entire explanation sequence makes use of repeated calls to EXPLAIN-ONE-STEP and hence will solve many similar problems. This raises the issue of ***incrementality**: can we re-use the computed data structures to achieve speed-ups in later calls?*

4 Optimal Constrained Unsatisfiable Subsets

The first two considerations from the previous section lead to the following definition.

Definition 6. *Let $\mathcal{F}$ be a formula, $f : 2^{\mathcal{F}} \to \mathbb{N}$ a cost function and p a predicate $p : 2^{\mathcal{F}} \to \{true, false\}$. We call $\mathcal{S} \subseteq \mathcal{F}$ an OCUS of $\mathcal{F}$ (with respect to f and p) if*

- *$\mathcal{S}$ is unsatisfiable,*
- *$p(\mathcal{S})$ is true*
- *all other unsatisfiable $\mathcal{S}' \subseteq \mathcal{F}$ for which $p(\mathcal{S}')$ is true satisfy $f(\mathcal{S}') \geq f(\mathcal{S})$.*

If we assume that the predicate p is specified itself as a CNF over (meta)-variables indicating inclusion of clauses of $\mathcal{F}$, and f is obtained by assigning a weight to each such meta-variable, then the complexity of the problem of finding an OCUS is the same as that of the SMUS (cardinality-minimal MUS) problem [Ignatiev *et al.*, 2015]: the associated decision problem is Σ_2^P-complete. Hardness follows from the fact that SMUS is a special case of OCUS, containment follows - intuitively - from the fact that this can be encoded as an $\exists\forall$-QBF using a Boolean circuit encoding of the costs.

When considering the procedure EXPLAIN-ONE-STEP from the perspective of OCUS defined above, the task of the procedure is to compute an OCUS of the formula $\mathcal{F} := \mathcal{C} \land I \land \overline{I_{end} \setminus I}$ with p the predicate that holds for subsets that contain exactly one literal of $\overline{I_{end} \setminus I}$, see Algorithm 2.

In order to compute an OCUS of a given formula, we propose to build on the hitting set duality of Proposition 5. For this, we will assume to have access to a solver CONDOPTHITTINGSET that can compute hitting sets of a given collection of sets that are *optimal* (w.r.t. a given cost function f) among all hitting sets *satisfying a condition p*. The choice of the underlying hitting set solver will thus determine which types of cost functions and constraints are possible. In our implementation, we use a cost function f as well as a condition p that can easily be encoded as linear constraints, thus allowing the use of highly optimized mixed integer programming (MIP) solvers. The CONDOPTHITTINGSET formulation is as follows:

$$\begin{aligned}
minimize_S \quad & f(S) \\
s.t. \quad & p(S) \\
& sum(H) \geq 1, \qquad \forall H \in \mathcal{H} \\
& s \in \{0,1\}, \qquad \forall s \in S
\end{aligned}$$

where S is a set of MIP decision variables, one for every clause in $\mathcal{F}$. In our case, p is expressed as $\sum_{s \in \overline{I_{end} \setminus I}} s = 1$. f is a weighted sum over the variables in S. For example, (unit) clauses representing previously derived facts can be given small weights and regular constraints can be given large weights, such that explanations are penalized for including constraints when previously derived facts can be used instead.

Algorithm 2: EXPLAIN-ONE-STEP-OCUS($\mathcal{C}, f, I, I_{end}$)

```
1 $p \leftarrow$ exactly one of $\overline{I_{end} \setminus I}$
2 return OCUS($\mathcal{C} \land I \land \overline{I_{end} \setminus I}, f, p$)
```

Algorithm 3: OCUS($\mathcal{F}, f, p$)

```
1 $\mathcal{H} \leftarrow \emptyset$
2 while true do
3     $\mathcal{S} \leftarrow$ CONDOPTHITTINGSET($\mathcal{H}, f, p$)
4     if $\neg$SAT($\mathcal{S}$) then
5         return $\mathcal{S}$
6     end
7     $\mathcal{S} \leftarrow$ GROW($\mathcal{S}, \mathcal{F}$)
8     $\mathcal{H} \leftarrow \mathcal{H} \cup \{\mathcal{F} \setminus \mathcal{S}\}$
9 end
```

Our generic algorithm for computing OCUSs is depicted in Algorithm 3. It combines the hitting set-based approach for MUSs of [Ignatiev *et al.*, 2015] with the use of a MIP solver for (weighted) hitting sets as proposed for maximum satisfiability [Davies and Bacchus, 2013]. The key novelty is the ability to add structural constraints to the hitting set solver, without impacting the duality principles of Proposition 5, as we will show.

Ignoring Line 7 for a moment, the algorithm alternates calls to a hitting set solver with calls to a SAT oracle on a subset $\mathcal{S}$ of $\mathcal{F}$. In case the SAT oracle returns true, i.e., the subset $\mathcal{S}$ is satisfiable, the complement of $\mathcal{S}$ is a correction subset of $\mathcal{F}$ and is added to $\mathcal{H}$.

As in the SMUS algorithm of Ignatiev *et al.* [2015], our algorithm contains an (optional) call to GROW. The purpose of the GROW is to expand a satisfiable subset of $\mathcal{F}$ further, to find a smaller correction subset and as such find stronger constraints on the hitting sets. In our case, the calls for hitting sets will also take into account the cost (f), as well as the meta-level constraints (p); as such, it is not clear a priori which properties a good GROW function should have here. We discuss the different possible implementations of GROW later and evaluate their performance in Section 6. For correctness of the algorithm, all we need to know is that it returns a satisfiable subset $\mathcal{S}'$ of $\mathcal{F}$ with $\mathcal{S} \subseteq \mathcal{S}'$.

Soundness and completeness of the proposal follow from the fact that all sets added to $\mathcal{H}$ are correction subsets, and Theorem 7, which states that what is returned is indeed a solution and that a solution will be found if it exists.

Theorem 7. *Let $\mathcal{H}$ be a set of correction subsets of $\mathcal{F}$. If $\mathcal{S}$ is a hitting set of $\mathcal{H}$ that is f-optimal among the hitting sets of $\mathcal{H}$ satisfying a predicate p, and $\mathcal{S}$ is unsatisfiable, then $\mathcal{S}$ is an OCUS of $\mathcal{F}$.*

If $\mathcal{H}$ has no hitting sets satisfying p, then $\mathcal{F}$ has no OCUSs.

	$\mathcal{S}$	SAT($\mathcal{S}$)	GROW ($\mathcal{S}, \mathcal{F}$)	$\mathcal{H} \leftarrow \mathcal{H} \cup \{\mathcal{F} \setminus \mathcal{S}\}$
1	$\emptyset$	*true*	$\{c_1, c_2, c_3, c_4, c_5\}$	$\{\{c_6, c_7\}\}$
2	$\{c_6\}$	*true*	$\{c_1, c_2, c_3, c_5, c_6\}$	$\{\{c_6, c_7\}, \{c_4, c_7\}\}$
3	$\{c_7\}$	*true*	$\{c_1, c_3, c_4, c_5, c_7\}$	$\{\{c_6, c_7\}, \{c_4, c_7\}, \{c_2, c_6\}\}$
4	$\{c_2, c_7\}$	*true*	$\{c_2, c_3, c_4, c_5, c_6, c_7\}$	$\{\{c_6, c_7\}, \{c_4, c_7\}, \{c_2, c_6\}, \{c_1\}\}$
5	$\{c_1, c_2, c_7\}$	*true*	$\{c_1, c_2, c_4, c_6, c_7\}$	$\{\{c_6, c_7\}, \{c_4, c_7\}, \{c_2, c_6\}, \{c_1\}, \{c_3, c_5\}\}$
6	$\{c_1, c_2, c_5, c_7\}$	*false*		

Table 1: Example of an OCUS-explanation computation.

Proof. For the first claim, it is clear that $\mathcal{S}$ is unsatisfiable and satisfies p. Hence all we need to show is f-optimality of $\mathcal{S}$. If there would exist some other unsatisfiable subset $\mathcal{S}'$ that satisfies p with $f(\mathcal{S}') \leq f(\mathcal{S})$, we know that $\mathcal{S}'$ would hit every minimal correction set of $\mathcal{F}$, and hence also every set in $\mathcal{H}$ (since every correction set is the superset of a minimal correction set). Since $\mathcal{S}$ is f-optimal among hitting sets of $\mathcal{H}$ satisfying p and $\mathcal{S}'$ also hits $\mathcal{H}$ and satisfies p, it must thus be that $f(\mathcal{S}) = f(\mathcal{S}')$.

The second claim immediately follows from Proposition 5 and the fact that an OCUS is an unsatisfiable subset of $\mathcal{F}$. $\square$

Perhaps surprisingly, correctness of the proposed algorithm does *not* depend on monotonicity properties of f nor p. In principle, any (computable) cost function and condition on the unsatisfiable subsets can be used. In practice however, one is bound by limitations of the chosen hitting set solver.

As an illustration, we now provide an example of one call to EXPLAIN-ONE-STEP-OCUS (Algorithm 2) and the corresponding OCUS-call (Algorithm 3) in detail:

Example 1. *Let $\mathcal{C}$ be a CNF formula over variables x_1, x_2, x_3 with the following four clauses:*

$$c_1 := \neg x_1 \vee \neg x_2 \vee x_3 \qquad c_2 := \neg x_1 \vee x_2 \vee x_3$$

$$c_3 := x_1 \qquad c_4 := \neg x_2 \vee \neg x_3$$

The final interpretation I_{end} is $\{x_1, \neg x_2, x_3\}$. Let the current interpretation I be $\{x_1\}$, then $\overline{I_{end} \setminus I} = \{x_2, \neg x_3\}$.

To define the input for the OCUS call, we add new clauses representing the already known facts I and the to-be-derived facts $\overline{I_{end} \setminus I}$:

$$c_5 := \{x_1\} \qquad c_6 := \{x_2\} \qquad c_7 := \{\neg x_3\}$$

The formula $\mathcal{F}$ in the OCUS*-call is thus:*

$$\mathcal{F} = \mathcal{C} \wedge I \wedge \overline{(I_{end} \setminus I)} = \{c_1 \wedge c_2 \wedge c_3 \wedge c_4 \wedge c_5 \wedge c_6 \wedge c_7\}$$

We define $p \triangleq$ exactly-one(c_6, c_7) and $f = \sum w_i c_i$ with clause weights $w_1 = 60, w_2 = 60, w_3 = 100, w_4 = 100, w_5 = 1, w_6 = 1, w_7 = 1$.

$\mathcal{H}$ is initialized as the empty set. At each iteration, the hitting set solver will search for a cost-minimal assignment that hits all sets in $\mathcal{H}$ and that furthermore contains exactly one of c_6 and c_7 (due to p). Table 1 shows the computed steps in the different iterations of Algorithm 3 given the above input. In this example, the GROW we used is the one called *Max-Actual-Unif* in Section 6.

5 Efficient OCUS Computation for Explanations

Algorithm 3 is generic and can also be used to find (unconstrained) OUSs, namely with a trivially true p. However, its constrainedness property allows to remove the need to compute a MUS/OUS for every literal. This decreases the complexity of explanation sequence generation from $O(n^2)$ calls to MUS to $O(n)$ calls to OCUS, namely, once for every step in the sequence.

We now discuss optimizations to the OCUS algorithm that are specific to explanation sequence generation, though they can also be used when other forms of domain knowledge are present.

Incremental OCUS Computation Inherently, generating a sequence of explanations still requires as many OCUS calls as there are literals to explain. Indeed, a greedy sequence construction algorithm calls EXPLAIN-ONE-STEP-OCUS iteratively with a growing interpretation I until $I = I_{end}$.

All of these calls to EXPLAIN-ONE-STEP-OCUS, and hence OCUS, are done with very similar input (the set of constraints does not change, and the I slowly grows between two calls). For this reason, it makes sense that information computed during one of the earlier stages can be useful in later stages as well.

The main question is, suppose two OCUS calls are done, first with inputs $\mathcal{F}_1$, f_1, and p_1, and later with $\mathcal{F}_2$, f_2, and p_2; how can we make use as much as possible of the data computations of the first call to speed-up the second call? The answer is surprisingly elegant. The most important data OCUS keeps track of is the collection $\mathcal{H}$ of correction subsets that need to be hit.

This collection in itself is not useful for transfer between two calls, since – unless we assume that $\mathcal{F}_2$ is a subset of $\mathcal{F}_1$, there is no reason to assume that a set in $\mathcal{H}_1$ should also be hit in the second call. However, each set H in $\mathcal{H}$ is the complement (with respect to the formula at hand) of a *satisfiable subset* of constraints, and this satisfiability remains true. Thus, instead of storing $\mathcal{H}$, we can keep track of a set **SSs** of *satisfiable subsets* (the sets $\mathcal{S}$ in the OCUS algorithm). When a second call to OCUS is performed, we can then initialize $\mathcal{H}$ as the complement of each of these satisfiable subsets with respect to $\mathcal{F}_2$, i.e.,

$$\mathcal{H} \leftarrow \{\mathcal{F}_2 \setminus \mathcal{S} \mid \mathcal{S} \in \mathbf{SSs}\}.$$

The effect of this is that we *bootstrap* the hitting set solver with an initial set $\mathcal{H}$.

For hitting set solvers that natively implement incrementality, we can generalize this idea further: we know that all calls to OCUS$(\mathcal{F}, f, p)$ will be cast with $\mathcal{F} \subseteq \mathcal{C} \cup I_{end} \cup \overline{I_{end} \setminus I_0}$, where I_0 is the start interpretation. Since our implementation uses a MIP solver for computing hitting sets (see Section 2), and we have this upper bound on the set of formulas to be used, we can initialize all relevant decision variables once. To compute the conditional hitting set for a specific $\mathcal{C} \cup I \cup \overline{I_{end} \setminus I} \subseteq \mathcal{C} \cup I_{end} \cup \overline{I_{end} \setminus I_0}$ we need to ensure that the MIP solver only uses literals in $\mathcal{C} \cup I \cup \overline{I_{end} \setminus I}$, for example by giving all other literals infinite weight in the cost function. In this way, the MIP solver will automatically maintain and reuse previously found sets-to-hit in each of its computations.

Explanations with Bounded OUS Instead of working OCUS-based, we can now also generate optimal explanations by replacing the MUS call by an OUS call in Algorithm 1 (where OUS is computed as in Algorithm 3, but with a trivially true p). When doing this, we know that every OCUS of cost greater than or equal to $f(X_{best})$ will be discarded by the check on Line 4 of Algorithm 1. As such, a next optimization is to, instead of searching for an OUS, perform a *bounded OUS check*, which only computes an OUS in case one of cost smaller than a given bound ub exists. In our specific implementation, bounded OUS is performed by interrupting this OUS-call (after Line 3 Algorithm 3) if $f(\mathcal{S}) > ub$.

Since the bounding on the OUS cost has the most effect if cheap OUSs are found early in the loop across the different literals, we keep track of an upper bound of the cost of an OUS for each literal to explain. This is initialized to a value greater than any OUS, e.g., as $f(\mathcal{C} \wedge I_0 \wedge \overline{I_{end} \setminus I_0})$, and is updated every time an OUS explaining that literal is found; when going through the loop in Line 2 of Algorithm 3, we then handle literals in order of increasing upper bounds.

Domain-Specific Implementations of GROW The goal of the GROW procedure is to turn $\mathcal{S}$ into a larger subformula of $\mathcal{F}$. The effect of this is that the complement added to $\mathcal{H}$ will be smaller, and hence, a stronger restriction for the hitting set solver is found.

Choosing an effective GROW procedure requires finding a difficult balance: on the one hand, we want our subformula to be as large as possible (which ultimately would correspond to computing the maximum satisfiable subformula), but on the other hand we also want the procedure to be very efficient as it is called in every iteration.

For the case of explanations we are in, we make the following observations:

- Our formula at hand (using the notation from the EXPLAIN-ONE-STEP-OCUS algorithm) consists of three types of clauses: **(i)** (translations of) the problem constraints (this is $\mathcal{C}$) **(ii)** literals representing the assignment found thus far (this is I), and **(iii)** the negations of literals not-yet-derived (this is $\overline{I_{end} \setminus I}$).
- $\mathcal{C}$ and I together are satisfiable, with assignment I_{end}, and *mutually supportive*, by this we mean that making more constraints in $\mathcal{C}$ true, more literals in I will automatically become true and vice versa.
- The constraint p enforces that each hitting set will contain **exactly** one literal of $\overline{I_{end} \setminus I}$

Since the restriction on the third type of elements of $\mathcal{F}$ are already strong, it makes sense to use the GROW ($\mathcal{S}$,$\mathcal{F}$) procedure to search for a *maximal* satisfiable subset of $\mathcal{C} \cup I$ with hard constraints that $\mathcal{S}$ should be satisfied, using a call to an efficient (partial) MaxSAT solver. Furthermore, we can initialize this call as well as any call to a SAT solver with the polarities for all variables set to the value they take in I_{end}.

We evaluate different grow strategies in the experiments section, including the use of partial MaxSAT as well as weighted partial MaxSAT based on the weights in the cost function f.

Example 1 (cont.) Consider line 0 in table 1. During the GROW procedure, the MaxSAT solver *Max-Actual-Unif* with polarities set to I_{end} branches when multiple assignment to a literal are possible. By hinting the polarities of the literals, we guide the solver and it assigns all values according to the end interpretation and neither c_6 nor c_7 is taken.

6 Experiments

We now experimentally validate the performance of the different versions of our algorithm. Our benchmarks were run on a compute cluster, where each explanation sequence generation was assigned a single core on a 10-core INTEL Xeon Gold 61482 (Skylake) processor, a timelimit of 120 minutes and a memory-limit of 4GB. Everything was implemented in Python on top of PySAT[1] and is available at https://github.com/ML-KULeuven/ocus-explain. For MIP calls, we used Gurobi 9.0, for SAT calls MiniSat 2.2 and for MaxSAT calls RC2 as bundled with PySAT (version 0.1.6.dev11). In the MUS-based approach we used PySAT's deletion-based MUS extractor MUSX [Marques-Silva, 2010].

All of our experiments were run on a direct translation to PySAT of the 10 puzzles of Bogaerts *et al.* [2020][2]. We used a cost of 60 for puzzle-agnostic constraints; 100 for puzzle-specific constraints; and cost 1 for facts. When generating an explanation sequence for such puzzles, the unsatisfiable subset identifies which constraints and which previously derived facts should be combined to derive new information. Our experiments are designed to answer the following research questions:

Q1 What is the effect of requiring optimality of the generated MUSs on the **quality** of the generated explanations?

Q2 Which **domain-specific GROW methods** perform best?

Q3 What is the effect of the use of **constrainedness** on the time required to compute an explanation sequence?

Q4 Does **re-use** of information across the different iterations improve efficiency?

Explanation quality To evaluate the effect of optimality on the quality of the generated explanations, we reimplemented a MUS-based explanation generator based on Algorithm 1. Before presenting the results, we want to stress that this is *not* a fair comparison with the implementation of Bogaerts *et al.* [2020], since there – in order to avoid the quality problems we will illustrate below – an extra inner loop was used that employs *even more* calls to MUS for a selected set of subsets of $\mathcal{C}$ of increasing size. While this yields better explanations, it comes at the expense of computation time, thereby leading to several hours to generate the explanation of a single puzzle.

To answer **Q1**, we ran the MUS-based algorithm as described in Algorithm 1 and compared at every step the cost of the produced explanation with the cost of the optimal explanation. These costs are plotted on a heatmap in Figure 1,

[1]https://pysathq.github.io

[2]In one of the puzzles, an error in the automatic translation of the natural language constraints was found and fixed.

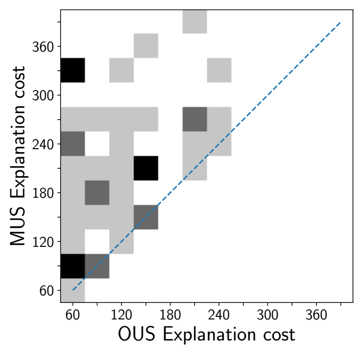

Figure 1: Q1 - Explanation quality comparison of optimal versus subset-minimal explanations in the generated puzzle explanation sequences.

where the darkness represents the number of occurrences of the combination at hand. We see that the difference in quality is striking in many cases, with the MUS-based solution often missing very cheap explanations (as seen by the two dark squares in the column around cost 60), thereby confirming the need for a cost-based OUS/OCUS approach.

Domain-specific GROW In our OCUS algorithm, we do not just aim to find any satisfiable subsets, but we prefer *high quality* satisfiable subsets: subsets that impose strong constraints on the assignments the optimal hitting set solver can find. This induces a trade-off between *efficiency* of the GROW strategy and *quality* of the produced satisfiable subset.

Thus, to answer **Q2**, we compared variations of OCUS that only differ in which GROW strategy they use. Figure 2 depicts the average (over all the puzzles) cumulative explanation time to derive a number of literals. Note that most puzzles only contain 150 literals, except for 2, which contain 96 and 250 literals respectively. When a method times out for a puzzle at one step, a runtime value of 7200 is used in computing the averages for all future steps. The configurations used are as follows:

- *Max* refers to growing with a MaxSAT solver and *Greedy* to growing using a heuristic method implemented by repeated sat calls, while *no-grow* refers to skipping the GROW step.
- *Full* refers to using the full unsatisfiable formula $\mathcal{F}$ while *Actual* refers to using only the constraints that hold in the final interpretation (see Section 5). For instance, for the MaxSAT-based calls, *Actual* means that only the previously derived facts and the original constraints are taken into account when computing optimality.
- The MaxSAT solver (*Max*) is combined with different weighing schemes: uniform weights (*unif*), cost-function weights (*pos*) (equal to the weights in f), or the inverse of these costs (*inv*) defined as $\max_j(w_j)+1-w_i$.

We can observe that not using a grow strategy performs badly, as do weighted MaxSAT grows with costs w_i (-Pos). Greedy grow strategies improve on not using a grow strategy, but not substantially. The two approaches that work best use the domain-specific knowledge of doing a MaxSAT grow on

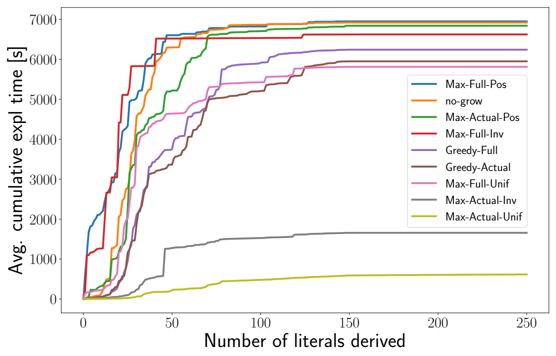

Figure 2: Q2 - Explanation specific GROW strategies for OCUS.

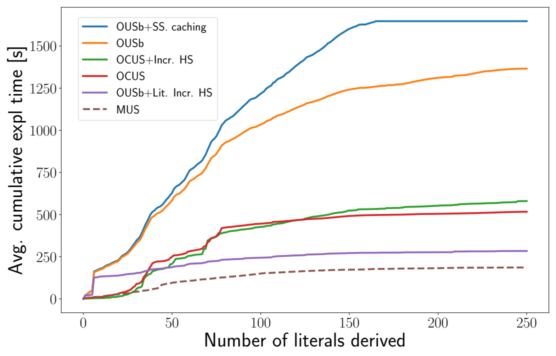

Figure 3: Q3 - Cumulative runtime evolution enhancements on incrementality and constrainedness.

$\mathcal{C} \cup I$, with the unweighted variant the only one that never times out.

Constrainedness and incrementality To answer **Q3** and **Q4**, we compare the effect of constrainedness in the search for explanations (C) and incrementality. Next to OCUS, we also include the bounded OUS approach (OUSb), where we call the OUS algorithm for every literal in every step, but we reuse information by giving it the current best bound $f(X_{best})$ and iterating over the literals that performed best in the previous call first. Based on the previous experiment, we always use (both for OUSb and OCUS) *Max-Actual-Unif* as grow strategy. For *OCUS*, incrementality (*+Incr. HS*) is achieved by reusing the same incremental MIP hitting set solver throughout the explanation calls, as explained in Section 5. To have a better view on how incrementality also affects OUSb, we add incrementality to it in the following ways:

- *SS. caching* keeps track of the satisfiable subsets, which are used to initialize $\mathcal{H}$ for a fresh hitting set solver instance each time.
- *Lit. Incr. HS* uses a separate incremental hitting set solver for every literal to explain, throughout the explanation calls. Once the literal is explained, the hitting set solver is discarded.

Figure 3 shows the results, where In this figure, the configurations are compared in a similar fashion to Figure 2.

When comparing the configurations, we see that the plain *MUS*-based implementation is faster than the *O(C)US* implementations, as it solves a simpler problem (with worse quality results as shown in **Q1**). Replacing MUS by bounded OUS calls (orange line) leads to a much larger computation cost. The generic SS caching technique adds additional overhead.

The OCUS variants significantly improve runtime compared to those two bounded OUS approaches, by reducing the number of OUS calls. For OCUS, using an incremental hitting set solver across all steps seems to be slightly faster for deriving literals earlier in the sequence, while inducing a small overhead for the literals at the end of the sequence.

When looking at the runtime to explain the entire sequence, best results are obtained with OUSb + Lit. Incr. HS, that is, using an incremental hitting set solver *for every individual literal to explain*. However, for the first literals, we can see that it takes much more computation time and that reducing the number of OUS calls from n to 1 per explanation step improves runtime (OCUS). However, making each of the n calls incremental and bounded across the entire explanation sequence generation process leads to an even faster process overall (OUSb+Lit. Incr. HS).

7 Related Work

In the last few years, driven by the increasingly many successes of Artificial Intelligence (AI), there is a growing need for **eXplainable Artificial Intelligence (XAI)** [Miller, 2019]. In the research community, this need manifests itself through the emergence of (interdisciplinary) workshops and conferences on this topic [Miller *et al.*, 2019; Hildebrandt *et al.*, 2020] and American and European incentives to stimulate research in the area [Gunning, 2017; Hamon *et al.*, 2020; FET, 2019].

While the main focus of XAI research has been on explaining black-box machine learning systems [Lundberg and Lee, 2017; Guidotti *et al.*, 2018; Ignatiev *et al.*, 2019], also model-based systems, which are typically considered more transparent, are in need of explanation mechanisms. Indeed, by advances in solving methods in research fields such as constraint programming [Rossi *et al.*, 2006] and SAT [Biere *et al.*, 2009], as well as by hardware improvement, such systems now easily consider millions of alternatives in short amounts of time. Because of this complexity, the question arises how to generate human-interpretable explanations of the conclusions they make. Explanations for model-based systems have been considered mostly for explain *unsatisfiable* problem instances [Junker, 2001], and have recently seen a rejuvenation in various subdomains of constraint reasoning [Fox *et al.*, 2017; Čyras *et al.*, 2019; Chakraborti *et al.*, 2017; Bogaerts *et al.*, 2020].

In this context, we recently introduced *step-wise explanations* [Bogaerts *et al.*, 2020] and applied them to Zebra puzzles; similar explanations, but for a wider range of puzzles, have been investigated by Espasa *et al.* [2021]. Our current work is motivated by a concrete algorithmic need: to generate these explanations efficiently, we need algorithms that can find optimal MUSs with respect to a given cost function, where the cost function approximates human-understandability of the corresponding explanation step. The closest related works can be found in the literature on generating or enumerating MUSs [Lynce and Silva, 2004; Liffiton *et al.*, 2016]. Different techniques are employed to find MUSs, including manipulating resolution proofs produced by SAT solvers [Goldberg and Novikov, 2003; Gershman *et al.*, 2008; Dershowitz *et al.*, 2006], incremental solving to enable/disable clauses and branch-and-bound search [Oh *et al.*, 2004], or by BDD-manipulation methods [Huang, 2005]. Other methods work by means of translation into a so-called Quantified MaxSAT [Ignatiev *et al.*, 2016], a field that combines the expressivity of Quantified Boolean Formulas (QBF) [Kleine Büning and Bubeck, 2009] with optimization as known from MaxSAT [Li and Manyà, 2009], or by exploiting the so-called hitting set duality [Ignatiev *et al.*, 2015] bootstrapped using MCS-enumeration [Marques-Silva *et al.*, 2020]. An *abstract* framework for describing hitting set–based algorithms, including optimization was developed by Saikko *et al.* [2016]. While our approach can be seen to fit the framework, the terminology is focused on MaxSAT rather than MUS and would complicate our exposition. To the best of our knowledge, only few have considered *optimizing* MUSs: the only criterion considered yet is cardinality-minimality [Lynce and Silva, 2004; Ignatiev *et al.*, 2015].

8 Conclusion, Challenges and Future work

We presented a hitting set–based algorithm for finding *optimal constrained* unsatisfiable subsets, with an application in generating explanation sequence for constraint satisfaction problems. We extended our methods with *incrementality*, as well as with a *domain-specific method for extending satisfiable subsets (*GROW*)*. This domain-specific GROW method was key to generating explanation sequences in a reasonable amount of time. We noticed that, independently, incrementality and constrainedness have major benefits on explanation-generation time. The best method on the tested puzzles was the incremental, bounded but non-constrained variant. It remains an open question how to make constrainedness and incrementality work together more effectively, as well as how to further optimize O(C)US-based explanations, for instance using disjoint MCS enumeration [Marques-Silva *et al.*, 2020].

With the observed impact of different 'GROW' methods, an open question remains whether we can quantify precisely and in a generic way what a *good* or even the best set-to-hit is in a hitting set approach. The synergies of our approach with the more general problem of QMaxSAT [Ignatiev *et al.*, 2016] is another open question.

The concept of bounded (incremental) OUS and OCUS are not limited to explanations of satisfaction problems and we are keen to explore other applications too. A general direction here are explanations of *optimisation* problems and the role of the objective function in explanations.

Acknowledgments

This research received partial funding from the Flemish Government (AI Research Program); the FWO Flanders project

G070521N; and funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (Grant No. 101002802, CHAT-Opt).

References

[Biere *et al.*, 2009] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh. *Handbook of Satisfiability*. 2009.

[Bogaerts *et al.*, 2020] Bart Bogaerts, Emilio Gamba, Jens Claes, and Tias Guns. Step-wise explanations of constraint satisfaction problems. In *Proceedigns of ECAI*, pages 640–647, 2020.

[Chakraborti *et al.*, 2017] Tathagata Chakraborti, Sarath Sreedharan, Yu Zhang, and Subbarao Kambhampati. Plan explanations as model reconciliation: moving beyond explanation as soliloquy. In *Proceedings of IJCAI*, pages 156–163, 2017.

[Čyras *et al.*, 2019] Kristijonas Čyras, Dimitrios Letsios, Ruth Misener, and Francesca Toni. Argumentation for explainable scheduling. In *Proceedings of AAAI*, pages 2752–2759, 2019.

[Davies and Bacchus, 2013] Jessica Davies and Fahiem Bacchus. Exploiting the power of MIP solvers in MAXsat. In *Proceedings of SAT*, pages 166–181, 2013.

[Dershowitz *et al.*, 2006] Nachum Dershowitz, Ziyad Hanna, and Alexander Nadel. A scalable algorithm for minimal unsatisfiable core extraction. In *Proceedings of SAT*, pages 36–41, 2006.

[Espasa *et al.*, 2021] Joan Espasa, Ian P. Gent, Ruth Hoffmann, Christopher Jefferson, and Alice M. Lynch. Using small muses to explain how to solve pen and paper puzzles. *ArXiv*, abs/2104.15040, 2021.

[FET, 2019] Fetproact-eic-05-2019, fet proactive: emerging paradigms and communities, call, 2019. Horizon 2020 Framework Programme.

[Fox *et al.*, 2017] Maria Fox, Derek Long, and Daniele Magazzeni. Explainable planning. In *Proceedings of IJCAI'17-XAI*, 2017.

[Freuder *et al.*, 2001] Eugene C Freuder, Chavalit Likitvivatanavong, and Richard J Wallace. Explanation and implication for configuration problems. In *IJCAI 2001 workshop on configuration*, pages 31–37, 2001.

[Gershman *et al.*, 2008] Roman Gershman, Maya Koifman, and Ofer Strichman. An approach for extracting a small unsatisfiable core. *Formal Methods in System Design*, 33(1-3):1–27, 2008.

[Goldberg and Novikov, 2003] Evgueni Goldberg and Yakov Novikov. Verification of proofs of unsatisfiability for CNF formulas. In *Proceedings of DATE*, pages 10886–10891, 2003.

[Guidotti *et al.*, 2018] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. A survey of methods for explaining black box models. *ACM computing surveys (CSUR)*, 51(5):1–42, 2018.

[Gunning, 2017] David Gunning. Explainable artificial intelligence (xai). *Defense Advanced Research Projects Agency*, 2, 2017.

[Hamon *et al.*, 2020] Ronan Hamon, Henrik Junklewitz, and Ignacio Sanchez. Robustness and explainability of artificial intelligence. *Publications Office of the European Union*, 2020.

[Hildebrandt *et al.*, 2020] Mireille Hildebrandt, Carlos Castillo, Elisa Celis, Salvatore Ruggieri, Linnet Taylor, and Gabriela Zanfir-Fortuna, editors. *Proceedings of FAT**, 2020.

[Huang, 2005] Jinbo Huang. Mup: A minimal unsatisfiability prover. In *Proceedings of the Asia and South Pacific Design Automation Conference, ASP-DAC*, pages 432– 437 Vol. 1, 2005.

[Ignatiev *et al.*, 2015] Alexey Ignatiev, Alessandro Previti, Mark Liffiton, and Joao Marques-Silva. Smallest MUS extraction with minimal hitting set dualization. In *Proceedings of CP*, 2015.

[Ignatiev *et al.*, 2016] Alexey Ignatiev, Mikolás Janota, and João Marques-Silva. Quantified maximum satisfiability. *Constraints*, 21(2):277–302, 2016.

[Ignatiev *et al.*, 2019] Alexey Ignatiev, Nina Narodytska, and Joao Marques-Silva. Abduction-based explanations for machine learning models. In *Proceedings of AAAI*, pages 1511–1519, 2019.

[Junker, 2001] Ulrich Junker. QuickXPlain: Conflict detection for arbitrary constraint propagation algorithms. In *IJCAI'01 Workshop on Modelling and Solving problems with constraints*, 2001.

[Kleine Büning and Bubeck, 2009] Hans Kleine Büning and Uwe Bubeck. Theory of quantified boolean formulas. In *Handbook of Satisfiability*, pages 735–760. 2009.

[Li and Manyà, 2009] Chu Min Li and Felip Manyà. MaxSAT, hard and soft constraints. In *Handbook of Satisfiability*, pages 613–631. 2009.

[Liffiton and Sakallah, 2008] Mark H. Liffiton and Karem A. Sakallah. Algorithms for computing minimal unsatisfiable subsets of constraints. *J. Autom. Reasoning*, 40(1):1–33, 2008.

[Liffiton *et al.*, 2016] Mark H Liffiton, Alessandro Previti, Ammar Malik, and Joao Marques-Silva. Fast, flexible mus enumeration. *Constraints*, 21(2):223–250, 2016.

[Lundberg and Lee, 2017] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Proceedings of NIPS*, pages 4765–4774, 2017.

[Lynce and Silva, 2004] Inês Lynce and João P. Marques Silva. On computing minimum unsatisfiable cores. In *Proceedings of SAT*, 2004.

[Marques-Silva *et al.*, 2020] Joao Marques-Silva, Carlos Mencía, et al. Reasoning about inconsistent formulas. In *Proceedings of IJCAI, Survey track*, 2020.

[Marques-Silva, 2010] Joao Marques-Silva. Minimal unsatisfiability: Models, algorithms and applications. In *2010 40th IEEE International Symposium on Multiple-Valued Logic*, 2010.

[Miller *et al.*, 2019] Tim Miller, Rosina Weber, and Daniele Magazzeni, editors. *Proceedings of the IJCAI 2019 Workshop on Explainable Artificial Intelligence*, 2019.

[Miller, 2019] Tim Miller. Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelligence*, 267:1–38, 2019.

[Oh *et al.*, 2004] Yoonna Oh, Maher N. Mneimneh, Zaher S. Andraus, Karem A. Sakallah, and Igor L. Markov. AMUSE: a minimally-unsatisfiable subformula extractor. In *Proceedings of DAC*, pages 518–523, 2004.

[Reiter, 1987] Raymond Reiter. A theory of diagnosis from first principles. *AIJ*, 32(1):57–95, 1987.

[Rossi *et al.*, 2006] Francesca Rossi, Peter van Beek, and Toby Walsh, editors. *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*. Elsevier, 2006.

[Saikko *et al.*, 2016] Paul Saikko, Johannes Peter Wallner, and Matti Järvisalo. Implicit hitting set algorithms for reasoning beyond NP. In *Proceedings of KR*, pages 104–113, 2016.

[Sqalli and Freuder, 1996] Mohammed H Sqalli and Eugene C Freuder. Inference-based constraint satisfaction supports explanation. In *AAAI/IAAI, Vol. 1*, pages 318–325, 1996.