Efficiently Explaining CSPs with Unsatisfiable Subset Optimization

Anonymous

Abstract

Recently, a novel method for explaining solutions of constraint satisfaction problems was proposed. An explanation here is a *sequence* of simple inference steps, where the simplicity of an inference step depends on the number and types of constraints used, eventually explaining all logical consequences of the problem. The current paper builds on these formal foundations and tackles 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. We show that this approach can be used to effectively find sequences of *optimal* explanations for constrained 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], Bogaerts *et al.* [2020] recently introduced a method that takes as input a satisfiable constraint program and explains the solution-finding process in a human-understandable way. Explanations in that work are sequences of simple inference steps, involving as few constraints and facts as possible. In order to generate these sequences, they 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 Bogaerts *et al.* [2020] 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, this high number of MUS calls causes severe efficiency problems, with the explanation generation process taking several hours.

Motivated by these observations, we develop algorithms for explaining CSPs that 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 taht is also used for generating cardinality-minimal MUSs [Ignatiev *et al.*, 2015].
- We observe that many of the 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 subset (OCUS)* problem and we show how $O(n^2)$ calls to MUS 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 OCUS 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.

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. Section 6 experimentally validates the approach, with various branches of related work discussed in Section 7 and conclusions Section 8.

2 Background

We will 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 a set $\mathcal{S} \subseteq \mathcal{F}$ that is unsatisfiable such that 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 subset $\mathcal{S} \subseteq \mathcal{F}$ 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 $\mathcal{F}$.*

It is well-known that 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.

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

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

```
X_best ← nil;
for l ∈ {I_end \ I} do
    X ← MUS(C ∧ I ∧ ¬l);
    if f(X) < f(X_best) then
        X_best ← X;
    end
end
return X_best
```

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 set 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.

To obtain a sequence of such steps, Bogaerts *et al.* [2020] iteratively search for the best (least costly) explanation step and add i ts consequence to the partial interpretation I. 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 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 subsequenty selects the best of the found such 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 $\mathcal{C}$ and I.

Experiments have shown that such a MUS-based approach can easily take hours, 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 algorithms presented in the next section. **(i)** 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). Most likely, the reason for choosing MUSes is that currently, *there are no algorithms for unsatisfiable subset* ***optimization***. **(ii)** Second, this algorithm uses MUS calls for every literal to explain separately. The goal of all of 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 on the number of literals from $\overline{(I_{end} \setminus I)}$ to include. **(iii)** Finally, the algorithm that computes an entire explanation sequence makes use of repeated calls to EXPLAIN-ONE-STEP and hence will solve many very similar problems. This raises the issue of ***incrementality****: can we re-use the computed data structures to achieve speed-ups in later calls?*?

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($\mathcal{C} \wedge I \wedge \overline{I_{end} \setminus I}, f, p$)

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, $f : 2^{\mathcal{F}} \rightarrow \mathbb{N}$ a cost function and p a predicate $p : 2^{\mathcal{F}} \rightarrow \{\boldsymbol{t}, \boldsymbol{f}\}$. 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 $\mathcal{S}' \subseteq \mathcal{F}$ with $p(\mathcal{S}') = \boldsymbol{t}$ satisfy $f(\mathcal{S}') \geq f(\mathcal{S})$.*

Rephrased in these terms, the task of the procedure EXPLAIN-ONE-STEP is to compute an OCUS of the formula $\mathcal{F} := \mathcal{C} \wedge I \wedge \overline{I_{end} \setminus I}$ with p the predicate that holds for subsets that contain exactly one literal of $\overline{I_{end}}$, 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 many constraints when previously derived facts can be used instead.

Our generic algorithm for computing OCUSs is depicted in Algorithm 3. It combines the hitting set-based approach for MUSes 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 6 for a moment, the algorithm alternates calls to a hitting set solver with calls to a SAT oracle on a

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

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

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 and is added to $\mathcal{H}$. In this way, the hitting set size always grow and a hitting set $\mathcal{S}$ will always be a subset of $\mathcal{F}$.

Instead of directly adding the complement of $\mathcal{S}$ to $\mathcal{H}$ as done by Ignatiev *et al.* [2015], our algorithm includes a call to GROW, which extends $\mathcal{S}$ into a larger subset of $\mathcal{F}$ that is still satisfiable (if possible). We discuss the different possible implementations of GROW later and evaluate their performance in Section 6.

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 $MCSs(\mathcal{F})$. 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.

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 some other unsatisfiable subset $\mathcal{S}'$ that satisfies p would exists 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 $\mathcal{F}$. □

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.

5 Efficient OCUS Computation for Explanations

While our OCUS algorithm in 3 is generic and can also be used to find OUSs, its constrainedness property is key to decreasing the complexity of explanation sequence generation. We now discuss optimizations to the OCUS algorithm that our 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 only marginally changes 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 sets-to-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 $\mathbf{SSs}$ 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.,

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

Our actual implementation goes a bit further: we know that all calls to $\text{OCUS}(\mathcal{F}, f, p)$ 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, and we have this upper bound on the set of formulas to be used, we can initialise all relevant decision variables once, and reuse the same MIP solving instance. We can temporarily disable every literal not in $\mathcal{F}$ by settings its weight to infinity in the objective function.

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, but on the other hand we also want the procedure to be very efficient. Indeed, if the computation cost of the GROW were not important, we could use a MaxSAT call to obtain a subset-maximal subset of $\mathcal{F}$, however, by doing that we expect the GROW costs to be too high to make the overall OCUS algorithm faster.

However, 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: 1) (translations of) the problem constraints (this is $\mathcal{C}$) 2) literals representing the assignment found thus far (this is I) and 3) the negations of literals not-yet-derived (this is $\overline{I_{end} \setminus I}$).
- $\mathcal{C}$ and I together are satisfiable 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 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}. This is likely to lead to a fairly large satisfiable subset and hence a good correction set.

6 Experiments

We now experimentally validate the performance of the different versions of our algorithm. Our benchmarks were ran 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. All code was implemented in Python on top of PySAT.[1] The MIP solver used is Gurobi 9.0 and when a (Max)SAT solver is used it is RC2 as bundled with PySAT. In the MUS-based approach we used PySAT's deletion-based MUS extractor MUSX [Marques-Silva, 2010].

All of our experiments were ran on a direct translation to PySAT of the 10 puzzles of Bogaerts *et al.* [2020][2]. In all puzzles, we used a cost of 60 for puzzle-agnostic constraints; 100 for puzzle-specific constraints; and cost 1 for facts. The selected clauses in an explanation for such a puzzle consist of one or more constraints together with some previously derived facts. 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 **contrainedness** on the time required to compute an explanation sequence?
Q4 Does **re-use** of computed satisfiable subsets 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 they — in order to avoid the quality problems we will illustrate below — implemented an extra inner loop with *even more* calls to MUS for a selected set of subsets of $\mathcal{C}$ of increasing size (this is their Algorithm 3). While this

[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.

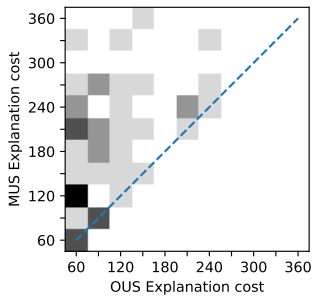

Figure 1: Q4 - Grow strategies until timeout or full explanation sequence generation

yields better explanations, it comes at the expense of computation time, thereby leading to several hours to generate the explanation of a single puzzle. We will see in our later experiments is that we can both *outperform*, in terms of computational cost, this simple MUS based implementation while also guaranteeing cost-optimality, as such achieving a Pareto improvement.

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, 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 large dark vertical column around cost 60), thereby confirming the need for a cost-based OUS/OCUS approach.

Domain-specific GROW The core computation complexity for generating explanations not only lies in the calls to the optimal hitting set solver, but more importantly in computing *high quality* satisfiable subsets. This requires a trade-off between efficiency of the grow strategy and quality of the produced satisfiable subset.

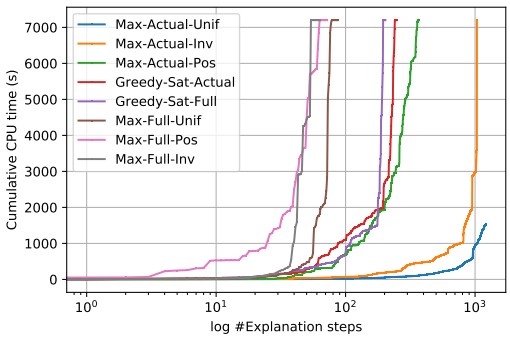

Figure 2: Q2 - Explanation specific GROW strategies

Thus, to answer **Q2**, we depict in Figure 2 the cumulative time of different grow strategies to generate the explanation sequence of all puzzles combined. The configurations are the following: *Full* refer to using the full unsatisfiable formula $\mathcal{F}$ given with every call of OCUS while *Actual* refers to using only the constraints that hold in the actual interpretation (Section 5); *Greedy* to growing using a sat-based refinement heuristic; the weighing scheme is either with uniform weights (*unif*), with the original weights (*pos*) or with the inverse of the original costs (*inv*) when using the MaxSAT solver (*Max*).

config	#HS	%HS	%SAT	%Grow	Time [s]
Greedy-SAT-Actual	8679	89.0	0.0	9.0	66782
Greedy-SAT-Full	8787	88.0	0.0	10.0	66297
Max-Actual-Unif	6831	81.0	0.0	19.0	5449
Max-Full-Unif	17280	85.0	0.0	15.0	72000

Table 1: Number of hitting sets and total time (in seconds) spent in core parts of OCUS algorithm across all puzzles.

As conjectured in Section 5 The MaxSAT-GROW on the actual interpretation indeed performs best. When repeating this experiment on other variants of the implementation (e.g., disabling constrainedness) the pattern was observed. The greedy strategy provides some improvement on the cumulative runtime, with only marginal gains when the actual interpretation is used.

We also observed (see Table 1) that in each of our configurations, the majority of the time is spent on computing hitting sets. As such, it makes sense to optimize the GROW procedure to provide results that are as good as possible and limit the choices of the hitting set solver as much as possible. Slightly surprising, performing a MaxSAT-GROW with the full interpretation takes *less* time than on the actual interpretation. The gains of using the actual interpretation seems to come most from the fact that it generates better sets to hit.

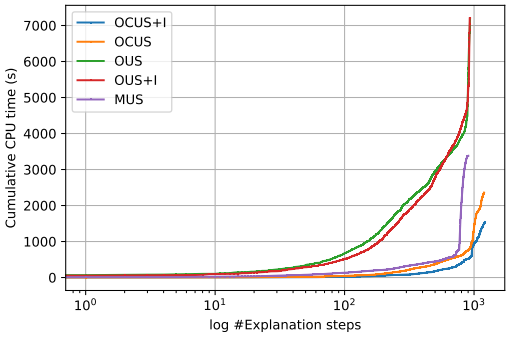

Figure 3: Q3 - Cumulative runtime evolution enhancements on incrementality and constraidness

Constrainedness and incrementality To answer **Q3** and **Q4**, we compare the effect of constrainedness in the search for explanations (C), and incrementality, i.e., reusing satisfiable subsets between OUS calls (+I). Figure 3 shows the results of the different variants, including the MUS-based variant discussed above. In this figure, the total number of explanation steps is not equal for all configurations, the reason for this is that the two OUS implementations timed-out before completing two of the puzzles (hence explaining their lower number) and the quality of the MUS-based explanations is worse, as observed in **Q1**. Worse explanations (that make more assumptions) often explain more literals at once and hence the total number of explanation steps is lower. All

O(C)US variants use Max-Actual-Unif as grow strategy, after we experimentally verified that this result of **Q2** also applied to the other OUS variants.

Constrainedness When comparing all five configurations, we see that the non-constrained optimal implementations cannot keep up with the plain MUS based implementation, as is to be expected since a harder problem is tackled. However, when adding constrainedness, by switching to Algorithm 2, we effectively reduce the number of calls to the OUS solver. The constrainedness enhancement as shown in Figure 3 proves the importance of our approach. The explanation generation times for a whole sequence are on close to and sometimes even outperform the MUS approach, while still solving the harder problem of unsatisifable subset optimization.

Incrementality To our surprise, we noticed that in the non-constrained OUS setting, incrementality had no noticable effect. When combining constrainedness with incrementality, on the other hand, we see that this incrementality yields important improvements. One potential explanation for this behaviour is the fact that, as discussed above, in the OCUS+I implementation, we only initialize a MIP and reulse it across the different EXPLAIN-ONE-STEP calls of algorithm 2. By keeping the MIP solver warm, we take advantage of its efficiency to handle the many satisfiable subsets which are represented under the form of constraints on the variables of the problem specification. Figure 3 demonstrates that contrary to to *OUS+I*, incrementality in **OCUS +I** significantly decreases the overall computation time.

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].

Our current work is motivated by a concrete algorithmic need that arose in this context. Specifically, the work of Bogaerts *et al.* [2020] shows the need for 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 work can be found in the literature on generating or enumerating MUSs [Lynce and Silva, 2004]. 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]. 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 focussed 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. Constraindness, incrementality and a domain-specific grow method were key to generating such sequences in a reasonable amount of time.

With the observed impact of different 'grow' methods, an open question remains wether 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. While we focussed on generating entire sequences, another question could be what the best method is for finding a single explanation step, that is a single OCUS. This would be important for example in interactive configuration applications [Van Hertum *et al.*, 2017]; where incrementality can also play across different queries. The synergies of our approach with the more general problem of QMaxSAT [Ignatiev *et al.*, 2016] is another open question.

The concept of OUS, incremental OUS and constrained OUS are not limited to explanations of satisfaction problems and we are keen to explore other applications too. A general direction here are *optimisation* problems and the role of the objective function in explanations. Within satisfaction, both MUSs and hitting set–based algorithms are also investigated in the context of explaining machine learning decisions [Ignatiev *et al.*, 2019]. In this context, an open direction for future work is to examine the potential benefit of using optimal MUS search and whether constrainedness properties can further boost its possibilities.

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.

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

[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 (DARPA)*, 2, 2017.

[Hamon *et al.*, 2020] Ronan Hamon, Henrik Junklewitz, and Ignacio Sanchez. Robustness and explainability of artificial intelligence. 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. volume 1, pages 432– 437 Vol. 1, 02 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*, pages 173–182. Springer, 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.

[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, 2010] Joao Marques-Silva. Minimal unsatisfiability: Models, algorithms and applications. In *2010 40th IEEE International Symposium on Multiple-Valued Logic*, pages 9–14. IEEE, 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.

[Van Hertum *et al.*, 2017] Pieter Van Hertum, Ingmar Dasseville, Gerda Janssens, and Marc Denecker. The kb paradigm and its application to interactive configuration. *Theory and Practice of Logic Programming*, 17(1):91–117, 2017.