Interactive hierarchical explanations in constraint solving

Wout Piessens
KU Leuven, Belgium

Ignace Bleukx
KU Leuven, Belgium

Tias Guns
KU Leuven, Belgium

Abstract

Many applications of constraint solving require explanations for unsatisfiability and guidance on how to restore satisfiability. A classical approach is to compute Minimal Unsatisfiable Subsets (MUSes), which pinpoint a set of constraints causing a conflict, and Minimal Correction Subsets (MCSes), which suggest a set of constraints that should be relaxed to restore feasibility. However, many MUSes and MCSes may exist for a given constraint problem, making it difficult for end-users to interpret and act upon them. Observing that MUSes often overlap substantially, we investigate computing explanations at higher levels of abstraction to explore the explanation space more effectively. Constraint models typically exhibit a natural structure, as modellers write constraints in groups (for example, row, column, and block constraints for Sudoku). We leverage this structure to derive explanations at multiple hierarchy levels, resulting in a framework for interactive hierarchical explanations. To support the efficient use of this framework, we extend the MARCO algorithm to support explanations at successive abstraction levels while reusing computations across levels through incrementality. Preliminary experiments demonstrate significant efficiency gains compared to directly using MARCO. The interactive setting enables users to progressively refine explanations and select suitable repair options, while maintaining control over the number of generated conflict sets and avoiding the combinatorial explosion associated with MUS and MCS enumeration.

2012 ACM Subject Classification Theory of computation → Constraint and logic programming; Human-centered computing → User centered design

Keywords and phrases Constraint solving, Explainable Artificial Intelligence

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

Acknowledgements This research is funded by the Research Foundation-Flanders (FWO-Vlaanderen, G020525N)

1 Introduction

Combinatorial problems are frequently solved using constraint solving. The user writes down their problem as a constraint model, which is then solved using a state-of-the-art solver.

Following wider trends in the field of *Explainable Artificial Intelligence* (XAI), attention has been devoted to explaining the output of constraint solvers to the user. In particular, we focus on explaining why a problem is unsatisfiable, that is, why it admits no solution. Traditionally, this is done by calculating Minimal Unsatisfiable Subsets (MUSes) and/or Minimal Correction Subsets (MCSes), which communicate to the user, respectively, a minimal set of constraints that are unsatisfiable together and a minimal set of constraints to remove to make the problem satisfiable. However, directly calculating MUSes and MCSes from the original problem constraints has its limitations. Firstly, every MUS only gives a localized, partial view of the underlying cause of inefficiency. Moreover, calculating all MUSes and MCSes to obtain a global overview is possible due to enumeration techniques that have

been developed in recent years [9], however, a worst-case exponential number of MUSes and MCSes may exist in terms of the number of constraints. Apart from inefficiency, this also reduces user interpretability due to the large amount of information. However, different MUSes tend to have many overlapping constraints; this raises the question of whether this information can be aggregated.

Our goal is to maintain a global overview of the cause of conflict while also enabling user interaction. In this way, we contribute to the relatively limited body of research on user-centered explanations in constraint solving. Previous work on this topic [18] has addressed the need for user support to be meaningful, useful, practical and flexible. To address these challenges, we propose a setting in which information about the cause of a conflict is revealed incrementally, upon user request. This approach avoids overwhelming the user while still enabling interpretable explanations and effective repairs. We formalize the approach as *interactive hierarchical explanations* in constraint solving.

2 Preliminaries

A CSP is a tuple $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ [17]. Here $\mathcal{X}$ is a set of variables, with domains defined by $\mathcal{D}$, and $\mathcal{C}$ is a set of constraints. A constraint $c \in \mathcal{C}$ is a tuple (R_j, S_j) where R_j is a relation on the variables in $S_j = \mathit{scope}(c) \subseteq \mathcal{X}$. This means that for all variables occurring in c, R_j is a subset of the Cartesian product of the domains of those variables. An assignment that satisfies all constraints in a CSP is called a solution of the problem. When no solution of a CSP exists, we say the CSP is *unsatisfiable*, and we seek to obtain explanations.

▶ **Definition 1** (Minimal Unsatisfiable Subset (MUS) [12]). *A Minimal Unsatisfiable Subset (MUS) is an unsatisfiable subset $U \subseteq \mathcal{C}$ for which it holds that each strict subset $U' \subsetneq U$ is satisfiable.*

An MUS is often used as an explanation of *why* a CSP is unsatisfiable, as it pinpoints a cause of conflict in the constraints [6, 3].

▶ **Definition 2** (Maximal Satisfiable Subset (MSS) [2]). *A Maximal Satisfiable Subset (MSS) is a satisfiable subset $N \subseteq \mathcal{C}$ for which it holds that any strict superset $N' \supsetneq N$ is unsatisfiable.*

An MSS can be used as an explanation to show which subset of the CSP can be satisfied. The complement of an MSS is a Minimal Correction Subset.

▶ **Definition 3** (Minimal Correction Subset (MCS) [11]). *A Minimal Correction Subset (MCS) is a subset $M \subseteq \mathcal{C}$ for which $\mathcal{C} \setminus M$ is satisfiable, and for any $M' \subsetneq M$, $\mathcal{C} \setminus M'$ is unsatisfiable.*

For an end-user, an MCS is a subset of constraints that, when removed, leaves the resulting CSP satisfiable. We will refer to MUSes and MCSes as *conflict sets*.

A partition Π of a set X is a collection of nonempty, pairwise disjoint subsets of X whose union is X. Given two partitions Π and Π' of X, we say that Π' is a refinement of Π iff for every $P' \in \Pi'$ there exists $P \in \Pi$ such that $P' \subseteq P$. We denote this by $\Pi' \preceq \Pi$, and by $\Pi' \prec \Pi$ if $\Pi' \neq \Pi$.

3 Interactive hierarchical explanations

Jussien and Ouis [7] observe that constraint models can typically be grouped by means of a hierarchy representing the problem, where constraints with similar meanings are grouped together at a higher level of abstraction. This hierarchy can be derived automatically from the

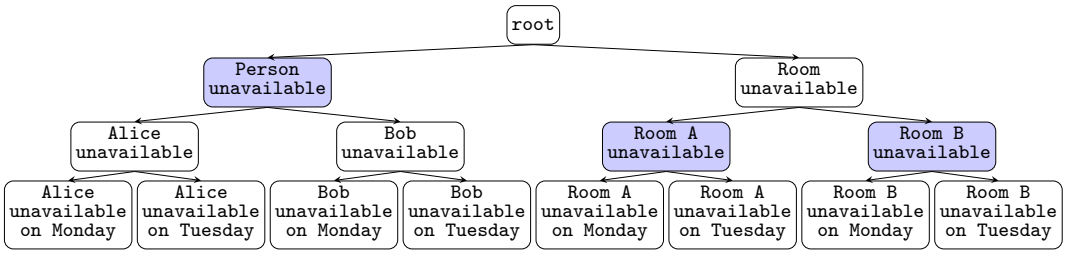

Figure 1 Hierarchy of constraints for a simple planning problem, with the final abstraction set Π' in Example 6 marked in blue.

model specification, e.g., using *constraint paths* in MiniZinc [15, 8], or it can be constructed manually by the user.

In what follows, we assume that the hierarchy of constraints has been derived in one of the ways outlined above. Using a hierarchy of constraints in an explanation setting has been explored in earlier work [7, 8]; however, this paper presents the first formalization of this idea.

Consider a CSP $(\mathcal{X}, \mathcal{D}, \mathcal{C})$. We refer to every constraint $c \in \mathcal{C}$ as a *primitive* constraint. Note that primitive constraints can still be complex, nested expressions written by the user, and do not necessarily map to a *solver-constraint* such as a single global constraint or linear inequality. A *constraint group* is a subset $G \subseteq \mathcal{C}$, representing the conjunction of all primitive constraints $c \in G$. Constraint groups can correspond to useful abstractions of their primitive constraints, hence why they are also referred to as *abstract* constraints in the literature [8].

▶ **Example 4** (Constraint group)**.** In a model representing the rules of Sudoku where $\textsc{AllDifferent}(row_i)$ for $i \in \{1, \ldots, 9\}$ are *primitive constraints*, they can be grouped together into a *constraint group* representing all row constraints : $\forall i \in 1..9 : \textsc{AllDifferent}(row_i)$.

A partition Π of $\mathcal{C}$ is equivalent to a set of constraint groups that together cover all primitive constraints in $\mathcal{C}$. Given that Π reflects the level of abstraction used to view the problem, we will refer to it as an *abstraction set* in what follows.

Now the concept of a *group-oriented* MUS, MSS and MCS generalizes from Definition 1, 2 and 3. This concept exists in the literature for SAT solving [1, 10, 14], where it is used on high-level constraints that are treated as indivisible groups of clauses; we generalize it here.

▶ **Definition 5.** *Given an unsatisfiable set of constraints $\mathcal{C}$, explicitly partitioned into constraint groups $\Pi = \{G_1, \cdots, G_k\}$, a* group-oriented minimal unsatisfiable subset *of Π is a set of groups $\mathcal{G} \subseteq \{G_1, \ldots, G_k\}$ such that:*

- *$\bigcup_{G \in \mathcal{G}} G$ is unsatisfiable, and*
- *for every $G_i \in \mathcal{G}$, the formula $\bigcup_{G \in \mathcal{G} \setminus \{G_i\}} G$ is satisfiable.*

Group-oriented MCSes and MSSes can be defined equivalently.

Using these concepts, we can fully explain the setting of interactive hierarchical explanations in Example 6. Given a constraint hierarchy as in Figure 1, the primitive constraints $\mathcal{C}$ are equal to the leaf nodes in the tree, and any node in the tree represents a constraint group G. From the hierarchy we therefore obtain a set of constraint groups $\mathcal{G}$, as well as a minimal refinement Π_G for every $G \in \mathcal{G}$ with $|G| > 1$, induced by the children in the tree, so that $\Pi_G \prec \{G\}$.

▶ **Example 6** (Interactive hierarchical explanations)**.** Consider a simple planning problem where *events* have to be planned during certain days: for every event, there is a list of *people*

that have to attend, and a list of *rooms* that can be used for the event. Every person and room has a list of available days. Now suppose that a certain event could not be scheduled, in other words, the problem is unsatisfiable. Assume that all foreground constraints of the problem are grouped in a constraint hierarchy as in Figure 1, together with a natural language description for each group.

Initially, we call a MUS and MCS enumeration algorithm on the abstraction set Π = {`Person unavailable`, `Room unavailable`}. Suppose that the enumeration algorithm returns one group-oriented MUS ({`Person unavailable`, `Room unavailable`}), and two group-oriented MCSes ({`Person unavailable`}, {`Room unavailable`}). In human language, this means that people and rooms together are the cause of conflict, and that the conflict can be fixed either by making people available or by making rooms available.

This explanation gives an initial indication of the cause of the conflict, but it is typically not sufficiently detailed for an end-user. The user can then, for instance, examine G = `Room unavailable` and zoom in on its specifics by applying its minimal refinement Π_G = {`Room A unavailable`, `Room B unavailable`}. Now the full abstraction set is Π' ={`Person unavailable`, `Room A unavailable`, `Room B unavailable`}, and it holds that $\Pi' \prec \Pi$.

Suppose that the new group-oriented MUS and MCSes are ({`Person unavailable`, `Room A unavailable`}), and ({`Person unavailable`}, {`Room A unavailable`}), respectively. In human terms, this indicates that the rooms mentioned in the previous explanation correspond only to Room A. Therefore, not all *active* constraint groups (groups in the current abstraction set) must be *relevant* constraint groups (groups in at least one group-oriented MUS or MCS); Room B plays no role in the conflict on this abstraction level and is therefore not relevant.

As discussed in previous work [8], it is preferable to only refine relevant constraint groups. However, we extend the setting described in this paper by allowing the user to select and refine any subset of relevant constraint groups.

4 Incremental enumeration algorithm

4.1 Background and motivation

Having introduced the setting of interactive hierarchical explanations, it is clear that we must be able to enumerate conflict sets for different abstraction sets efficiently. This motivates the development of an incremental algorithm for enumerating MUSes and MCSes.

Our algorithm builds on MARCO [9]: given a set of constraints C, MARCO can enumerate all MUSes and MSSes / MCSes by exploring the power set $\mathcal{P}(C)$. This is done by the *map solver* which keeps track of already explored parts of $\mathcal{P}(C)$, and solves for an unexplored subset of C.

Every constraint $c \in C$ is assigned a Boolean indicator variable b_c. Any complete assignment to these indicator variables indicates a subset $X \subseteq C$, where $b_c = \mathit{true} \Leftrightarrow c \in X$. Starting from a subset X found by the map solver, if the *satisfiability solver* finds that X is satisfiable, it is grown to an MSS, whereas if X is found to be unsatisfiable, it is shrunk to a MUS.

After a MUS U has been found, all supersets of U must necessarily also be unsatisfiable. Therefore, we add a *block-up* constraint $\bigvee_{c \in U} \neg b_c$ to the map solver expressing that the next subset of C cannot be a superset of U. After a MSS S has been found, we know that all subsets of S must necessarily also be satisfiable. Therefore, we add a *block-down* constraint $\bigvee_{c \in C \setminus S} b_c$ expressing that at least one constraint not in S must be in the next subset of C found by the map solver. Afterwards the map solver solves for a new subset $X \subseteq C$ respecting

the blocking constraints, and this is repeated until the map solver returns unsatisfiable; at that point the search space has been fully explored.

However, in the setting introduced in Example 6 the set of constraints is not static: the abstraction set Π containing constraint groups is subject to refinement. It would be desirable to reuse previous calculations as much as possible; more specifically, no new constraints should be posted to the satisfiability solver when Π is refined, and the map solver should still be able to block already explored parts of the search space when Π is refined.

Let $\mathcal{G}$ be the set of constraint groups derived from the constraint hierarchy. To make the satisfiability solver incremental, it suffices for every $G \in \mathcal{G}$ to be assigned an indicator variable b_G, and to add $b_G \Rightarrow \bigwedge_{c \in G} c$ to the satisfiability solver.

For the map solver, the important insight is that the blocking constraints are not only valid for MUSes and MSSes, but for unsatisfiable and satisfiable subsets in general; it must also hold that their supersets and subsets are unsatisfiable and satisfiable respectively.

Note that if U is a group-oriented MUS of Π, then for any $\Pi' \prec \Pi$, the set $\Pi'|_U = \{G' \in \Pi' \mid \exists G \in U : G' \subseteq G\}$ is a (group-oriented) unsatisfiable subset, since it partitions the same primitive constraints as U does. $\Pi'|_U$ is not necessarily minimal since the refinement introduces additional granularity, but the block-up constraint containing b_G for all $G \in \Pi'|_U$ remains valid nevertheless. An equivalent argument holds for group-oriented MCSes and block-down constraints. As a consequence, existing blocking constraints can be rewritten when the abstraction set is refined, by replacing a group G by the groups that refine it.

4.2 Implementation

We present Algorithm 2, an algorithm that builds on MARCO [9] and dynamically rewrites the blocking constraints based on the abstraction set Π, with the initialization of the map solver as an auxiliary function in Algorithm 1.

First, we must clarify the meaning of two new assumption variables. Given the abstraction set Π, $a_G, G \in \mathcal{G}$ is set to *true* iff $G \in \Pi$, in other words, if G is an *active* group. $p_G, G \in \{G' \in \mathcal{G} || G'| > 1\}$ is set to *true* iff there exists a $\Pi_G \prec \{G\}$ so that $\Pi_G \subseteq \Pi$. In other words, G is a *partitioned* group, intuitively this means that the current abstraction set refines G.

For every $G \in \mathcal{G}$, we generalize block-up and block-down constraints to work with separate variables u_G and d_G respectively, instead of directly using the indicator variables (line 15 and line 20). This way the map solver will never return unsatisfiable; there is always a valid assignment (assign $d_G = \mathit{true}$ and $u_G = \mathit{false}$ for all $G \in \mathcal{G}$). This is desirable, since we do not yet know the abstraction set Π, and therefore the search space $\mathcal{P}(\Pi)$ to consider. Now, we wish to substitute variables u_G and d_G with the appropriate indicator variables based on Π.

We remark that for $G \in \mathcal{G}$, the indicator variables b_G must be used in the blocking constraints iff G is active. Therefore, during initialization, we add constraints to the map solver requiring u_G and d_G to be substituted by b_G if G is active (see line 4). By performing this substitution, the map solver will return unsatisfiable iff all subsets of $\mathcal{P}(\Pi)$ have been explored; this follows from the MARCO algorithm and is proven in the foundational work introducing MARCO [9].

Next, we restate that for all partitioned groups G, conflict sets can be rewritten by substituting G by the groups that refine it. In line 9 variables u_G and d_G are substituted in such a way that existing blocking constraints are dynamically rewritten to refer to the indicator variables belonging to groups in the current abstraction set. The minimal refinement of every partitioned constraint group is iteratively applied until the currently active constraint groups are reached.

Algorithm 1 Initialize map solver

1: **procedure** INITIALIZE($\mathcal{G}', map$)
Require: Set of constraint groups $\mathcal{G}'$, map solver map
2: **for all** $G \in \mathcal{G}'$ **do**
3: $a_G, b_G, d_G, u_G \leftarrow$ BOOLVAR()
4: $map \leftarrow map \wedge (a_G \Rightarrow ((d_G \Leftrightarrow b_G) \wedge (u_G \Leftrightarrow b_G)))$
5: **if** $|G| > 1$ **then**
6: $\Pi_G \leftarrow$ MINIMALREFINEMENT(G)
7: $map \leftarrow$ INITIALIZE(Π_G, map)
8: $p_G \leftarrow$ BOOLVAR()
9: $map \leftarrow map \wedge (p_G \Rightarrow ((d_G \Leftrightarrow \bigvee_{G' \in \Pi_G} d_{G'}) \wedge (\neg u_G \Leftrightarrow \bigvee_{G' \in \Pi_G} \neg u_{G'})))$
10: **end if**
11: **end for**
12: **return** map
13: **end procedure**

This extension to MARCO is fully incremental, making use of assumption variables; the indicator variables b_G for the satisfiability solver, and the variables a_G and p_G for the map solver. GETASSUMPVARS in line 5 returns the assumption variables a_G, p_G that are true given Π. Then calls to the map solver (line 8) and satisfiability solver (line 10) are made with the appropriate assumption variables. For GROW and SHRINK existing algorithms can be used. After MARCO has fully explored $\mathcal{P}(\Pi)$, **ChangeAbstractionLevel()** in line 23 asks the user to examine a subset of relevant constraint groups on a new abstraction level, leading to a new abstraction set Π.

5 Preliminary experiments

We present some preliminary experiments, showing the usefulness of Algorithm 2 when iteratively refining the abstraction set Π compared to directly using MARCO on every abstraction set separately.

We test this on an extended version of the planning problem described in Example 6: this use case motivated the development of interactive hierarchical explanations. The implementation of Algorithm 2 builds on the MARCO implementation in CPMpy [5], and is tested against this implementation as a baseline. Both the map solver and the satisfiability solver used are Exact [16] [4], due to its support for assumption variables and incremental solving. All experiments were executed on a machine equipped with an AMD Ryzen 7 PRO 250 CPU running at 3.30 GHz. The Python version used is Python 3.13.5.

We consider three instances of the planning problem, and for every instance, we consider an initial explanation step for a given abstraction set Π (S1), followed by 3 refinement steps (S2, S3, S4). For all four explanation steps, we benchmark two separate parts of Algorithm 2: *initialization* of the assumption models and the solvers (line 1 to line 3), and *enumeration* of conflict sets (from line 5 to line 22). Experimental results are shown in Table 1. We see that the improvements of incremental MARCO compared to the baseline are twofold: first, we do not need to reinitialize the satisfiability solver for every refinement step, since separate indicator variables for every $G \in \mathcal{G}$ exist. Therefore, despite the initialization time of incremental MARCO being slightly slower due to Algorithm 1, it only needs to happen once. Secondly, we also obtain time gains during conflict set *enumeration*; since the satisfiability

Algorithm 2 Incremental MARCO

Require: Set of primitive constraints $\mathcal{C}$ with initial abstraction set Π
Ensure: Group-oriented MUSes and MCSes of different abstraction sets Π of $\mathcal{C}$

```
map, sat ← SOLVER()
map ← INITIALIZE({C}, map)
continue ← true
while continue do
    assump ← GETASSUMPVARS(Π)
    // Π is a correction subset, since it partitions all primitive constraints
    map ← map ∧ ⋀_{G∈Π} d_G
    while map(assump) is satisfiable do
        seed ← GETSOLUTION(map) ∩ {b_G | G ∈ Π}
        if sat(seed) then
            GroupMSS ← GROW(seed)
            GroupMCS ← Π \ GroupMSS
            output GroupMCS
            // Block subsets of GroupMSS
            map ← map ∧ ⋀_{G∈GroupMCS} d_G
        else
            GroupMUS ← SHRINK(seed)
            output GroupMUS
            // Block supersets of GroupMUS
            map ← map ∧ ⋀_{G∈GroupMUS} ¬u_G
        end if
    end while
    Π, continue ← ChangeAbstractionLevel()
end while
```

Table 1 Time (in seconds) of initialization and enumeration for 3 instances, and 4 steps per instance

Instance	Method	Initialization				Enumeration			
		S1	S2	S3	S4	S1	S2	S3	S4
Inst. 1	Baseline	**14.93**	13.19	12.99	12.77	1.38	45.62	88.56	110.99
	Incremental	15.62	**0.00**	**0.00**	**0.00**	**1.13**	**41.54**	**55.08**	**54.63**
Inst. 2	Baseline	**0.51**	0.45	0.44	0.44	0.10	0.38	11.68	36.33
	Incremental	0.59	**0.00**	**0.00**	**0.00**	**0.08**	**0.24**	**10.30**	**30.31**
Inst. 3	Baseline	**44.10**	44.42	44.31	43.77	**4.73**	10.16	18.97	50.50
	Incremental	46.36	**0.00**	**0.00**	**0.00**	4.98	**4.16**	**18.32**	**50.25**

solver works equivalently to the baseline during enumeration, we see that making the map solver incremental leads to additional time gains when refining Π in new explanation steps (especially in Instance 1), since already explored parts of the search space $\mathcal{P}(\Pi)$ remain blocked. However, this behavior is not observed for every refinement step on every instance, raising the question of when map solver incrementality provides the greatest benefit.

6 Conclusion

We present an interactive framework for explaining unsatisfiability in constraint solving, improving efficiency over a baseline approach through incrementality. This efficiency improvement is observed in the preliminary experiments, both with respect to solver initialization and enumeration of conflict sets. However, the effect of incrementality on explanation time requires further investigation.

As user interaction is central to this setting, an important next step is the development of intuitive visualizations for end users. We see promise in integrating this method with an intelligent user interface, since our approach respects the well-recognized Shneiderman's Visual Information Seeking mantra [19]: *overview first, zoom and filter, details on demand.*

Additionally, it is worthwhile to explore further techniques for aggregating all conflict information, in cases where a large number of conflict sets remains unavoidable. In such scenarios, partial conflict set enumeration and aggregation techniques such as power indices [13] offer promising directions.

References

1. M. Fareed Arif, Carlos Mencía, and João Marques-Silva. Efficient MUS enumeration of horn formulae with applications to axiom pinpointing. *CoRR*, abs/1505.04365, 2015. URL: http://arxiv.org/abs/1505.04365, arXiv:1505.04365.
2. Jaroslav Bendík and Kuldeep S. Meel. Counting maximal satisfiable subsets. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 3651–3660. AAAI Press, 2021. URL: https://doi.org/10.1609/aaai.v35i5.16481, doi:10.1609/AAAI.V35I5.16481.
3. Ignace Bleukx, Tias Guns, and Dimos Tsouros. Explainable constraint solving: A hands-on tutorial, 2024. doi:10.5281/zenodo.10694140.
4. Jan Elffers and Jakob Nordström. Divide and conquer: Towards faster pseudo-boolean solving. In Jérôme Lang, editor, *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pages 1291–1299. ijcai.org, 2018. URL: https://doi.org/10.24963/ijcai.2018/180, doi:10.24963/IJCAI.2018/180.
5. Tias Guns. Increasing modeling language convenience with a universal n-dimensional array, cppy as python-embedded example. In *Proceedings of the 18th workshop on Constraint Modelling and Reformulation at CP (Modref 2019)*, volume 19, 2019.
6. Sharmi Dev Gupta, Begum Genc, and Barry O'Sullivan. Explanation in constraint satisfaction: A survey. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, pages 4400–4407. ijcai.org, 2021. URL: https://doi.org/10.24963/ijcai.2021/601, doi:10.24963/IJCAI.2021/601.
7. Narendra Jussien and Samir Ouis. User-friendly explanations for constraint programming. In Anthony J. Kusalik, editor, *Proceedings of the Eleventh Workshop on Logic Programming Environments (WLPE'01), Paphos, Cyprus, December 1, 2001*, 2001. URL: https://arxiv.org/abs/cs/0111037.
8. Kevin Leo and Guido Tack. Debugging unsatisfiable constraint models. In Domenico Salvagnin and Michele Lombardi, editors, *Integration of AI and OR Techniques in Constraint Programming - 14th International Conference, CPAIOR 2017, Padua, Italy, June 5-8, 2017, Proceedings*, Lecture Notes in Computer Science, pages 77–93. Springer, 2017. doi:10.1007/978-3-319-59776-8_7.

9 Mark H. Liffiton, Alessandro Previti, Ammar Malik, and João Marques-Silva. Fast, flexible MUS enumeration. *Constraints An Int. J.*, 21(2):223–250, 2016. URL: https://doi.org/10.1007/s10601-015-9183-0, doi:10.1007/S10601-015-9183-0.

10 Mark H. Liffiton and Karem A. Sakallah. Algorithms for computing minimal unsatisfiable subsets of constraints. *J. Autom. Reason.*, 40(1):1–33, 2008. URL: https://doi.org/10.1007/s10817-007-9084-z, doi:10.1007/S10817-007-9084-Z.

11 João Marques-Silva, Federico Heras, Mikolás Janota, Alessandro Previti, and Anton Belov. On computing minimal correction subsets. In Francesca Rossi, editor, *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, pages 615–622. IJCAI/AAAI, 2013. URL: http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6922.

12 João Marques-Silva and Inês Lynce. On improving MUS extraction algorithms. In Karem A. Sakallah and Laurent Simon, editors, *Theory and Applications of Satisfiability Testing - SAT 2011 - 14th International Conference, SAT 2011, Ann Arbor, MI, USA, June 19-22, 2011. Proceedings*, Lecture Notes in Computer Science, pages 159–173. Springer, 2011. doi:10.1007/978-3-642-21581-0_14.

13 Pablo Martínez-Naredo, Raúl Mencía, João Marques-Silva, and Carlos Mencía. Explanations of unsatisfiability beyond minimal subsets. In Giovanni Casini, Besik Dundua, and Temur Kutsia, editors, *Logics in Artificial Intelligence - 19th European Conference, JELIA 2025, Kutaisi, Georgia, September 1-4, 2025, Proceedings, Part II*, Lecture Notes in Computer Science, pages 141–158. Springer, 2025. doi:10.1007/978-3-032-04590-4_10.

14 Alexander Nadel. Boosting minimal unsatisfiable core extraction. In Roderick Bloem and Natasha Sharygina, editors, *Proceedings of 10th International Conference on Formal Methods in Computer-Aided Design, FMCAD 2010, Lugano, Switzerland, October 20-23*, pages 221–229. IEEE, 2010. URL: https://ieeexplore.ieee.org/document/5770953/.

15 Nicholas Nethercote, Peter Stuckey, Ralph Becket, Sebastian Brand, Gregory Duck, and Guido Tack. Minizinc: Towards a standard cp modelling language. pages 529–543, 09 2007. doi:10.1007/978-3-540-74970-7_38.

16 Nonfiction Software. Exact, 2026. Accessed: 2026-05-04. URL: https://gitlab.com/nonfiction-software/exact.

17 Francesca Rossi, Peter van Beek, and Toby Walsh, editors. *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*. Elsevier, 2006. URL: https://www.sciencedirect.com/science/bookseries/15746526/2.

18 Ilankaikone Senthooran, Matthias Klapperstück, Gleb Belov, Tobias Czauderna, Kevin Leo, Mark Wallace, Michael Wybrow, and Maria Garcia de la Banda. Human-centred feasibility restoration in practice. *Constraints An Int. J.*, 28(2):203–243, 2023. URL: https://doi.org/10.1007/s10601-023-09344-5, doi:10.1007/S10601-023-09344-5.

19 Ben Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Proceedings of the 1996 IEEE Symposium on Visual Languages, Boulder, Colorado, USA, September 3-6, 1996*, pages 336–343. IEEE Computer Society, 1996. doi:10.1109/VL.1996.545307.