Efficient MUS Extraction with High-Level Model Rotation

Author: Please provide author information

Abstract

The extraction of *Minimal Unsatisfiable Subsets (MUSes)* has been extensively studied in SAT, but it is also useful in other formalisms such as pseudo-Boolean solving (PB) and constraint programming (CP), especially for explaining unsatisfiability. In this paper we focus on highly efficient deletion-based MUS extraction, where the main bottleneck is the number of solver calls needed to find a MUS. This number can be greatly reduced by techniques such as model rotation (MR), which manipulates a satisfying assignment to find new critical constraints. However, MR has mostly been studied only for (grouped) clausal constraints. We propose generalisations of MR for pseudo-Boolean and CP constraints, and empirically evaluate them on PB and CP instances, including SAT encodings. Our results show that MR is most effective when applied directly at the level of the original formalism, increasing the number of solved instances for PB and reducing solving times for CP.

2012 ACM Subject Classification Theory of computation → Constraint and logic programming

Keywords and phrases MUS extraction, explanations, constraint programming, pseudo-Boolean

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

Category Workshop Paper

1 Introduction

Minimal Unsatisfiable Subsets (MUSes) provide concise explanations of why a constraint model is infeasible, and have been used for debugging, scheduling, and step-wise explanation [14, 4, 6].

In essence, MUS extraction involves solving different subsets of a problem until a MUS is found. To mitigate the bottleneck of the solver calls, techniques have been developed to either make the solver calls faster, or to reduce the number of solver calls. In this work we focus on the highly effective technique of model rotation [16, 25].

While the concept of MUS extraction applies to any paradigm, earlier work has largely focused on clausal formulas. However, real-world problems are typically modelled first at a higher abstraction level, such as pseudo-Boolean (PB) or constraint programming (CP). The constraints can then be translated to clausal or conjunctive normal form (CNF), but the resulting unsatisfiable subsets may be less helpful to users because of the introduced clauses and auxiliary variables. This can be addressed by i) grouping clauses that represent a high-level constraint [22] or ii) direct extraction of the MUS on the high-level formula [26, 19, 11]. The former has been the state of the art (SOTA) because it benefits from the efficiency techniques mentioned in the previous paragraph. In this paper, we take the latter approach and develop algorithms for high-level model rotation for PB and CP constraints.

2 Background

A Constraint Satisfaction Problem (CSP) is defined as a triple $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ [20]. Here, $\mathcal{X}$ is a set of discrete variables with each an associated set of values (i.e. a domain) in $\mathcal{D}$. The domain of a variable x is written as $\mathcal{D}[x]$. A variable can be a finite-domain integer ($\mathcal{D}[x] \subset \mathbb{Z}$) or a Boolean ($\mathcal{D}[x] = \{true, false\}$). An assignment ρ maps each variable x to a value in its domain $\mathcal{D}[x]$. $\mathcal{C}$ is a set of constraints and we use $vars(\mathcal{C})$ to denote its variables, or $vars(C)$ to denote the variable of a constraint $C \in \mathcal{C}$. Each constraint maps an assignment ρ to $true$ or $false$, we denote the value an assignment ρ is mapped to by a constraint C as ρ^C. We write $\rho(x)$ for the value of a variable x in a

© Jane Open Access and Joan R. Public;
licensed under Creative Commons License CC-BY 4.0

solution ρ. The value of variable x can be set to a specific value v, which we write as $\rho\mid_{x=v}$. An assignment that is mapped to $true$ by all constraints in $\mathcal{C}$ is called a *solution*. A CSP where no ρ is a solution is *unsatisfiable* or *UNSAT* for short. The term pseudo-Boolean (PB) refers to a 01-linear inequality. We identify 1 with $true$ and 0 with $false$. A literal ℓ is a Boolean variable x or its negation $\neg x = 1 - x$. We assume PB constraints are in *normalised form* $\sum_i c_i\ell_i \geq \delta$, with pairwise distinct literals, positive coefficients, and literals sorted by decreasing coefficient. A PB constraint with degree 1 is a clause, and $lits(C)$ denotes the literals of C. CP solvers use constraint propagators, while PB and SAT solve encodings into PB constraints and clauses respectively. A formula of clauses is a CNF; when clauses are partitioned into groups, it is a GCNF. A group G is satisfied iff all its clauses are satisfied.

▶ **Definition 1** (MUS). *We call $\mathcal{M} \subseteq \mathcal{C}$ a MUS if it is unsatisfiable and removing any constraint from $\mathcal{M}$ makes it satisfiable, i.e. $\mathcal{S} \subsetneq \mathcal{M}$ is satisfiable.*

▶ **Definition 2** (Critical Constraint). *Let $\mathcal{S}$ be any set of constraints and $C \in \mathcal{S}$. If $\mathcal{S}$ is unsatisfiable and $\mathcal{S}\backslash\{C\}$ is satisfiable, then C is a critical constraint of $\mathcal{S}$.*

2.1 Deletion-based MUS extraction

We focus on deletion-based MUS extraction [15], which works as follows. Note that several MUSes may exist for one CSP, but a critical constraint of a set $\mathcal{S}$ belongs to every MUS of $\mathcal{S}$ [15]. Starting from $\mathcal{C}' = \mathcal{C}$ and $\mathcal{M} = \emptyset$, the algorithm removes constraints one by one. If $\mathcal{M} \cup \mathcal{C}'$ becomes satisfiable, the removed constraint C is critical and is added to $\mathcal{M}$, otherwise it is not needed to explain the unsatisfiability and C is discarded. Because this requires up to one solver call per constraint, speeding these solver calls up or reducing the number of them is central to efficient MUS extraction. Examples of the former include incremental solving [1], redundant constraint addition [23] and constraint scheduling [22]. Examples of the latter are unsat core refinement [16], symmetry exploitation [5] or model rotation [16]. In this work we will focus on model rotation, review how it works for clausal formulas and in Section 3 we will show how to generalise it to PB and CP constraints.

2.2 Model Rotation for Clausal Constraints

Model rotation (MR) finds additional critical constraints after a SAT result in the MUS procedure. It starts from an *associated assignment* α that violates the critical constraint C but satisfies $\mathcal{C}'\backslash\{C\}$. MR modifies α into a new assignment α' so that C becomes satisfied. This is done iteratively by flipping the value of each literal $\ell \in lits(C)$. Since $\mathcal{C}'$ is unsatisfiable there must be at least one constraint in $\mathcal{C}'$ which is unsatisfied. If exactly one other constraint C' is violated, C' is also critical and no extra solver call is needed [16]. We use the eager recursive variant of MR [17], which worked well in preliminary experiments. We will refer to the clausal version of MR as SAT-MR. SAT-MR has been generalised for grouped clausal constraints (GSAT-MR), but GSAT-MR is known to be less effective than SAT-MR [18]. However, when dealing with finite-domain constraints it is crucial to use GSAT-MR to preserve the correctness of the MUS.

3 Model Rotation for PB and CP constraints

In this section we describe how we can generalise MR to PB and CP constraints. SAT-MR does not work for PB or CP constraints, because flipping the value of a variable is not defined for integer domains, and even for Boolean domains, a single flip may not satisfy a critical PB constraint. We develop two specialised algorithms: `PB-MR`, which works specifically for PB constraints, and `CP-MR`, which works for CP constraints.

Algorithm 1 PB-MR

Input: critical constraint C, associated assignment α, MUS set $\mathcal{M}$, unchecked constraints $\mathcal{C}'$, newly found critical constraints $\mathcal{R}$, cascading option $cas?$
Output: updated MUS set $\mathcal{R}$

```
for ℓ ∈ lits(C) do
    if α(ℓ) is false then
        α ← α|_{ℓ=1}                                   // flip value of ℓ
        if α^C then
            U ← {C' | C' ∈ C' ∪ M; ¬α^{C'}}
            if |U| = 1 then
                C' ← e ∈ U
                if C' ∉ R then
                    R ← R ∪ {C'}
                    R ← PB-MR(C', α, M, C', R, cas?)
            α ← α|_{ℓ=0}                               // flip the value of ℓ back
        else if ¬cas? then
            return R                                   // no need to flip lits with lower coeffs
return R
```

PB Model Rotation (PB-MR) Recall that PB constraints are in normalised form $\sum c_i \ell_i \geq \delta$, with literals sorted by decreasing coefficient. Unlike clauses, setting one literal *true* does not necessarily satisfy a PB constraint. So in PB-MR we skip literals already set to *true*. If flipping a *false* literal still does not satisfy C, then no single flip on a literal with a smaller coefficient can satisfy the constraint either. We can therefore either stop immediately, or keep the last flip and continue accumulating flips until C becomes satisfied. We call the latter variant *cascading MR (PB-MRcas)*. The rest of the procedure is the same as SAT-MR. This lets us detect critical constraints that standard GSAT-MR can miss, as the next example illustrates. We show the pseudo-code for PB-MR in Algorithm 1.

▶ **Example 3.** Consider the PB problem with constraints $C_1 : x+y+z \geq 2$ and $C_2 : \neg x+\neg y \geq 2$. This problem is UNSAT and its MUS is simply $\{C_1, C_2\}$. A possible GCNF representation is $G_1 : \{(x \vee \neg a), (x \vee \neg b), (y \vee a), (y \vee \neg b), (z \vee \neg a), (z \vee b)\}$ and $G_2 : \{(\neg x), (\neg y)\}$, where a and b are auxiliary variables. Let G_1 be the first group removed during MUS extraction, with associated assignment $\alpha = \{x = 0, y = 0, z = 1, a = 1, b = 1\}$. Since G_1 has multiple falsified clauses and no literal appears in all of them, no single flip of x or y satisfies the whole group. With PB-MRcas, flipping x or y to 1 satisfies C_1 and falsifies C_2, so C_2 is identified as critical without another solver call. Although one could in principle combine multiple flips in the GCNF formula, e.g. $\{x = 1, y = 0, z = 1, a = 1, b = 0\}$, no systematic algorithm for this is known to us.

CP Model Rotation (CP-MR) CP-MR requires a few adaptations compared to SAT-MR. Since integer variables cannot simply be flipped, we instead traverse the domain values of each variable $\mathcal{D}[x]$. As in PB-MR, each change must be checked explicitly because it may still leave C unsatisfied. To limit the overhead for large domains, we consider only values within a radius r of the current value. Once a change yields a new critical constraint C', we stop exploring further values for that variable, since additional changes are likely to rediscover the same constraint C. Before moving to the next variable, we restore x to its value in α. The next example shows that CP-MR can find critical constraints that GSAT-MR misses on an encoding of the same problem.

▶ **Example 4.** Consider the CSP with variables $\mathcal{X} = \{x, y\}$, domains $\mathcal{D}[x] = \mathcal{D}[y] = \{1, 2\}$, and constraints $\mathcal{C} = \{C_1 : ((x = 1) \leftrightarrow (y = 1)), C_2 : ((x = 2) \leftrightarrow (y = 1))\}$. This CSP is UNSAT and both constraints belong to the MUS. Now encode it using the direct encoding [24], introducing a Boolean variable x_i for each value $i \in \mathcal{D}[x]$ indicating whether $x = i$, and similarly for y. This also requires `exactly-one` constraints to ensure that each variable takes exactly one value. A possible GCNF representation is $\mathcal{C}_{GCNF} = \{G_0 : \{x_1 \vee x_2, \neg x_1 \vee \neg x_2, y_1 \vee y_2, \neg y_1 \vee \neg y_2\}, G_1 : \{x_1 \vee \neg y_1, \neg x_1 \vee y_1\}, G_2 : \{x_2 \vee \neg y_1, \neg x_2 \vee y_1\}\}$.

Let G_1 be the first group removed from $\mathcal{C}'$ during MUS extraction. Then the remaining formula $\mathcal{C}'$ is satisfiable, for example with associated assignment $\alpha = \{x_1 = 0, x_2 = 1, y_1 = 1, y_2 = 0\}$. Notice that GSAT-MR can only flip the variables in the critical group G_1, i.e. x_1 or y_1. However, flipping either of those variables will violate the `exactly-one` constraints in G_0 (without also flipping x_2 or y_2 respectively).

At the CP level, this is not an issue: an associated assignment for C_1 is $\alpha = \{x = 2, y = 1\}$. CP-MR can then change x to 1, which satisfies C_1 and falsifies C_2. Thus it identifies C_2 as critical without an additional solver call.

4 Experiments

We implemented our MR variants in *CPMpy* [10], a solver-independent modelling library. We evaluate the effectiveness of the techniques on UNSAT PB instances and UNSAT CP instances.

From the PB'25 competition [21], we selected the 264 UNSAT instances solved within 10 minutes as our PB benchmark. For the CP benchmark, we collected 200 instances from all editions of past XCSP competitions. These instances include global constraints such as `AllDifferent`, `Count`, `InDomain`, `Cumulative`, `Table`, `NoOverlap`, and `Element`. Among them, 31 instances contain only Boolean variables, 126 have small integer domains (maximum size 3–100), and 43 have large integer domains (maximum size above 100).

Our implementation uses three back-end solvers: CaDiCaL 1.9.5 through PySAT 1.8.25 [12, 13, 3], Exact 2.2.1 [9, 8], and OR-Tools CP-SAT 9.14.6206 [7] as SAT, PB, and CP representatives respectively. Since CPMpy automatically encodes instances to the formalism of the back-end solver, each solver can be run on both benchmarks.

We also compare our implementations to the state of the art MUS extractor MUSer, specifically version 2.0 [2], with CaDiCaL as a back-end solver. To use MUSer on the PB and CP instances, we convert these instances to GCNF using CPMpy.

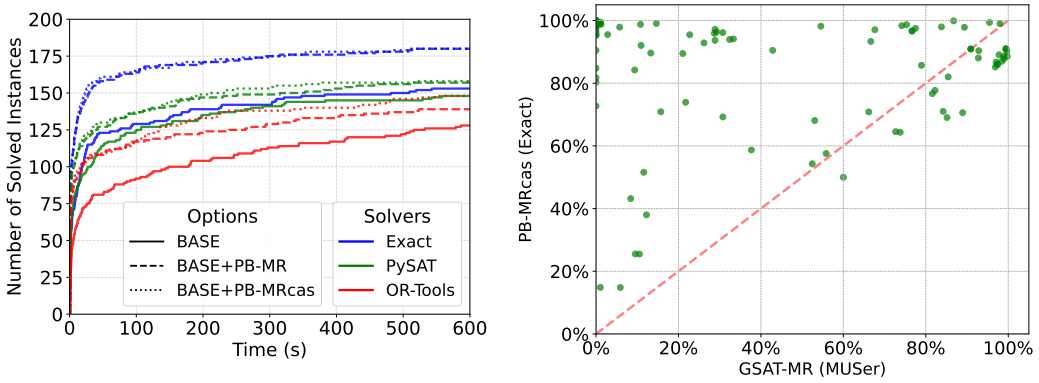
Our experiments were run on a pool of identical 32 Intel(R) Xeon(R) Silver 4514Y (2GHz) CPUs with 256GB of shared memory. We give all solvers 31GB of memory and a time limit of 600 seconds.

For both benchmarks our research question is: How effective are the generalised MR algorithms?

4.1 Effectiveness of model rotation

The results are summarised in Figure 1 and Table 1. In Figure 1a, we see that both PB-MR variants improve over the baseline without MR for all solvers, and PB-MRcas is consistently slightly better than PB-MR. For CP-MR, Table 1 shows the effect of the radius from Section 3 on solved instances, PAR2, and the percentage of critical constraints found by MR. In general, CP-MR speeds up MUS extraction for all radii, although only Exact consistently solves more instances. Larger radii find more critical constraints, but the best trade-off in solved instances and PAR2 appears at radius 10. The benchmark contains only a small number of instances with large domains, so using radius ∞ causes little overhead overall, although it is costly on a few instances with the largest domains.

We also compare the percentage of critical constraints found by our generalised MR methods with MUSer using GSAT-MR. The scatter plot is shown in Figure 1b. For this comparison, we use



(a) ECDF for variants of PB **(b)** Scatterplot PB

Figure 1 (a) ECDF of the PB-MR variants on the PB benchmark. (b) Scatterplot comparing the percentage of critical constraints found by MR; top-left dots favour the generalised algorithms, bottom-left dots favour GSAT-MR. Only instances solved by both methods are shown.

Exact with PB-MRcas on the PB benchmark. The percentage is measured relative to the size of the extracted MUS, so higher values mean that more solver calls were avoided. Although MR depends on the satisfying assignment returned by the solver, the plot shows that PB-MRcas is substantially more effective than GSAT-MR at finding additional critical constraints. A similar effect can be observed for CP-MR, but that figure was omitted for space reasons.

5 Conclusion

In this paper we develop generalised model rotation (MR) algorithms for PB and CP constraints. We evaluate their effectiveness and compare them to GSAT-MR on grouped clausal formulas. We find that the generalised algorithms find more critical constraints for both PB and CP constraints, leading to more efficient MUS extraction.

For future work, it would be interesting to extend the comparison to MUS extractors that work directly on high-level constraints such as ILP or SMT. Other techniques studied for clausal MUS extraction could also be applied to more general finite-domain constraints.

Table 1 Performance on the CP benchmark in terms of solved instances, PAR2 solve time in seconds, and the percentage of critical constraints found by CP-MR. PAR2 counts timeouts as 2×600s. MR% is computed only on instances solved by all configurations. Higher solved counts and lower PAR2 are better. MR% is not directly correlated with performance.

Radius	PySAT (INC)			Exact (INC)			OR-Tools		
	Solved	PAR2	MR%	Solved	PAR2	MR%	Solved	PAR2	MR%
0	150	324.5	0	149	321.9	0	165	239.0	0
1	151	318.2	57.7	154	291.3	57.9	165	237.8	56.8
2	150	322.3	60.3	153	301.0	62.3	164	242.0	60.7
5	151	319.3	63.3	152	300.5	63.0	165	237.4	65.6
10	151	323.5	64.0	155	286.6	67.1	166	**229.6**	66.1
20	151	318.0	65.1	154	294.0	67.1	165	236.2	66.2
50	150	322.7	65.8	156	283.8	66.9	165	241.4	69.7
∞	150	325.0	69.8	156	286.7	**72.4**	**167**	232.8	70.9

References

1. Gilles Audemard, Jean-Marie Lagniez, and Laurent Simon. Improving Glucose for Incremental SAT Solving with Assumptions: Application to MUS Extraction. In Matti Järvisalo and Allen Van Gelder, editors, *Theory and Applications of Satisfiability Testing – SAT 2013*, pages 309–317, Berlin, Heidelberg, 2013. Springer. doi:10.1007/978-3-642-39071-5_23.
2. Anton Belov and Joao Marques-Silva. MUSer2: An Efficient MUS Extractor1: System description. *Journal on Satisfiability, Boolean Modeling and Computation*, 8(3-4):123–128, December 2012. doi:10.3233/SAT190094.
3. Armin Biere, Tobias Faller, Katalin Fazekas, Mathias Fleury, Nils Froleyks, and Florian Pollitt. Cadical 2.0. In *International Conference on Computer Aided Verification*, pages 133–152. Springer, 2024.
4. Ignace Bleukx, Ryma Boumazouza, Tias Guns, Nadine Laage, and Guillaume Poveda. Modeling and Explaining an Industrial Workforce Allocation and Scheduling Problem. In Maria Garcia de la Banda, editor, *31st International Conference on Principles and Practice of Constraint Programming (CP 2025)*, volume 340 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:24, Dagstuhl, Germany, 2025. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CP.2025.6.
5. Ignace Bleukx, Hélène Verhaeghe, Bart Bogaerts, and Tias Guns. Exploiting Symmetries in MUS Computation (Extended version), December 2024. arXiv:2412.13606, doi:10.48550/arXiv.2412.13606.
6. Bart Bogaerts, Emilio Gamba, and Tias Guns. A framework for step-wise explaining how to solve constraint satisfaction problems. *Artificial Intelligence*, 300:103550, 2021.
7. Thibaut Cuvelier, Frederic Didier, Vincent Furnon, Steven Gay, Sarah Mohajeri, and Laurent Perron. Or-tools' vehicle routing solver: A generic constraint-programming solver with heuristic search for routing problems. In *24e congrès annuel de la société française de recherche opérationnelle et d'aide à la décision*, 2023.
8. Devriendt, Jo. Exact. https://gitlab.com/nonfiction-software/exact, 2026.
9. Jan Elffers and Jakob Nordström. Divide and Conquer: Towards Faster Pseudo-Boolean Solving. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, pages 1291–1299, Stockholm, Sweden, July 2018. International Joint Conferences on Artificial Intelligence Organization. doi:10.24963/ijcai.2018/180.
10. 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.
11. Ofer Guthmann, Ofer Strichman, and Anna Trostanetski. Minimal unsatisfiable core extraction for smt. In *2016 Formal Methods in Computer-Aided Design (FMCAD)*, pages 57–64. IEEE, 2016.
12. Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. PySAT: A Python toolkit for prototyping with SAT oracles. In *SAT*, pages 428–437, 2018. doi:10.1007/978-3-319-94144-8_26.
13. Alexey Ignatiev, Zi Li Tan, and Christos Karamanos. Towards universally accessible SAT technology. In *SAT*, pages 4:1–4:11, 2024. URL: https://doi.org/10.4230/LIPIcs.SAT.2024.16, doi:10.4230/LIPICS.SAT.2024.16.
14. 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*, pages 77–93, Cham, 2017. Springer International Publishing. doi:10.1007/978-3-319-59776-8_7.
15. Joao Marques-Silva. Minimal Unsatisfiability: Models, Algorithms and Applications (Invited Paper). In *2010 40th IEEE International Symposium on Multiple-Valued Logic*, pages 9–14, Barcelona, Spain, 2010. IEEE. doi:10.1109/ISMVL.2010.11.
16. Joao Marques-Silva and Ines Lynce. On Improving MUS Extraction Algorithms. In Karem A. Sakallah and Laurent Simon, editors, *Theory and Applications of Satisfiability Testing - SAT 2011*, volume 6695, pages 159–173. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. doi:10.1007/978-3-642-21581-0_14.

17 Alexander Nadel, Vadim Ryvchin, and Ofer Strichman. Efficient MUS extraction with resolution. In *2013 Formal Methods in Computer-Aided Design*, pages 197–200, October 2013. doi:10.1109/FMCAD.2013.6679410.

18 Alexander Nadel, Vadim Ryvchin, and Ofer Strichman. Accelerated Deletion-based Extraction of Minimal Unsatisfiable Cores. *Journal on Satisfiability, Boolean Modelling and Computation*, 9(1):27–51, June 2014. doi:10.3233/SAT190100.

19 Yash Puranik and Nikolaos V Sahinidis. Deletion presolve for accelerating infeasibility diagnosis in optimization models. *INFORMS Journal on Computing*, 29(4):754–766, 2017.

20 Francesca Rossi, Peter Van Beek, and Toby Walsh. *Handbook of constraint programming*. Elsevier, 2006.

21 Roussel, Olivier. Pseudo-Boolean Competition of 2025. https://www.cril.univ-artois.fr/PB25/, 2025.

22 Vadim Ryvchin and Ofer Strichman. Faster Extraction of High-Level Minimal Unsatisfiable Cores. In Karem A. Sakallah and Laurent Simon, editors, *Theory and Applications of Satisfiability Testing - SAT 2011*, volume 6695, pages 174–187. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. doi:10.1007/978-3-642-21581-0_15.

23 Hans Van Maaren and Siert Wieringa. Finding Guaranteed MUSes Fast. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, Gerhard Weikum, Hans Kleine Büning, and Xishun Zhao, editors, *Theory and Applications of Satisfiability Testing – SAT 2008*, volume 4996, pages 291–304. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. doi:10.1007/978-3-540-79719-7_27.

24 Toby Walsh. Sat v csp. In *International Conference on Principles and Practice of Constraint Programming*, pages 441–456. Springer, 2000.

25 Siert Wieringa. Understanding, Improving and Parallelizing MUS Finding Using Model Rotation. In Michela Milano, editor, *Principles and Practice of Constraint Programming*, volume 7514, pages 672–687. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. doi:10.1007/978-3-642-33558-7_49.

26 Jianmin Zhang, ShengYu Shen, Jun Zhang, Weixia Xu, and Sikun Li. Extracting minimal unsatisfiable subformulas in satisfiability modulo theories. *Comput. Sci. Inf. Syst.*, 8(3):693–710, 2011. doi:10.2298/CSIS101019024Z.