

On the Superiority of Conflict-Driven Search in MUS Enumeration

Roxane Koitz-Hristov¹ and Franz Wotawa¹

¹Graz University of Technology, Graz, Austria
{rkoitz,wotawa}@ist.tugraz.at

Abstract

The extraction of minimal unsatisfiable cores from an unsatisfiable set of constraints is a computationally hard problem that finds application in a variety of tasks such as model checking, configuration, or diagnosis. Domain-agnostic algorithms for online minimal unsatisfiable subset enumeration allow the computation of all conflicts and can be applied to any type of constraint system. We aim at extending this research by combining two well-known approaches from different research communities; on the one hand, we exploit the traversal of the power set as suggested by the MARCO algorithm in the domain of infeasibility analysis and on the other hand, we take advantage of the implicit exploration of the search space as proposed by HS-DAG in model-based diagnosis. In particular, we show that the conflict-driven search utilized by HS-DAG renders MARCO's SAT calls unnecessary and given a certain problem structure a combination of both is advantageous in domains where consistency checks are expensive.

1 Introduction

Various artificial intelligence tasks can be formulated as constraint satisfaction problems. There are many scenarios where the underlying constraint set can become over-determined and in these cases, we are interested in explanations of the infeasibility. In the Boolean domain, these parsimonious explanations are referred to as Minimal Unsatisfiable Subsets (MUSes). Given an unsatisfiable formula, a minimal unsatisfiable subset (MUS) is a subset of clauses, such that removing any clause of the MUS turns the formula satisfiable. In recent years, the importance of computing minimal unsatisfiable subsets for formal verification has led to a significant number of work in this area. For instance, minimal unsatisfiable cores are used for abstraction refinement [1] and debugging of declarative specifications [2]. Besides formal verification, MUS extraction has been a topic of interest in different fields. Product configuration is a typical domain where systems can become over-constrained, since the customer's needs and technical constraints can contradict one another [3]. In the context of model-based diagnosis, Reiter [4] exploits MUSes to derive parsimonious diagnoses based on the hitting set relation

between minimal conflicts, i.e., MUSes, and minimal diagnoses, i.e., Minimal Correction Subsets (MCSes). Later, Greiner et al. [5] proposed a variation of Reiter's approach named Hitting Set Directed Acyclic Graph (HS-DAG) that corrects a fault in the original procedure regarding non-minimal conflicts.

Besides research on single MUS extraction algorithms [6; 7; 3; 8], the need to derive all MUSes for various constraint types has prompted the development of domain-agnostic approaches [9; 10; 11; 12]. These methods can be applied to any type of constraint set as there are no dependencies between the features of the constraints and the algorithms. Furthermore, by exploiting domain specific MUS extraction routines, these approaches can capitalize on any advancement in single MUS computation for a specific constraint type.

Although these methods may exploit specialized MUS extraction procedures, Bendik et al. [13] observe that these constraint-agnostic algorithms are not necessarily efficient when used in domains other than the Boolean one. The reason is that consistency checks are time-consuming for systems based on more expressive representations than Boolean formulas. Many practical problems, however, require a richer modeling language and thus can be more naturally formulated as an SMT instance such as spreadsheet debugging [14]. In these cases, ideally, the number of performed satisfiability checks during MUS enumeration would be minimized.

In this paper, we compare HS-DAG as proposed in the context of model-based diagnosis to the recently developed online MUSes computation procedure MARCO [12]. While the relation between MUS enumeration and HS-DAG has been mentioned previously [10], we describe how HS-DAG can be modified to obtain all MUSes. As our presented HS-DAG version ensures conflicts are minimized before continuing with the construction of the graph, the pruning methods as corrected by Greiner et al. [5] are not necessary. Additionally, we show that HS-DAG's strategy to conquer the search space is similar to MARCO's. However, while both methods have the benefit of being domain independent and anytime algorithms, HS-DAG does not require any SAT solver calls in contrast to MARCO. Before concluding the paper, we present a combination of HS-DAG and MARCO that can reduce the number of constraint solver calls on certain samples as well as an initial experiment.

2 Preliminaries

Given a finite set of constraints \mathcal{C} , an MUS is a set of constraints that cannot be satisfied simultaneously, while every proper subset of an MUS is consistent.

Definition 1 (Minimal Unsatisfiable Subset (MUS)). *Given an inconsistent set of constraints \mathcal{C} , a subset $U \subseteq \mathcal{C}$ is an MUS if U is inconsistent and $\forall u \in U : U \setminus \{u\}$ is consistent.*

MUSes can be computed either directly or via exploiting their hitting set dual MCSes [4]. An irreducible hitting set h for a set of sets Σ is a subset of $\bigcup_{\sigma \in \Sigma} \sigma$ such that $\forall \sigma \in \Sigma : \sigma \cap h \neq \emptyset$ and there exists no other hitting set h' for Σ such that $h' \subseteq h$.

Definition 2 (Minimal Correction Subset (MCS)). *Given an inconsistent set of constraints \mathcal{C} , a subset $M \subseteq \mathcal{C}$ is an MCS if $\mathcal{C} \setminus M$ is consistent and $\forall m \in M : \mathcal{C} \setminus (M \setminus \{m\})$ is inconsistent.*

An MCS contains constraints that correct the inconsistency when removed. Each complement of an MCS is a maximal set of constraints that is satisfiable and is referred to as a Maximal Satisfiable Subset (MSS).

Definition 3 (Maximal Satisfiable Subset (MSS)). *Given an inconsistent set of constraints \mathcal{C} , a subset $S \subseteq \mathcal{C}$ is an MSS if S is consistent and $\forall s \in (\mathcal{C} \setminus S) : S \cup \{s\}$ is inconsistent.*

Example Consider the set of Boolean constraints \mathcal{C} with $c_1 = a$, $c_2 = \bar{a}$, $c_3 = \bar{a} \vee b$, and $c_4 = \bar{b}$. The combination of constraint c_1 with c_2 and c_1 with c_3 and c_4 results in \mathcal{C} being inconsistent; hence, $\text{MUSes}(\mathcal{C}) = \{\{c_1, c_2\}, \{c_1, c_3, c_4\}\}$. By hitting set computation we derive the following $\text{MCSes}(\mathcal{C}) = \{\{c_1\}, \{c_2, c_3\}, \{c_2, c_4\}\}$ and subsequently can determine all Maximal Satisfiable Subsets (MSSes): $\text{MSSes}(\mathcal{C}) = \{\{c_2, c_3, c_4\}, \{c_1, c_4\}, \{c_1, c_3\}\}$.

3 Related Work

MUS enumeration procedures can be categorized into direct and indirect approaches. While the former rely on the enumeration of constraint subsets to determine their satisfiability, the latter exploit the hitting set relation between MCSes and MUSes. The first proposed direct approaches utilize a tree-like structure to examine every subset of constraints in regard to its feasibility [15; 16]. Independently from one another Previti and Marques-Silva [10] as well as Liffiton and Malik [11] proposed to iteratively enumerate all MSSes and MUSes based on the idea that the power set of all constraints can be represented as a Boolean formula. Later, their approaches were merged into the MARCO algorithm [12]. By adapting the Boolean formula whenever a subset of interest is identified, it is ensured that already explored portions of the power set are not considered again in the search. Derivations of MARCO include, for instance, MUSesHunter [17] and TOME [13]. MUSesHunter focuses on deriving MUSes by blocking supersets as well as subsets of found MUS and generating MUSes even during the search for MSSes. TOME relies on the notion of chains of the power set and local MUSes/MSSes. The algorithm differentiates from MARCO, as during its main loop only local MSSes and local MUSes are constructed. Once the entire lattice has been explored the MSSes and MUSes are extracted from the local MSSes and MUSes via subset inclusion checks.

Indirect approaches rely on the duality between MUSes and MCSes. The CAMUS algorithm [18] first collects all

MCSes and afterward obtains the MUSes via irreducible hitting set computation. Bailey and Stuckey [9] rely on the same notion and present an interleaved process between computing an MSS/MCS and exploiting the symmetry to extract MUSes. In the context of model-based diagnosis, Stern et al. [19] introduce a method that given a collection of MUSes already computed derives a new hitting set. Based on the satisfiability of this hitting set, it is either an MCS or another MUS can be computed. Given the duality between MCSes and MUSes, the algorithm can be exploited as a conflict-directed search for diagnoses or a diagnosis-directed search for conflicts. Liffiton et al. [12] compared CAMUS to MARCO and observed that exploiting the hitting set duality is more runtime efficient than computing MUSes directly. However, as the number of MCSes can be exponential in the number of constraints in \mathcal{C} , the first phase in which the MCSes are enumerated may be intractable. In these cases, CAMUS cannot output any MUSes, while MARCO iteratively returns at least some conflicts.

4 MARCO and HS-DAG

In this section, we first discuss the general idea behind MARCO and HS-DAG. We show that the way these two approaches explore the subsets of constraints is similar. However, as HS-DAG is a conflict-driven search method, it is advantageous in comparison to MARCO as it does not require an explicit representation of the search space, but the strategy is implicitly encoded within the construction of the graph. Hence, in Section 4.3 we propose a combination of both methods that exploits the structure of HS-DAG and the search space representation of MARCO.

4.1 Exploration of the Power Set with MARCO

As mentioned MARCO [12] incrementally computes MUSes and MSSes by exploring the power set of constraints. The method exploits that an MUS defines a ‘‘low point’’ in an infeasible region of the power lattice, while an MSS is a ‘‘high point’’ in a satisfiable part. Each iteration of the approach starts with a seed representing an unexplored subset of constraints. To determine regions within the power lattice not yet processed, MARCO relies on a Boolean formula encoding of constraint subsets whose feasibility still has to be determined. Given the Boolean formula representing a map of the search space, a SAT solver computes the next seed as a satisfying truth assignment of the formula. To establish whether the constraint set returned as seed lies within a satisfiable or unsatisfiable region, a suitable constraint solver checks the consistency of the subset of constraints. If it is unsatisfiable, an MUS is extracted by employing any single MUS extraction method. Otherwise, the subset is expanded until a maximal satisfiable constraint set, an MSS, is obtained.

Whenever an MUS (MSS) is found, the Boolean formula is updated to exclude all supersets (subsets) of the MUS (MSS) from later iterations. To block all supersets of an MUS from further consideration a clause $B_{\uparrow} = \bigvee_{m \in \text{MUS}} \neg m$ is added to the Boolean formula, while a clause $B_{\downarrow} = \bigvee_{m \notin \text{MSS}} m$ is appended to exclude the subsets of a found MSS. The next iteration then starts again by determining an unexplored point in the lattice. Once the Boolean formula has become unsatisfiable the entire search space has been processed and thus all MUSes and MSSes have been uncovered.

In Algorithm 1 we show MARCO’s pseudo code. First, the Boolean formula representing the *map* is defined. As long as the *map* is satisfiable, there are still MUSes/MSSes to compute. In the simplest scenario, a single SAT solver call returns any satisfiable truth assignment of the *map* representing an unexplored constraint subset. Instead of using an arbitrary satisfying truth assignment as seed, calculating the maximal model (`getMaxModel`) of the Boolean formula, i.e., the maximum number of literals is true without violating a clause, to determine an entry point as high as possible within the lattice biases the procedure towards MUSes [10; 20]. This strategy ensures that each returned satisfiable constraint set already represents an MSS. The computation of a maximal model, however, involves several SAT solver calls instead of just a single one in case of a simple truth assignment. Once a seed has been obtained, its feasibility is determined by a constraint solver (`isConsistent`). Each inconsistent constraint set is reduced to an MUS using a single MUS extraction algorithm (`MUSExtraction`), while a consistent seed represents an MSS. After the computation of an MUS/MSS, the Boolean formula is updated accordingly.

Input : unsatisfiable constraint set \mathcal{C}
Output: all MUSes and MSSes of \mathcal{C}

```

1 map  $\leftarrow$  BoolFormula( $|\mathcal{C}|$ );
2 while map is satisfiable do
3   | seed  $\leftarrow$  getMaxModel(map);
4   | if isConsistent(seed) then
5   |   | outputMSS(seed);
6   |   | map  $\leftarrow$  map  $\wedge$  blockSubsets(seed);
7   | else
8   |   | MUS  $\leftarrow$  MUSExtraction(seed);
9   |   | outputMUS(MUS);
10  |   | map  $\leftarrow$  map  $\wedge$  blockSupersets(MUS);
11  |   end
12 end

```

Algorithm 1: MARCO [12]

Example (cont.) Given our constraint set from before, we can apply MARCO as shown in Table 1. Starting with the first seed, the method `getMaxModel`¹ returns $\{c_1, c_2, c_3, c_4\}$. Since the constraint set is inconsistent, it is reduced to the first MUS $\{c_1, c_2\}$, as marked by an ellipse-shaped blue node in the lattice in Figure 1. In Table 1, we again see the same color coding of the MUS in the first row. After uncovering the first MUS the clause $\bar{c}_1 \vee \bar{c}_2$ is added to the *map*, which marks all supersets of the MUS $\{c_1, c_2\}$ as explored. We indicate this in Figure 1 by coloring all supersets of $\{c_1, c_2\}$ and the paths leading from $\{c_1, c_2\}$ to its supersets in the same blue color.

The second maximal seed is $\{c_2, c_3, c_4\}$, which is consistent and thus represents an MSS. Again we color the node in the lattice—this time in red and with a rectangular-shaped node to indicate that it is an MSS. As every subset of an MSS is consistent as well, the clause c_1 is added to *map* blocking all constraint sets further down in the lattice. At this point *map* = $(\bar{c}_1 \vee \bar{c}_2) \wedge (c_1)$, hence it is satisfiable. Once the last MSS $\{c_1, c_3\}$ has been uncovered, the *map* becomes unsatisfiable, indicating that all MUSes and MSSes have been found.

¹Assuming Arif et al.’s [20] maximal model method.

4.2 Conflict-Driven Search via HS-DAG

Reiter’s [4] approach for computing parsimonious diagnoses is based on the duality between MCSes and MUSes and operates on a tree, which is constructed in a breadth-first manner. Each node is either labeled with a conflict or constitutes a minimal diagnosis, i.e., a minimal hitting set of all conflicts. In the original version, to obtain the refutations on demand, the algorithm relies on a theorem prover that returns conflicting constraints in case they exist. Each node n is equipped with an edge label $H(n)$, containing all edge labels on the path from the root to node n . Starting from the root node consisting of an empty edge label, whenever a new node n is added to the tree it is checked for consistency using all constraints except the ones in $H(n)$. Suppose the solver returns a conflict, then the node is marked with the refutation, which is guaranteed to be disjoint to the current node’s edge label; otherwise, the node is a leaf and its edge label represents a minimal diagnosis, i.e., MCS. Each node characterized by a conflict is expanded such that for each element c in the conflict an edge is created and labeled c . The child node’s edge label then contains the path label of the parent combined with c . Some inadequacies of the original algorithm in regard to non-minimal conflict sets were corrected by Greiner et al. [5] and they devised HS-DAG operating on a directed acyclic graph instead of a tree.

HS-DAG can be modified to compute all MUSes either by ensuring that the constraint solver returns MUSes instead of arbitrary conflicts or by minimizing each returned refutation to a parsimonious one right away. Algorithm 2² shows a modified HS-DAG that (1) does not rely on a theorem prover returning conflicts, but only assumes that a call to `isConsistent` returns true if the constraint set is consistent and false otherwise and that (2) extracts a MUS whenever a node is inconsistent (Line 20). Starting from the root, the algorithm operates level-wise and first tests whether the currently processed node can be closed due to being the superset of an already derived MCS (`checkClose`). If it remains open, the algorithm examines all previously computed conflicts for reuse (`checkReuseConflict`). A MUS can be reused if it is disjoint to the node’s edge label. If there is such a conflict, then this MUS is used to expand the node in Line 12 to 14; otherwise, $\mathcal{C} \setminus H(n)$ is checked for consistency. In case it is consistent, the node n represents an MCS, is marked \checkmark , and the MCS is communicated; otherwise, the unsatisfiable set of constraints is minimized to an MUS and subsequently, the MUS is outputted. The current node n is then labeled with the conflict and for each element of the MUS a new child is created with the edge label of the node and the conflict element as path label. Once there are no more nodes to process, the algorithm terminates and all MUSes and MCSes have been computed.

Example (cont.) Figure 2 and Table 2 depict the HS-DAG and its execution given the constraint set. For the root node with an empty edge label, the algorithm checks the consistency of all constraints in the constraint set. This set of constraints is inconsistent. We assume that the MUS extraction algorithm first returns the MUS $\{c_1, c_2\}$. Node n_0 thus is marked $\{c_1, c_2\}$ and for each element in the conflict a new edge and child is created. Note here that we used the same

²As we ensure that all returned conflicts are reduced to an MUS, the pruning steps corrected by Greiner et al. [5] for non-minimal refutations are obsolete.

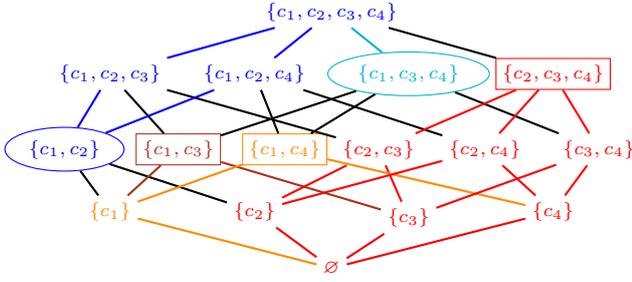


Figure 1: Power set explored with MARCO

# SAT	seed	isConsistent	MUS/MSS	append to map
5	$\{c_1, c_2, c_3, c_4\}$	false	$\{c_1, c_2\}$	$B_{\uparrow} = \bar{c}_1 \vee \bar{c}_2$
5	$\{c_2, c_3, c_4\}$	true	$\{c_2, c_3, c_4\}$	$B_{\downarrow} = c_1$
5	$\{c_1, c_3, c_4\}$	false	$\{c_1, c_3, c_4\}$	$B_{\uparrow} = c_1 \vee \bar{c}_3 \vee \bar{c}_4$
4	$\{c_1, c_4\}$	true	$\{c_1, c_4\}$	$B_{\downarrow} = c_2 \vee c_3$
2	$\{c_1, c_3\}$	true	$\{c_1, c_3\}$	$B_{\downarrow} = c_2 \vee c_4$

Table 1: Sample execution of MARCO

color coding as in Figure 1 and Table 1 to explicitly show the connection between the two algorithms. The first MUS in the execution of MARCO is equivalent to this first conflict of HS-DAG. Hence, both are colored in the same blue in their corresponding figures and tables.

At node n_1 the constraint set \mathcal{C} minus the edge label c_1 is checked for consistency, which in this case is $\{c_2, c_3, c_4\}$. As this set is consistent it represents an MSS, the node is marked \checkmark , and the MCS $\{c_1\}$ is returned. The consistency check in n_2 reveals that $\{c_1, c_3, c_4\}$ is not satisfiable and subsequently the second MUS is extracted. Node n_3 can be closed as its path label $\{c_1, c_2\}$ is a subset of the already identified MCS $\{c_1\}$. n_4 and n_5 are consistent, hence, the edge labels represent MCSes and the computation terminates as no more nodes to process remain.

4.3 Combining MARCO and HS-DAG

It is apparent that given its exhaustive search HS-DAG computes all MUSes and all MSSes by deriving the MCSes. In contrast to MARCO, it implicitly avoids already explored regions of the search space through its construction and closing rules instead of explicitly encoding it within a Boolean formula. This is advantageous as in MARCO every time a seed is generated a satisfying assignment is computed and given that the maximal model is utilized a set of successive calls to a SAT solver is necessary [20].

HS-DAG does not need to create a seed because the strategy to conquer the search space is given in the way the graph is constructed, and in addition, it always utilizes a maximal model by checking the consistency of the constraint set minus the path label. In case the construction would lead to a region in the power lattice already processed, such as in node n_3 where the consistency of $\{c_3, c_4\}$ would be checked even though the feasibility of $\{c_2, c_3, c_4\}$ is already known, the algorithm takes advantage of the previously computed MCSes/MSSes, i.e., nodes marked \checkmark . Thus, HS-DAG does not require any SAT calls, but subset checks with all already computed MCSes. Without considering the MUS extraction procedure, MARCO requires five SAT calls to determine the status of the map, sixteen satisfiability checks to compute the maximal models for the seeds, and five constraint solver calls to determine the feasibility of the seeds. In contrast, HS-DAG requires five consistency checks to determine the

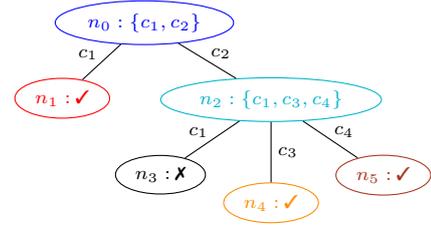


Figure 2: HS-DAG constructed

n	$H(n)$	$\mathcal{C} \setminus H(n)$	isConsistent	MUS/MCS
n_0	\emptyset	$\{c_1, c_2, c_3, c_4\}$	false	$\{c_1, c_2\}$
n_1	$\{c_1\}$	$\{c_2, c_3, c_4\}$	true	$\{c_1\}$
n_2	$\{c_2\}$	$\{c_1, c_3, c_4\}$	false	$\{c_1, c_3, c_4\}$
n_3			closed	
n_4	$\{c_2, c_3\}$	$\{c_1, c_4\}$	true	$\{c_2, c_3\}$
n_5	$\{c_2, c_4\}$	$\{c_1, c_3\}$	true	$\{c_2, c_4\}$

Table 2: Sample execution of HS-DAG

Input : unsatisfiable constraint set \mathcal{C}

Output: all MUSes and MCSes of \mathcal{C}

```

1 root ← new Node( $\emptyset$ );
2 nodesToProcess ← < root >;
3 while nodesToProcess  $\neq \emptyset$  do
4   newNodes ←  $\emptyset$ ;
5   foreach  $n \in$  nodesToProcess do
6     if checkClose( $H(n)$ ) then
7        $n.mark \leftarrow \mathcal{X}$ ;
8       continue;
9     end
10     $CO \leftarrow$  checkReuseConflict( $H(n)$ );
11    if  $CO \neq \emptyset$  then
12      foreach  $c \in CO$  do
13        newNodes ←
14          newNodes  $\cup$  new Node( $c \cup H(n)$ );
15      end
16    else
17      if isConsistent( $\mathcal{C} \setminus H(n)$ ) then
18         $n.mark \leftarrow \checkmark$ ;
19        outputMCS( $H(n)$ );
20      else
21         $MUS \leftarrow$  MUSExtraction( $\mathcal{C} \setminus H(n)$ );
22        outputMUS( $MUS$ );
23         $n.mark \leftarrow MUS$ ;
24        foreach  $c \in MUS$  do
25          newNodes ←
26            newNodes  $\cup$  new Node( $c \cup H(n)$ );
27        end
28      end
29    end
30  end
31  nodesToProcess ←
32    nodesToProcess  $\cup$  newNodes;
33 end

```

Algorithm 2: HS-DAG modified from Reiter [4] and Greiner et al. [5]

satisfiability of the nodes and five subset checks to examine whether nodes can be closed.

The version we have presented of MARCO favors the construction of MUSes early on by calculating a maximal model for the seed. HS-DAG already implicitly utilizes a maximal seed as proposed by Previti and Marques-Silva [10] in the context of MARCO. For many purposes finding all MUSes is sufficient, i.e., the byproduct of MSSes is not required. In these scenarios, MARCO can further be tailored towards MUS enumeration by blocking also subsets of MUSes. Since all subsets of an MUS are by definition satisfiable, marking these constraint sets as explored does not block any other MUSes [17]. This strategy may exclude some MSSes from being computed, yet it speeds up the process of deriving all explanations as the search space is pruned more efficiently and fewer consistency checks are necessary.

Even though HS-DAG is a conflict-directed search approach, we can only be sure that all MUSes have been uncovered once the entire graph has been constructed as there is no explicit representation of the power set as in MARCO allowing us to block up and down whenever an MUS is obtained. Hence our idea is to exploit on the one hand the implicit exploration given by the construction of the graph in case of HS-DAG and on the other hand the termination criteria utilized by MARCO. Algorithm 3 shows the necessary adaptations of HS-DAG. In particular, we create a Boolean formula for the map and update it whenever an MUS or MCS is found. If the node is an MCS, we mark all subsets of its complement as explored and for an MUS, we block all subsets and supersets of the conflict. After each adaptation of the map, we check whether the map is still satisfiable. If not we are sure to have outputted all MUSes and thus can stop the procedure. The correctness and completeness of our approach are directly given through the results of the original HS-DAG (Theorem 4.1 in [5]) and the Boolean formula encoding (Theorem 1 in [12]).

```

1  map ← BoolFormula(|C|);
15 ...
16 if isConsistent(C \ H(n)) then
17   n.mark ← ✓;
18   outputMCS(H(n));
19   map ← map ∧ blockSubsets(C \ H(n));
20   if map is unsatisfiable then
21     return;
22   end
23 else
24   MUS ← MUSExtraction(C \ H(n));
25   outputMUS(MUS);
26   map ← map ∧ blockSupersets(MUS) ∧
      blockSubsets(MUS);
27   if map is unsatisfiable then
28     return;
29   end
30   ...
31 end
32 ...

```

Algorithm 3: HS-DAG adaptation

Example (cont.) Considering the example from before, we simply keep a *map* as does MARCO and update it whenever we encounter a new MUS or MCS/MSS. The compu-

tation again starts with the root in Figure 3, where we check the consistency of $\{c_1, c_2, c_3, c_4\}$ and retrieve the first MUS $\{c_1, c_2\}$. We append $(\bar{c}_1 \vee \bar{c}_2)$ and $(c_3 \vee c_4)$ to the *map* as shown in Table 3. Adding these clauses to *map* marks all supersets and subsets of $\{c_1, c_2\}$ as explored in the lattice in Figure 4. As before, we continue with n_1 . n_1 is consistent, thus it represents a minimal hitting set and we include the clause c_1 in the *map*. At node n_2 , we encounter the second MUS $\{c_1, c_3, c_4\}$ and in order to block up and down add $(\bar{c}_1 \vee \bar{c}_3 \vee \bar{c}_4)$ and (c_2) to the Boolean formula. Now, $map = ((\bar{c}_1 \vee \bar{c}_2) \wedge (c_3 \vee c_4) \wedge (c_1) \wedge (\bar{c}_1 \vee \bar{c}_3 \vee \bar{c}_4) \wedge (c_2))$, thus has become unsatisfiable. As this indicates that the entire lattice has been processed, the execution is terminated.

This adaptation of HS-DAG still computes all MUSes in an iterative and constraint-agnostic fashion. Yet keeping the map can avoid unnecessary operations and consistency checks in the last level of the graph. The simple adaptation we have shown does not require any more consistency checks, but only additional SAT calls specifically a single satisfiability check of the map at every node. Comparing the number of SAT calls for MARCO and the adapted HS-DAG on the example, we see that MARCO requires over all twenty-one satisfiability checks, while the adapted HS-DAG only requires three. In regard to constraint solver calls, the original HS-DAG and MARCO require five calls, while the adapted HS-DAG can terminate after three consistency checks.

Certain observations are crucial. The early detection that all MUSes have been computed is only possible in cases, where at least a single MSS is a subset of an MUS. This is apparent as this is the only possibility an MSS can be marked as explored by an MUS before it is explicitly derived. Consider the set of MUSes = $\{\{c_1, c_2\}, \{c_3, c_4\}\}$ with the corresponding MSSes = $\{\{c_1, c_3\}, \{c_1, c_4\}, \{c_2, c_3\}, \{c_2, c_4\}\}$. Blocking up and down when uncovering the MUSes does not reach the MSSes as none of these are a subset of the MUSes. Even if an MSS is a subset of an MUS there is no guarantee that our approach can terminate early. Particularly, an MSS, which is not a subset of an MUS, generated in the last level of the HS-DAG cannot be excluded a-priori. Thus, this approach is not applicable in all cases. Another observation from Algorithm 3 is that there are as many satisfiability checks of the map as there are calls to the constraint solver. This entails that even in cases where an early termination of the HS-DAG is possible, the SAT calls need to be inexpensive in comparison to consistency checks in order to gain computational speed.

4.4 Initial Experiments

In an initial experimental study, we compare HS-DAG to MARCO and to our adaptation of HS-DAG to determine the extent to which consistency and satisfiability checks may be saved using our method. Hence, we do not focus on runtime, but on the number of constraint/SAT solver calls. All methods are implemented in Java and to bias MARCO towards MUSes we exploit the strategy to use a maximal model for the seed [20]. As a SAT solver to derive the satisfiability of the map we utilize SAT4J³ [21]. By means of a sample generator, we randomly constructed 100 artificial conflict sets, i.e., MUSes, with different overlap, i.e., the number of shared elements, between the conflicts. For each ex-

³www.sat4j.org/

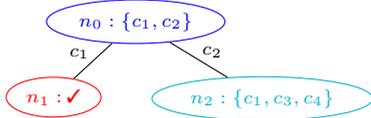


Figure 3: Adapted HS-DAG

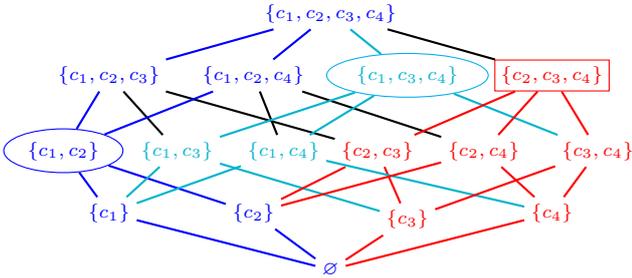


Figure 4: Power set explored with the adapted HS-DAG

n	# SAT	isConsistent	MUS/MCS	appended to map
n_0	1	false	$\{c_1, c_2\}$	$B_{\uparrow} = (\bar{c}_1 \vee \bar{c}_2), B_{\downarrow} = (c_3 \vee c_4)$
n_1	1	true	$\{c_1\}$	$B_{\downarrow} = (c_1)$
n_2	1	false	$\{c_1, c_3, c_4\}$	$B_{\uparrow} = (\bar{c}_1 \vee \bar{c}_3 \vee \bar{c}_4), B_{\downarrow} = (c_2)$

Table 3: Sample execution of the adapted HS-DAG

ample, we recorded (1) the overall number of MUSes, i.e., $|\text{MUSes}|$, (2) the minimum, maximum, mean, and standard deviation of the size of a MUS, and (3) the minimum, maximum, mean, and standard deviation of the overlap between MUSes. The rows in Table 4 present the results for the entire set of examples. Since our experiments contain artificial MUSes, we utilized a mock-up consistency checker to determine whether a given set of elements is satisfiable. As we are interested in the difference between SAT and constraint solver calls between the approaches, we assume that all approaches rely on the same MUS extraction procedure for which we do not count the consistency checks required.

Each algorithm was invoked ten times on each example. The results show that HS-DAG is preferable over MARCO due the avoidance of any SAT-calls in the conflict-driven search. On our instances, both approaches require on average 18.7 constraint solver calls (MAX=142, SD=18.2), while MARCO additionally needs on average 181.8 satisfiability checks (MAX=1381, SD=193.4). Thus, even in cases where SAT calls are rather inexpensive in comparison to constraint solver queries, MARCO always requires a computation overhead from the SAT calls. Comparing HS-DAG to our adaptation, it is apparent that on samples where a premature termination is not possible our approach leads to additional SAT calls. This was the case for 35% of our samples. Considering all examples, we record on average 11.3% savings in consistency checks (MAX=76.5%, SD=19.9%) with our method in comparison to the traditional HS-DAG.

	$ \text{MUSes} $	$ \text{MUS} $				overlap			
		MIN	MAX	AVG	SD	MIN	MAX	AVG	SD
MIN	1.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0
MAX	12.0	20.0	20.0	20.0	5.9	18.0	18.0	18.0	1.5
AVG	3.9	7.3	8.1	7.7	0.4	2.2	3.0	2.4	0.3
SD	2.9	7.6	7.3	7.4	0.7	5.2	5.2	5.2	0.4

Table 4: Example statistics

5 Conclusion

In this paper, we have examined and compared two approaches to the MUS enumeration problem: HS-DAG and MARCO. HS-DAG has first been proposed as a conflict-driven search for diagnoses in the context of model-based diagnosis, while MARCO is a recently developed iterative MUS enumeration algorithm. Both approaches are very similar and share that they are constraint type independent and generate MUSes online. Yet, while MARCO explicitly records already explored regions via its Boolean map, HS-DAG implicitly prunes the search space via its construction procedure and node closing rules. Even though MARCO is the state of the art direct anytime MUS enumeration procedure, we argue that HS-DAG’s avoidance of explored regions is advantageous as both algorithms require the same number of consistency checks, while MARCO requires additionally SAT calls.

An advantage of MARCO is that it can further be tailored to traverse the search space faster by blocking with each generated MUS all its supersets as well as subsets. This technique biases the search even more towards MUS and may overlook certain MSSes. To exploit this focus strategy in addition to the search space traversal of HS-DAG, we adapted HS-DAG by tracking the constraint subsets explored using MARCO’s map representation. In a simple version, we examine the map’s satisfiability after a new MCS or MUS is uncovered. Since in this case every consistency check is associated with an additional SAT call, the approach is only convincing in scenarios where expenses for a constraint consistency check outweigh the costs of a SAT call. Furthermore, the suitability of our adaptation of HS-DAG depends on the structure of the underlying constraint set. In particular, in order to obtain any improvements at least a single MSS has to be a subset of an MUS to avoid explicitly computing the MSS.

For future work we plan on analyzing applications and their constraints to determine for which systems our method is advantageous. In addition, it would be interesting to compare our adaptation to other improvements of the HS-DAG such as 22’s [22] RC-Tree.

References

- [1] Kenneth L McMillan and Nina Amla. Automatic abstraction without counterexamples. In *Proceedings of the 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 2–17. Springer-Verlag, 2003.
- [2] Emina Torlak, Felix Chang, and Daniel Jackson. Finding minimal unsatisfiable cores of declarative specifications. *FM 2008: Formal Methods*, pages 326–341, 2008.
- [3] Ulrich Junker. QUICKXPLAIN: preferred explanations and relaxations for over-constrained problems. In *Proceedings of the 19th National Conference on Artificial Intelligence*, pages 167–172. AAAI Press, 2004.
- [4] Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987.
- [5] Russell Greiner, Barbara A Smith, and Ralph W Wilkerson. A correction to the algorithm in Reiter’s theory of diagnosis. *Artificial Intelligence*, 41(1):79–88, 1989.

- [6] Renato Bruni. Approximating minimal unsatisfiable subformulae by means of adaptive core search. *Discrete Applied Mathematics*, 130(2):85–100, 2003.
- [7] Yoonna Oh, Maher N Mneimneh, Zaher S Andraus, Karem A Sakallah, and Igor L Markov. AMUSE: A minimally-unsatisfiable subformula extractor. In *Proceedings of the 41st Annual Design Automation Conference*, pages 518–523. ACM, 2004.
- [8] Nachum Dershowitz, Ziyad Hanna, and Alexander Nadel. A scalable algorithm for minimal unsatisfiable core extraction. *Proceedings of the 9th International Conference on Theory and Applications of Satisfiability Testing*, pages 36–41, 2006.
- [9] James Bailey and Peter Stuckey. Discovery of minimal unsatisfiable subsets of constraints using hitting set dualization. *Practical Aspects of Declarative Languages*, pages 174–186, 2005.
- [10] Alessandro Previti and Joao Marques-Silva. Partial MUS enumeration. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*, pages 818–825. AAAI Press, 2013.
- [11] Mark H Liffiton and Ammar Malik. Enumerating infeasibility: Finding multiple MUSes quickly. In *International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 160–175. Springer, 2013.
- [12] Mark H Liffiton, Alessandro Previti, Ammar Malik, and Joao Marques-Silva. Fast, flexible MUS enumeration. *Constraints*, 21(2):223–250, 2016.
- [13] Jaroslav Bendík, Nikola Benes, Ivana Cerná, and Jirí Barnat. Tunable Online MUS/MSS Enumeration. In Akash Lal, S. Akshay, Saket Saurabh, and Sandeep Sen, editors, *36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2016)*, volume 65 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 50:1–50:13, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
- [14] Simon Außerlechner, Sandra Fruhmann, Wolfgang Wieser, Birgit Hofer, Raphael Spörk, Clemens Mühlbacher, and Franz Wotawa. The right choice matters! SMT solving substantially improves model-based debugging of spreadsheets. In *Proceedings of the 2013 13th International Conference on Quality Software*, pages 139–148. IEEE Computer Society, 2013.
- [15] Aimin Hou. A theory of measurement in diagnosis from first principles. *Artificial Intelligence*, 65(2):281–328, 1994.
- [16] Benjamin Han and Shie-Jue Lee. Deriving minimal conflict sets by CS-trees with mark set in diagnosis from first principles. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 29(2):281–286, 1999.
- [17] Thai Son Hoang, Shinji Itoh, Kyohei Oyama, Kuni-hiko Miyazaki, Hironobu Kuruma, and Naoto Sato. Consistency verification of specification rules. In *International Conference on Formal Engineering Methods*, pages 50–66. Springer, 2015.
- [18] Mark H Liffiton and Karem A Sakallah. Algorithms for computing minimal unsatisfiable subsets of constraints. *Journal of Automated Reasoning*, 40(1):1–33, 2008.
- [19] Roni Stern, Meir Kalech, Alexander Feldman, and Gregory Provan. Exploring the duality in conflict-directed model-based diagnosis. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, pages 828–834. AAAI Press, 2012.
- [20] M Fareed Arif, Carlos Mencía, and Joao Marques-Silva. Efficient MUS enumeration of Horn formulae with applications to axiom pinpointing. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 324–342. Springer, 2015.
- [21] Daniel Le Berre and Anne Parrain. The sat4j library, release 2.2, system description. *Journal on Satisfiability, Boolean Modeling and Computation*, 7:59–64, 2010.
- [22] Ingo Pill and Thomas Quaritsch. RC-tree: A variant avoiding all the redundancy in reiter’s minimal hitting set algorithm. In *Software Reliability Engineering Workshops (ISSREW), 2015 IEEE International Symposium on*, pages 78–84. IEEE, 2015.