ECAI 2012
Luc De Raedt et al. (Eds.)
© 2012 The Author(s).
This article is published online with Open Access by IOS Press and distributed under the terms of the Creative Commons Attribution Non-Commercial License.
doi:10.3233/978-1-61499-098-7-648

Optimizations for the Boolean Approach to Computing Minimal Hitting Sets

Ingo Pill and **Thomas Quaritsch**[1]

Abstract. The Boolean approach to computing minimal hitting sets proposed by Lin and Jiang is known to offer very attractive general performance, but also has its issues, specifically with a cardinality-restricted search. In this paper we propose optimizations regarding the refinement rules, also offering a revised decision strategy as well as optimized termination criteria that exploit cardinality bounds. Our experiments including artificial and real-world samples for the bounded and unbounded case show the potential of our work, where we could achieve speed-ups of up to two orders of magnitude.

1 INTRODUCTION

When describing a problem as a set of structures, minimal hitting sets that contain at least one element per structure often encode important aspects, and may even provide the solutions to essential problems. For instance, in model-based diagnosis, the minimal hitting sets for a set of unsatisfiable cores (conflict sets) provide the diagnoses [4, 9, 5], AI planning may profit from minimal hitting sets for a set of landmarks [2], and in software debugging hitting sets of program slices are used to identify program faults [12]. Even such low-level tasks as transforming conjunctive and disjunctive normal forms may draw on minimal hitting sets. The widespread application options entailed the development of many competing approaches, e.g. [9, 5, 4, 11, 1], where some more recent ones [10, 3] may trade performance with completeness. That is, aiming at *some* minimal solution, they might shed others during optimization stages, as long as the derived one is in fact a minimal one. For some applications, one's attention might be however on *complete* approaches, that is those that can deliver the entire set of subset-minimal hitting sets. Lin and Jiang proposed in [7] such an approach considering the sets to hit in a structured way, and showed how to efficiently implement their idea using bits for components. This Boolean approach is well-known to offer very good performance, but also has its limitations. That is, if we establish cardinality limits on the the desired solutions (due to resource limits, or if too complicated solutions are not of interest), the Boolean approach, unlike the approaches of Reiter [9, 5] (HS-DAG) and Wotawa [11] (HST), might struggle with exploiting these search space limitations. In this paper we focus on such issues and propose optimizations that help to tackle them.

We organized our paper as follows. We give a formal introduction of the minimal hitting set problem and the Boolean approach in Section 2. Section 3 offers an analysis of the issues at hand and provides the details of our optimizations. In Section 4, we discuss our test scenarios, with Section 4.1 focusing on the test platform and our results. Finally, we draw our conclusions in Section 5.

[1] Institute for Software Technology, Graz University of Technology, Inffeldgasse 16b/2, 8010 Graz, Austria, email: {ipill,quaritsch}@ist.tugraz.at

2 THE MINIMAL HITTING SET PROBLEM AND THE BOOLEAN ALGORITHM

According to Reiter [9], a hitting set and a minimal hitting set are formally defined as follows:

Definition 1 *Given a set of sets SCS, a set* $h \subseteq \bigcup_{CS_i \in SCS} CS_i \subseteq$ *COMP is a* hitting set *for SCS, iff for any set* $CS_i \in SCS$ *the intersection with h is non-empty, i.e.* $h \cap CS_i \neq \emptyset$.

Definition 2 *Given a hitting set h for some SCS, h is said to be* minimal *with respect to subset inclusion, iff there exists no other hitting set* h' *for SCS, such that* $h' \subset h$.

While from here on we assume subset-minimality when referring to *minimal hitting sets (MHSs)* there are also other attractive dimensions for minimality. Let us consider the following example.

Example 1 *Let SCS be the set* $\{\{1,2,5\},\{2,4\},\{2,3\}\}$. *Then, the sets* $\{2\}$, $\{1,4,3\}$, *and* $\{5,4,3\}$ *are (subset-)minimal hitting sets for SCS, while obviously the first one is much smaller than the latter two.*

Thus, when an MHS's cardinality correlates with a problem solution's complexity, one might actually consider cardinality restrictions on the search space, specifically as some algorithms such as HS-DAG [9, 5] and HST [11] tend to profit greatly from such restrictions via the resulting depth limits on their internal trees. Sometimes also limits on computation resources require us to focus only on the most attractive (in the current context smallest) solutions.

Another option in the same direction is to exploit probabilities (weights). That is, in the context of model-based diagnosis, de Kleer and Williams define the probability of an MHS (a diagnosis Δ) [4] as $\prod_{x\in\Delta} p_F(x) \cdot \prod_{x\in COMP\setminus\Delta}(1-p_F(x))$, where *COMP* is the set of components, and p_F defines the probability of $c \in$ *COMP*'s appearance in Δ while assuming stochastic independence between the components' involvement. While in this paper we take cardinality limits into account, our findings can be extended to consider other metrics like probabilities.

As mentioned earlier, in 2003 Lin and Jiang proposed the Boolean approach [7] using bits (propositions) for components $e \in$ *COMP* in order to derive all minimal hitting sets for a given *SCS*. They encode *SCS* as a Boolean formula in disjunctive normal form (DNF), with the conjuncts encoding the individual $CS_i \in$ *SCS* and consisting of the corresponding (negated) element bits. A recursive function $H(C)$ containing five rules (considered in ascending order) derives from this *SCS formula* another formula encoding the MHSs. That is, the result still needs some subset-checks (or the use of Boolean laws) in order to derive a canonical DNF where the conjuncts represent the individual *MHS*s. Assuming C a Boolean formula, e an atomic proposition with $\bar{e}$ denoting its negation, and *False*/*True* referring to $e \wedge \bar{e}/e \vee \bar{e}$, $H(C)$ is defined as:

R1: $H(\mathit{False}) = \mathit{True}, H(\mathit{True}) = \mathit{False}$;
R2: $H(\bar{e}) = e$;
R3: $H(\bar{e} \wedge C) = e \vee H(C)$;
R4: $H(\bar{e} \vee C) = e \wedge H(C)$;
R5: $H(C) = e \wedge H(C_1) \vee H(C_2)$ for some *arbitrary* atomic proposition e present in C, with $C_1 = \{c_i \mid c_i \in C \wedge \bar{e} \notin c_i\}$ and $C_2 = \{c_i \mid \bar{e} \notin c_i \wedge (c_i \in C \vee c_i \cup \{\bar{e}\} \in C)\}$.

Please note that like Lin and Jiang we consider a conjunction also as a set of elements. R5 encodes the algorithm's general strategy of how to conquer the search space. That is, splitting on some proposition $e \in \mathit{COMP}$, the algorithm forks two branches; the "left" one that considers those solutions containing e (and thus subsequently focusing on those CS_is not hit so far), while the "right" one assumes that e is not part of the solution (with a further focus on all sets, but with $\bar{e}$ removed from the problem description – C is replaced by C_2). R1 to R4 resolve specific situations, that is, R4 covers the situation when there is a CS_i with only one element (there is one obvious choice then), R3 and R2 those situations where $|\mathit{SCS}| = 1$, and R1 resolves *True* and *False*.

Obviously, the decision heuristic choosing the split element e in R5 has a significant impact on the actual traversal of the search space. A common heuristic (which we will refer to as H1) is to use one of those e's that hit the most CS_is. This ensures that the left branch has to deal only with a minimum of conjunctions/sets ($|C_1|$ is minimized), and for the right branch a maximum of conjunctions/sets shrink by one element. In practice, in the unbounded case, this heuristic offers very good general performance (see also Figure 1). But there are issues with H1 in the bounded case (see Figures 2 and 3), that is, when we establish cardinality limits, the performance is not that attractive.

3 THE BOOLEAN APPROACH: ON ISSUES AND OPTIMIZATIONS

Figures 1, 2 and 3 compare the Boolean approach's performance with that of HS-DAG [9, 5] for a test scenario composed of random conflict sets (see Section 4 for a detailed description). While the standard Boolean approach (Bool-Rec-V1-R4) easily outperforms HS-DAG in the unbounded case, it does so only for small $|\mathit{SCS}|$ in the bounded case. This fact motivated us to consider options for improving the Boolean approach's performance.

As R1 to R3 offer little to no room for improvements[2], let us have a closer look at Rule 4 (R4). This rule considers the situation, where SCS contains some CS_i of size one ($\mathit{CS}_i = \{e\}$). In this case, the algorithm makes the obvious decision of including element e in the current branch. However, the recursion still includes also those CS_is hit by e, so that we propose a new variant R4'.

Lemma 1 *Replacing R4 with R4' as follows does not affect the algorithm's correctness. Let R4' be $H(\bar{e} \vee C) = e \wedge H(C_1)$ with $C_1 = \{c_i \mid c_i \in C \wedge \bar{e} \notin c_i\}$*

Proof. By replacing C with those $\mathit{CS}_i \in C$ not hit by e, we loose those elements e' for future choices in $H(C)$ which are in $C \setminus C_1$. This has no ill effect at all, as for any element e'' in C, we have that

- if e'' could hit some CS_i in C_1, it would have to be present in C_1.
- if e'' cannot hit some CS_i in C_1, then it cannot hit any *further* set in C not hit so far. Remember that e was chosen in this step and hits exactly those $\mathit{CS}_i \in C \setminus C_1$. Thus choosing such an element e'' would result in non-minimal conjunctions in the hitting set formula to be removed later. □

Intuitively, R4' implements the same idea to remove unnecessary input-data from future consideration as the left branch of R5.

In the prior section we described a common heuristic H1 that offers very good performance in the unbounded case. In the bounded case, however, the Boolean performance using H1 is not that attractive. That is, while for rule R5 any "left" branch adds an element to the solution and thus increases cardinality, with H1 the amount of right branches that have to be considered is limited only by the number of components in C. Considering the case when we limit $|\mathit{MHS}|$ to 1, and we could stop after considering all the elements in some CS_i (any CS_i has to be hit), H1 seems rather unattractive. We thus propose to use the following heuristic H2 that chooses elements in a minimal-sized CS_i.

Definition 3 *Let R5, C_1, and C_2 be as defined above, but instead of some arbitrary e, use heuristic H2, that is some $e \in \mathit{CS}_i \in C$ such that there is no $\mathit{CS}_j \in C$ with $|\mathit{CS}_j| < |\mathit{CS}_i|$.*

Like for H1, the validity of H2 follows from the fact that as allowed by the original paper we could have chosen any $e \in \mathit{elements}(C)$, and we choose some $e \in \mathit{CS}_i \subseteq \mathit{elements}(C)$.

Intuitively, H2 strives to "clear" some (minimal) CS_i as fast as possible, where for the last element in CS_i R4 takes over. When discussing the options for cardinality-limit related *breaks* in the algorithm later on, it will become clear that this effectively limits the amount of right branches to be considered. If there are multiple sets of minimal length, we might encounter some (non-harming) non-determinism resulting in "hopping" between those sets. For any recursive call of $H(C)$ removing another element from a minimal CS_i, we also have the checks whether R1 to R4 would apply (even when $|\mathit{CS}_i| > 1$ and $|\mathit{SCS}| > 1$, which requires R5). Thus we propose the following variant of R5 that loops on a single CS_i directly.

Lemma 2 *Adapting R5' as follows does not affect the algorithm's correctness:*

$$H\left(C \vee \bigwedge_{i=1}^{n} \bar{e}_i\right) = \bigvee_{i=1}^{n-1} \left(e_i \wedge H(C_1^i)\right) \vee H(C^{n-1})$$

where $C^0 = C$, $\forall c_k \in C : |c_k| \geq n$, and the sets C^i and C_1^i are defined as

$$C^i = \{c_j \mid \bar{e}_i \notin c_j \wedge (c_j \in C^{i-1} \vee c_j \cup \{\bar{e}_i\} \in C^{i-1})\},$$
$$C_1^i = \{c_j \mid c_j \in C^{i-1} \wedge \bar{e}_i \notin c_j\}.$$

Proof. An essential pre-condition for the correctness is our focus on some cardinality-minimal CS_i as ensured by the prerequisites. Thus, when removing the $n-1$ elements from CS_i in the sequence of the sets C_1^i, there exists no other CS_j such that its refinements could trigger R1 to R4. Intuitively, and implementing the general split mechanism of R5, the disjunction of terms $e_i \wedge H(C_1^i)$ offers those terms established by the left branches of each recursion of the original R5, when choosing the same sequence of elements in CS_i for H2 as split elements (the right branch acting as interface between the recursive calls). $H(C^{n-1})$ is the last missing right branch, i.e. the one that triggers R4 in order to deal with the n-th element in CS_i. □

[2] R3 considers the case of SCS containing a single CS_i, so an implementation might use a direct loop in R3 instead of recursive calls to H leading to some intermediate checks whether R1 or R2 would apply.

The following example illustrates Lemma 2, where we abbreviate $\bar{b}\ \wedge \bar{c}\ \wedge \bar{g}$ with $\bar{b}\,\bar{c}\,\bar{g}$, and loop on the underlined CS_i.

Example 2 $H(C^0) = H(\bar{b}\,\bar{c}\,\bar{g} \vee \bar{a}\,\bar{f}\bar{g} \vee \bar{a}\,\bar{c}\,\bar{d}\,\bar{e} \vee \underline{\bar{b}\,\bar{e}\,\bar{f}})$
$= b\,H(C_1^1) \vee e\,H(C_1^2) \vee H(\overline{C^2})$, *where*
$C_1^1 = \xcancel{\bar{b}\,\bar{c}\,\bar{g}} \vee \bar{a}\,\bar{f}\bar{g} \vee \bar{a}\,\bar{c}\,\bar{d}\,\bar{e} \vee \xcancel{\bar{b}\,\bar{e}\,\bar{f}}$, $C_1^2 = \bar{c}\,\bar{g} \vee \bar{a}\,\bar{f}\bar{g} \vee \cancel{\bar{a}\,\bar{c}\,\bar{d}\,\bar{e}} \vee \xcancel{\bar{e}\bar{f}}$,
$C^1 = \xcancel{\bar{b}}\bar{c}\,\bar{g} \vee \bar{a}\,\bar{f}\bar{g} \vee \bar{a}\,\bar{c}\,\bar{d}\,\bar{e} \vee \xcancel{\bar{b}}\bar{e}\,\bar{f}$, $C^2 = \bar{c}\,\bar{g} \vee \bar{a}\,\bar{f}\bar{g} \vee \bar{a}\,\bar{c}\,\bar{d}\,\xcancel{\bar{e}} \vee \xcancel{\bar{e}}\bar{f}$.

R5' makes the connection between our strategy as implemented in both H2 and R5' with the tree-construction used in HS-DAG most obvious, as we capture all the essential details that allow HS-DAG to cut the depth of the internal tree. Thus, specifically for scenarios with $|MHS| = 1$, we expect a huge performance benefit.

Other options for performance gains are unveiled when focusing on restriction-related termination criteria in the algorithm. That is, in contrast to simply discarding larger solutions, the computation will focus only on branches that *can* produce viable *MHS*s. Explicitly keeping track of an intermediate solution's cardinality enables the following intuitive, trivial adaptions of $H(C)$:

Lemma 3 *Given a bound b,* $H(C, 0, b)$ *using the following adapted rules (termination criteria variant I) derives a Boolean formula encoding all MHSs with* $1 \leq |MHS| \leq b$:

RB1: $H(False, \ell, b) = True, H(True, \ell, b) = False;$
RB2: $H(\bar{e}, \ell, b) = False \text{ if } \ell \geq b, \text{ else } e;$
RB3: $H(\bar{e} \wedge C, \ell, b) = False \text{ if } \ell \geq b, \text{ else } e \vee H(C, \ell, b);$
RB4: $H(\bar{e} \vee C, \ell, b) = False \text{ if } \ell \geq b, \text{ else } e \wedge H(C, \ell + 1, b);$
RB5: $H(C, \ell, b) = False \text{ if } \ell \geq b, \text{ else } e \wedge H(C_1, \ell + 1, b) \vee H(C_2, \ell, b).$

Proof. The correctness follows from that of the original algorithm and the following considerations regarding a branch's potential for adding elements to the intermediate solution possibly resulting in violations of the new postcondition regarding bound b. Obviously, ℓ corresponds exactly to the size of a branch's intermediate solution, and returning *False* in some rule would remove the current branch from consideration. R1 cannot add to the solution, so it is left unchanged for RB1. For R2 that adds an element, any $\ell \geq b$ would result in a breach of bound b, so that we return *False* then in RB2. Also R3 would add at least one element in any case (the recursion would be in RB3 or RB2), so that we return *False* in RB3 for $\ell \geq b$. The same is obvious for RB4 and RB5 (where for R5 the right branch would add at least one element in the recursion to RB2-RB5). Thus only those branches leading to solutions violating the post-condition are removed from the original computation. □

Intuitively, if the length of an intermediate solution has reached the bound b, only R1 has to be considered for refinement. Any other rule would result in adding at least one further element to the current branch. Therefore, then only $e_1 \wedge e_2 \wedge \cdots \wedge e_b \wedge H(False, b, b)$ may lead to an *MHS* of length b.

Taking a closer look at situations $H(C, b-1, b)$ (i.e. we can add only one further element), unveils the potential for another optimization, as then only those elements present in all CS_is are of interest.

Lemma 4 *Let RB1' to RB4' be as RB1 to RB4, and RB5' as follows (termination criteria variant II). Then* $H(C, 0, b)$ *derives a Boolean formula encoding all MHSs with* $1 \leq |MHS| \leq b$.

$$RB5'\text{:}\ \ H(C, \ell, b) = \begin{cases} False & \text{if } \ell \geq b \vee I = \emptyset, \\ \bigvee_{e_j \in I} e_j & \text{if } \ell = b - 1, \\ \text{else } e \wedge H(C_1, \ell + 1, b) \vee H(C_2, \ell, b) & \end{cases}$$

with $I = \bigcap_{CS_i \in C} CS_i$ *and* C_1 *and* C_2 *as defined previously.*

Proof. This lemma's correctness follows directly from that for variant I and the fact that for $\ell = b - 1$ only and exactly those elements in the intersection of all CS_i in C can complete the current decisions to encode further MHS candidates. □

Corollary 1 *No combination of the proposed modifications in the form of Lemmas 1 to 4 and adopting H2 does affect the correctness of the algorithm.*

Proof. The corollary's correctness directly follows from the individual proofs and the absence of preconditions. □

4 EVALUATION

In our experiments, we evaluated both run-time and memory consumption of the Boolean approach in various configurations implemented in Python (cf. [8] for an evaluation of some MHS algorithm implementations in Java and Python), comparing it against Reiter's HS-DAG [9, 5] which we also implemented in Python. We investigated whether any advantages would manifest only under certain conditions, and report a conclusive selection of our results.

Regarding an implementation, there is the obvious option of a recursive approach, but one might also consider an iterative implementation that maintains a list of intermediate work-packages (containing tuples of intermediate Cs (*SCS*s) and established solution parts) to be iteratively resolved. Some advantage would be the option to be able to (stop and) continue the computation (i.e. with an increased cardinality limit). For our evaluation we implemented both options.

Test Scenario TS1: Artificial random CS_is $\in$ *SCS*. In this scenario, an *SCS* contains elements drawn randomly from a set of components, i.e. every component is included in a CS_i with a probability of 0.5. For this random setting we evaluated the bounded and unbounded case, with a well-sized sample set in order to avoid any bias from a specific *random* pattern.

Test Scenarios TS2 to TS4 : Real-world scenarios based on ISCAS benchmarks. To evaluate whether our findings from TS1 would transfer to real applications, for TS2 to TS4 we constructed *SCS*s as they would occur in logic circuit diagnosis for the ISCAS'85 benchmark suite [6]. This suite provides ten circuits such as interrupt controllers, modules for single-error-correction (SEC), double-error-detection (DED) and arithmetic logic units (ALU) with 160 to 3512 gates. Our test scenarios were constructed from the circuits *c499.isc* (32bit SEC, TS2), *c880.isc* (8bit ALU, TS3), and *c1355.isc* (32bit SEC, TS4) having 41/60/41 inputs, 32/26/32 outputs, and 202/383/546 gates respectively. Equipping every gate g with a behavioral assumption $\text{AB}(g)$ that encodes whether it operates correctly or not, we defined a SAT Problem P of the form

$$P = \bigwedge_{g_i \in G} \left(\neg \text{AB}(g_i) \Rightarrow \text{out}_{g_i} := f_{g_i}(\text{in}^1_{g_i}, \text{in}^2_{g_i}, \ldots)\right)$$

where G is the set of gates, and out_{g_i}, $\text{in}^j_{g_i}$ and f_{g_i} are a gate g_i's output-/input signals and its Boolean function. The CS_is are those unsatisfiable cores of these assumptions that violate the original, contradicting input-output observations. For our scenarios, we purposefully injected a single fault by altering the logic function of a gate and computed *SCS* as the set of CS_is derived during a cardinality-restricted ($|MHS| \leq 3$) on-the-fly diagnosis run using HS-DAG and Yices (`http://yices.csl.sri.com`) as theorem prover.

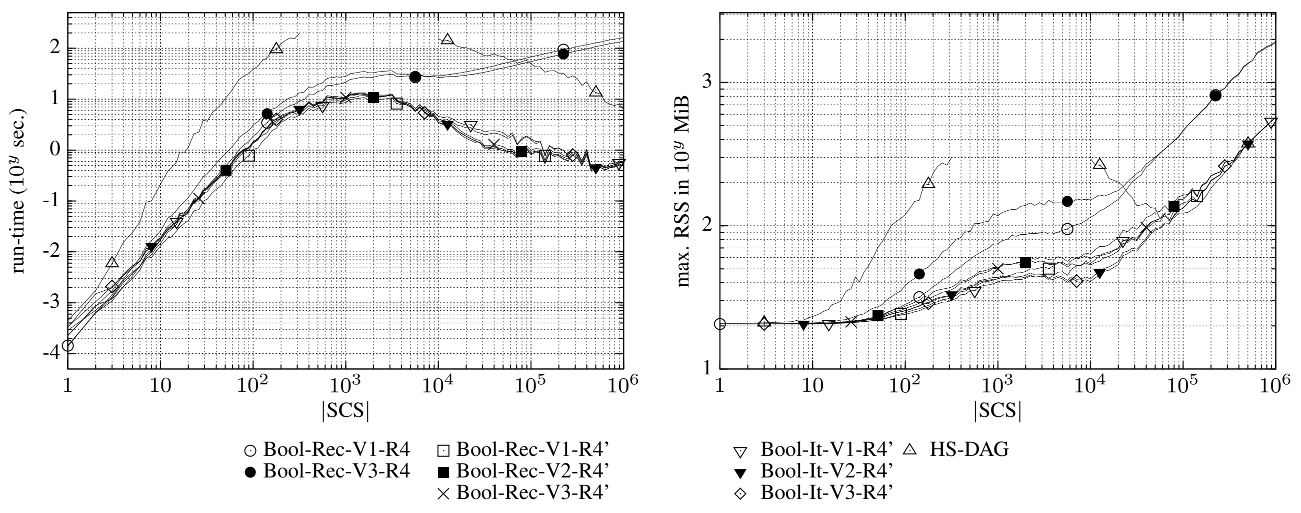

Figure 1. Run-times for TS1, $|COMP|=20$, and an unbounded MHS search.

4.1 Experimental results

Our experiments were executed on a 2011 generation MacBook Pro (Intel Core i5 CPU, 2.3 GHz, 4GiB RAM, SSD, Mac OS X 10.6) using CPython 2.7.1. With swapping and the GUI disabled, each sample faced resource limits of 300 seconds and 2GiB RAM. Regarding memory usage, we polled the process' resident set size (RSS) from the operating system in a separate process, and report its maximum. We plot run-times and memory usage on logarithmic scales, with approximately 120 points equally distributed on the x-axis (e.g. for Figure1 we have $|SCS| \approx 10^{6i/120}$ for $0 \leq i \leq 120$).

Figure 1 shows run-time and maximum RSS averaged over 20 samples for TS1 in the unbounded case with $|COMP| = 20$ and a varying $|SCS|$. V1 denotes using H1, while V2 amounts to using H2, and V3 implements Lemma 2. R4 vs. R4' should be self-explaining, while Rec indicates a recursive implementation and It an iterative one. The first thing to observe is that all Boolean variants are about one order of magnitude faster than HS-DAG in the range $[10, 300]$ and also consume significantly less memory. While using R4' ensures this also for larger *SCS*, the performance advantage diminishes otherwise, so that HS-DAG can outperform the Boolean approach when an *SCS* gets larger than approx. $6 \cdot 10^4$. While variants V2 and V3 suffer minor drawbacks for small $|SCS|$, they slightly gain in performance for higher amounts of CS_is. All variants using R4' feature very similar memory characteristics. Using a recursive or iterative implementation does not result in huge differences, with the recursive ones being a touch faster and the iterative ones a bit more memory-effective. While we were motivated mostly by issues with bounded computations, these results suggest that there is little to no computation overhead in the unbounded case, and specifically R4' can also help in an unbounded search. Please note that the missing plot segments for HS-DAG stem from (run-time) limit violations.

Figure 2 shows our results for TS1 and $|MHS|=1$. Those variants implementing variant II of the termination criteria are indicated by the suffix "Stop", all others use variant I. The graphs at the top plot the recursive implementations' performance and illustrate that, as expected, R4' has no influence on time or memory usage at all (due to the very small probability of *SCS* containing a CS_i of size one). Compared to V1 using termination criteria I, variants V2 and V3 already allow a significant boost, surpassed however by that offered by termination criteria II. For the latter the strategy also does not play a significant role, which is intuitive given the small amount of recursions allowed by those criteria. Obviously, as in this case Rule 5 is never really executed (in fact, only the intersection of all conflict sets is computed), all heuristics share the same time- and memory characteristics. Since our recursive and iterative implementations again perform similarly, we will not plot the iterative ones for our further figures due to space restrictions. Overall, while HS-DAG's performance is still superior for $|SCS|$ larger than about 10^3, the original threshold was below ten. Furthermore, we enhanced the performance of the Boolean approach by up to two orders of magnitude.

A last evaluation of TS1 is shown in Figure 3, where we restricted $|MHS|$ to 3. Like for small samples in the unbounded case, we encountered a negative impact of R4' (for the majority of the smaller samples) when using the original heuristic V1. For the other variants the impact of R4' diminishes, specifically when using termination criteria II. Again V3 outperforms V2, that in turn outperforms V1. Also for these tests, we could significantly raise the border where HS-DAG would take the lead from about 20 to approx. 200.

Finally, Figure 4 shows the performance of several implementations for 300 real-world samples (100 samples for each of TS2 to TS4) with cardinality limits $\{1, 2, 3\}$. Due to random fault injection, samples with similar $|SCS|$ may still have incommensurable structural complexity, leading to a large variance in both run-time and memory. In order to observe otherwise obscured trends, we applied a moving average filter that considers for any $|SCS|$ x_0 on the x-axis all samples within the range $\left[x_0/\sqrt{2}, x_0\sqrt{2}\right]$. Considering the graphs, we find that our conclusions from the artificial scenario would also transfer to these *SCS*s as extracted from a real-world application. While for the original heuristic (V1) R4' results in a run-time penalty, this drawback vanishes for termination criteria II. The variant ranking V3–V2–V1 is confirmed, as, e.g., evident from the graph for $|MHS| \leq 2$. However, the most notable result follows from the comparison with HS-DAG. While for $|MHS|=1$ and $|SCS| \geq 5$, HS-DAG is faster than the original Boolean variant, our termination criteria II alone (then the variant has almost no impact) leads to an average advantage of one order of magnitude compared to HS-DAG. When raising the maximum cardinality to $|MHS| = 2$ and 3, more and more Boolean variants catch up, until for max. $|MHS|=3$ only one of them (V1-R4') is slower than HS-DAG. Also the ranking V3-V2-V1 becomes more apparent then. Summing up, with our optimizations the Boolean approach is not only a performant contender in the unbounded case, but also for a bounded *MHS* search.

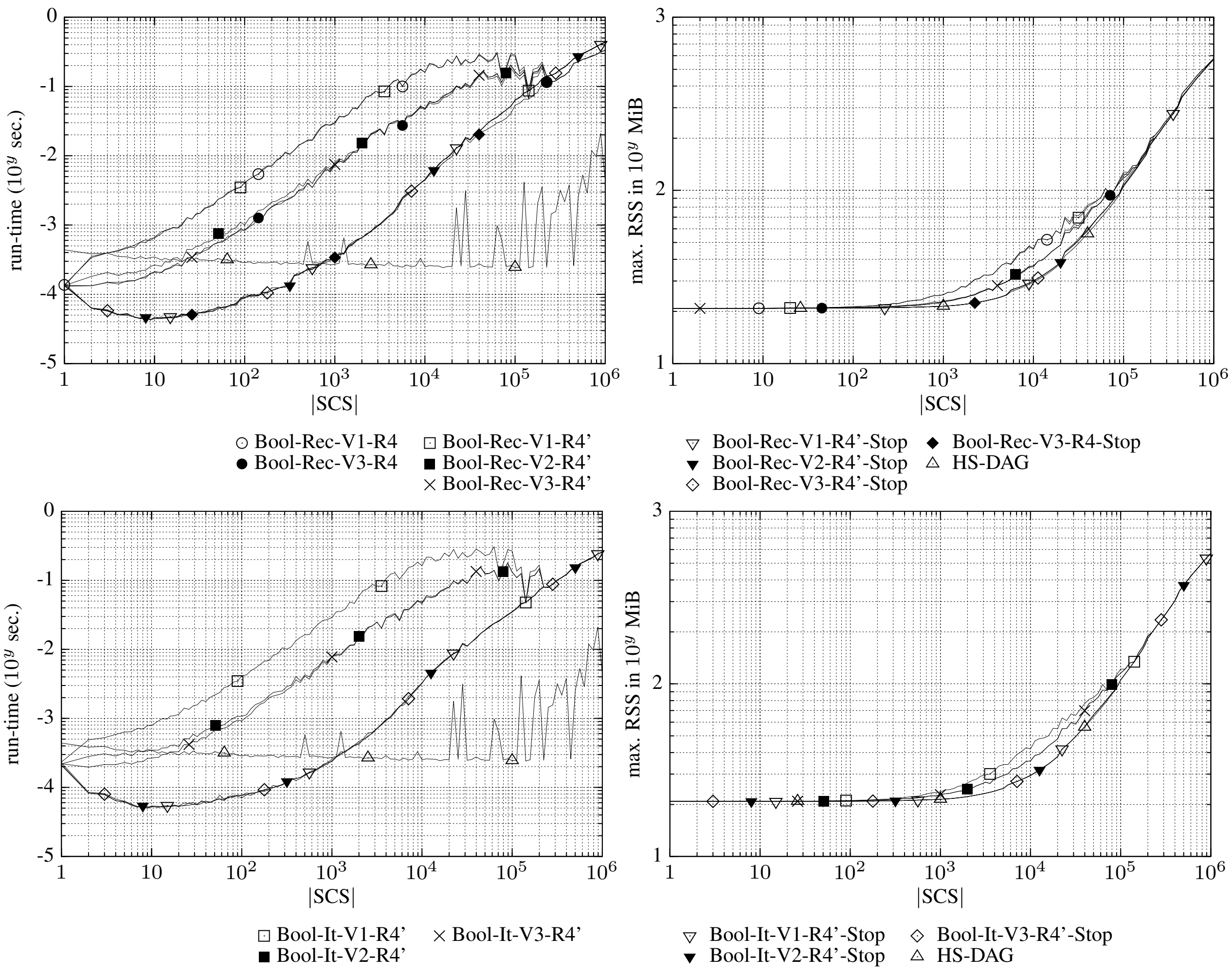

Figure 2. Results for TS1, $|COMP| = 20$, $|MHS| = 1$ with iterative versions in the bottom and recursive ones in the top graphs.

5 CONCLUSIONS

In this paper we showed how to improve the Boolean approach's performance by up to 2 orders of magnitude for artificial and real-world scenarios for which it was known to perform badly before. That is, we showed a new heuristic, a revised split strategy, as well as tight termination rules that tackle earlier disadvantages when conquering the search space for cardinality-bounded problems. Our experiments suggest that our optimizations would not significantly hinder performance in the unbounded case, and could sometimes also enhance it. We also proposed a slightly improved Rule 4 that avoids some duplicates and non-minimal solutions to be pruned anyway. Thus, the Boolean algorithm now is also an attractive alternative for the bounded case. In future work, we will try to tackle the Boolean algorithm's main drawback that it requires *SCS* to be known in advance, providing an interesting option for our main research regarding enabling model-based diagnosis of formal temporal specifications.

ACKNOWLEDGEMENTS

This work has been funded by the Austrian Science Fund (FWF) under grant P22959-N23. We would like to thank the reviewers for their comments and Franz Wotawa for fruitful discussions.

REFERENCES

[1] R. Abreu and A. van Gemund, 'A low-cost approximate minimal hitting set algorithm and its application to model-based diagnosis', in *Symposium on Abstraction Reformulation, and Approximation*, (2009).

[2] B. Bonet and M. Helmert, 'Strengthening landmark heuristics via hitting sets', in *Europ. Conf. on Art. Intelligence*, pp. 329–334, (2010).

[3] J. de Kleer, 'Hitting set algorithms for model-based diagnosis', in *Int. Workshop on the Principles of Diagnosis*, pp. 100–105, (2011).

[4] J. de Kleer and B. C. Williams, 'Diagnosing multiple faults', *Artificial Intelligence*, **32**(1), 97–130, (1987).

[5] R. Greiner, B. A. Smith, and R. W. Wilkerson, 'A correction to the alg. in Reiter's theory of diagnosis', *Art. Intelligence*, **41**(1), 79–88, (1989).

[6] M. Hansen, H. Yalcin, and J. P. Hayes, 'Unveiling the ISCAS-85 benchmarks: A case study in reverse engineering', *IEEE Design and Test*, **6**, 72–80, (1999). (http://www.cbl.ncsu.edu/benchmarks/ISCAS85).

[7] L. Lin and Y. Jiang, 'The computation of hitting sets: review and new algorithms', *Information Processing Letters*, **86**, 177–184, (2003).

[8] I. Pill, T. Quaritsch, and F. Wotawa, 'From conflicts to diagnoses: An empirical evaluation of minimal hitting set algorithms', in *22nd Int. Workshop on the Principles of Diagnosis*, pp. 203–210, (2011).

[9] R. Reiter, 'A theory of diagnosis from first principles', *Artificial Intelligence*, **32**(1), 57–95, (1987).

[10] L. Shi and X. Cai, 'An exact fast algorithm for minimum hitting set', in *Int. Joint Conference on Computational Science and Optimization - Vol. 01*, pp. 64–67, (2010).

[11] F. Wotawa, 'A variant of Reiter's hitting-set algorithm', *Information Processing Letters*, **79**, 45–51, (2001).

[12] F. Wotawa, 'On the relationship between model-based debugging and program slicing', *Artificial Intelligence*, **135**, 125–143, (2002).

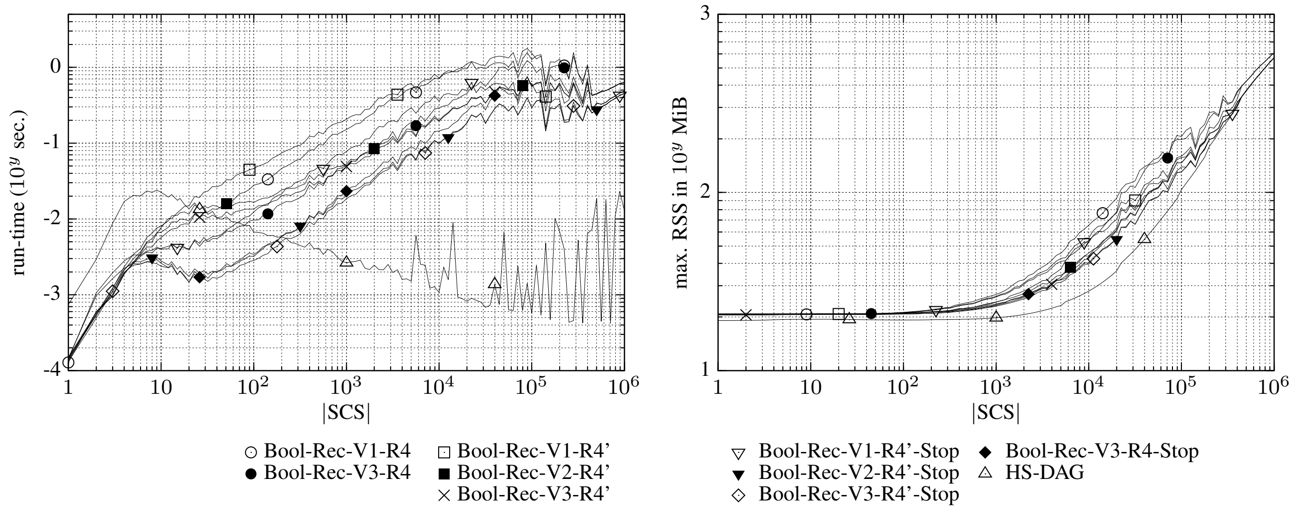

Figure 3. Results for TS1, $|COMP| = 20$, max. $|MHS| = 3$.

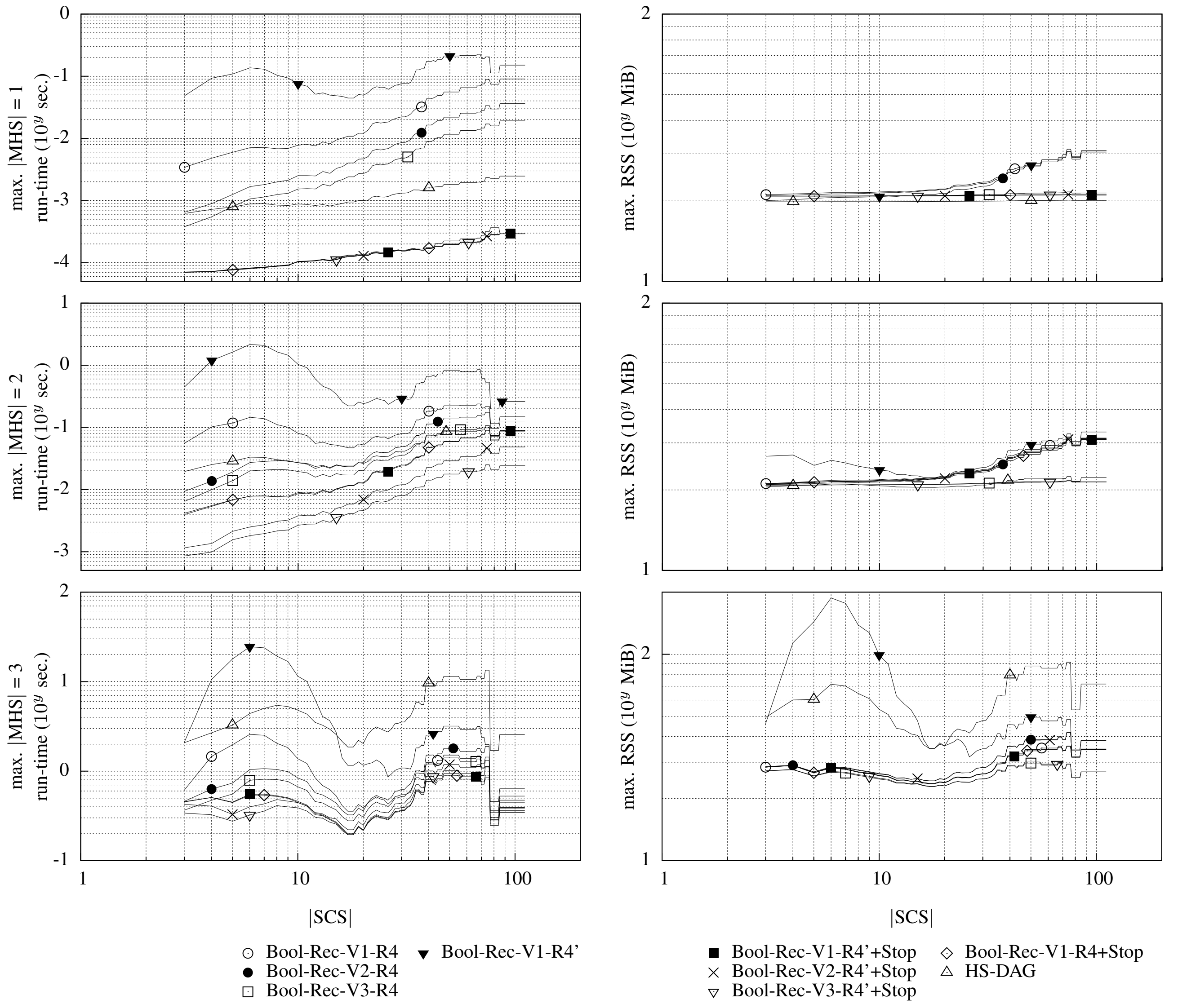

Figure 4. Results for TS2 to TS4 with varying max. $|MHS|$.