HELSINKI INSTITUTE FOR INFORMATION TECHNOLOGY

Depth-based Local Search Heuristics for Boolean Circuit Satisfiability

Belov, Anton; Järvisalo, Matti

HIIT Technical Reports 2010-2

Depth-based Local Search Heuristics for Boolean Circuit Satisfiability

Anton Belov
York University
Department of Computer Science and Engineering
4700 Keele Street
Toronto, Ontario
Canada M3J 1P3
antonb@cse.yorku.ca

Matti Järvisalo
University of Helsinki
Department of Computer Science
P.O. Box 68
FI-00014 University of Helsinki
Finland
matti.jarvisalo@cs.helsinki.fi

Helsinki Institute for Information Technology HIIT
HIIT Technical Reports 2010-2
ISBN 978-952-60-3415-7 (electronic)
ISSN 1458-9478 (electronic)

Depth-based Local Search Heuristics for Boolean Circuit Satisfiability[⋆]

Anton Belov[1] and Matti Järvisalo[2]

[1] Department of Computer Science and Engineering, York University, Canada
[2] Department of Computer Science, University of Helsinki, Finland

Abstract. We propose a structure-exploiting heuristic for the justification-based stochastic local search (SLS) method CRSat for Boolean circuit satisfiability. Experimental evaluation shows that the proposed depth-based heuristic significantly improved the performance of CRSat on structural instances arising from industrial applications. A proof of probabilistically approximate completeness (PAC) of CRSat relies on the same kind of variable ordering as the one imposed by the depth-based heuristic, and hence provides a theoretical motivation for the heuristic. Furthermore, we compare the behavior of (depth-based) CRSat to that of the justification-based SLS method BC SLS driven by justification frontiers.

Keywords: Boolean circuit satisfiability, stochastic local search, search heuristics, problem structure

1 Introduction

Boolean (or propositional) satisfiability (SAT) [6] solvers provide an efficient approach to solving hard combinatorial problems arising from various real-world domains. Today, the dominant approach to solving industrially relevant application instances is provided by DPLL-based complete algorithms, especially, conflict-driven clause learning (CDCL) [21, 9] solvers. In particular, this has been the case ever since the introduction of CDCL in the mid nineties. Before this, stochastic local search (SLS) methods [18] were considered a competitive alternative for applications of SAT.

At the present, local search is considered to be effective mainly in solving random SAT instances. Despite this fact—and partly due it—the study of novel SLS-based SAT algorithms has its motivations. The state-of-the-art SAT solvers aimed at real-world problem instances today are rather homogenic due to the fact that the solvers typically implement CDCL, and are highly optimized on the implementation level. Escaping from this monoculture requires alternative algorithmic ideas. One potential approach is to develop new hybrid solvers by combining ingredients of CDCL and SLS. Additionally, developing better SLS techniques for structural problem instances can be motivated by the fundamental question of the interplay between problem structure and search algorithms. For improving the applicability of SLS, further work is needed especially for handling variable dependencies, a problem identified as a major challenge in

[⋆] The second author is financially supported by Academy of Finland under grant #132812.

the context of SLS [17]. Compared to CDCL solvers that are implicitly able to focus the search space exploration through tightly bound conflict analysis and decision heuristics, developing structure-exploiting techniques in the context of SLS is challenging due to the intrinsic simplicity of the SLS approach.

One problem in developing efficient techniques for handling variable dependencies is that typically SLS solvers work on the flat CNF input format. Although some structure-based techniques have been developed for conjunctive normal form (CNF) level SLS (e.g., CNF encodings that are more suitable for CNF-level SLS [24]), there is room for SLS techniques that exploit variable dependencies more directly. There are also SAT solvers which—instead of demanding CNF translation before solving—work directly on non-clausal formulas. These solvers often use Boolean circuits as the compact representation for a propositional formula in a DAG-like structure. However, typically such solvers are complete DPLL style non-clausal algorithms, see e.g. [15, 19, 20, 29].

A few SLS methods have been proposed for non-clausal formulas. Common to most of these approaches is to explicitly exploit variable dependencies through independent (or input) variables, i.e., sets of variables such that a truth assignment for them uniquely determines the truth values of all other variables, by focusing the search on truth assignments over input variables [25, 16, 27, 23, 22, 28, 3].

An alternative circuit-level SLS method BC SLS was proposed in [14, 12], based on utilizing justification frontier heuristics (see e.g. [20]) applied in some complete circuit-level SAT solvers in electronic design automation. In BC SLS, the search is driven top-down in the overall structure of the circuit rather than in a bottom-up mode as is done in local search methods focusing on input variables. This is achieved by guiding the search using justification frontiers that enable exploiting observability dont cares and offer an early stopping criterion that allows to end the search when the circuit is de facto satisfiable, even if no concrete satisfying truth assignment has yet been found. Without including additional search techniques not typically present in CNF-level local search methods, the justification frontier based BC SLS was shown to clearly improve performance on real-world SAT instances w.r.t. its CNF-level counterparts.

Most recently, motivated by the general idea of justification-based SLS search applied in BC SLS, another justification-based circuit-level SLS methods, CRSAT, was proposed [4]. Compared to BC SLS, on one hand CRSAT is more simplistic in that the search is not explicitly based on the concept of justification frontier, motivated by the observation that the frontier is rather costly to maintain during search, especially in the context on SLS. On the other hand, CRSAT combines justification-based search with limited circuit-level Boolean constraint propagation, most notably, forward propagation that exploits the explicit circuit structure. It has been shown that variations of CRSAT exhibit greatly improved performance compared to BC SLS.

In this work we study structure-based search heuristics in the context of CRSAT. In particular, with the aim of further improving the performance of CRSAT, we propose a simple *depth-based* search heuristic that exploits the explicit circuit structure. An experimental evaluation reveals that the depth-based heuristic is promising, lifting the performance of the standard CRSAT further on real-world circuit SAT instances arising from industrial-style applications.

The practical improvement brought by the depth-based heuristic can be theoretically justified from a somewhat surprising perspective. Namely, the proof of *probabilistically approximate completeness* (PAC) [11] of CRSAT applies precisely the same style of variable ordering that is imposed by the proposed depth-based heuristic. In addition to this intriguing observation, we also analyze the differences between (depth-based) CRSAT and BC SLS. Combining these observations, we arrive at the hypothesis that restricting justification-based local search to the justification frontier may not be the optimal choice.

The rest of this report is organized as follows. After necessary background on Boolean circuits (Section 2) and the recently proposed justification-based circuit-level SLS methods BC SLS and CRSAT (Section 3), we introduce the proposed depth-based search heuristic for CRSAT, resulting in the variant DEPTH-CRSAT. We then analyze the behavior of DEPTH-CRSAT, comparing it to BC SLS, and also motivate the heuristic by revealing its relationship with the probabilistic approximate completeness of CRSAT (Section 5). Before conclusions, we provide an empirical evaluation of the proposed search heuristic (Section 6).

2 Constrained Boolean Circuits

A *Boolean circuit* over a finite set G of *gates* is a set C of equations of the form $g := f(g_1, \ldots, g_n)$, where $g, g_1, \ldots, g_n \in G$ and $f : \{0,1\}^n \to \{0,1\}$ is a Boolean function, with the additional requirements that (i) each $g \in G$ appears at most once as the left hand side in the equations in C, and (ii) the underlying directed graph

$$\langle G, E(C) = \{\langle g', g\rangle \in G \times G \mid g := f(\ldots, g', \ldots) \in C\}\rangle$$

is acyclic. If $\langle g', g\rangle \in E(C)$, then g' is a *child* of g and g is a *parent* of g'. For a gate g, the sets of children (i.e., the *fanin* of g) and parents (i.e., the *fanout* of g) are denoted by $\mathsf{fanin}(g)$ and $\mathsf{fanout}(g)$, respectively. The *descendant* and *ancestor* relations are defined in the usual way as the transitive closures of the child and parent relations, respectively. If $g := f(g_1, \ldots, g_n)$ is in C, then g is an f-gate (or of type f), otherwise it is an *input gate* (a gate with no children). A gate with no parents is an *output gate*. The sets of input gates and output gates in C are denoted by $\mathsf{inputs}(C)$ and $\mathsf{outputs}(C)$, respectively. A gate that is neither an input nor an output gate is an *internal gate*.

An *(truth) assignment* for C is a (possibly partial) function $\tau : G \to \{0,1\}$. A complete assignment τ for C is *consistent* if $\tau(g) = f(\tau(g_1), \ldots, \tau(g_n))$ for each $g := f(g_1, \ldots, g_n)$ in C. A circuit C has $2^{|\mathsf{inputs}(C)|}$ consistent complete assignments. The *domain* of τ, i.e., the set of gates assigned in τ, is denoted by $\mathit{dom}(\tau)$. We say that two assignments, τ and τ', *disagree* on a gate $g \in \mathit{dom}(\tau) \cap \mathit{dom}(\tau')$ if $\tau(g) \neq \tau'(g)$.

A *constrained Boolean circuit* C^α is a pair $\langle C, \alpha\rangle$, where C is a Boolean circuit and α is an assignment for C. With respect to a constrained circuit C^α, each $\langle g, v\rangle \in \alpha$ is a *constraint*, and g is *constrained* to v if $\langle g, v\rangle \in \alpha$. A complete assignment τ for C *satisfies* C^α if (i) τ is consistent with C, and (ii) it respects the constraints: $\tau \supseteq \alpha$. If some complete assignment satisfies C^α, then C^α is *satisfiable* and otherwise *unsatisfiable*. In this work we consider Boolean circuits in which the following Boolean functions are available as gate types; these circuits are often referred to as *And-Inverter Graphs* (AIGs).

– $\textsc{not}(v)$ is 1 iff v is 0.
– $\textsc{and}(v_1, v_2)$ is 1 iff both v_1 and v_2 are 1.

However, notice that the techniques discussed in this paper can be adapted for a wider range of types. In order to keep the presentation and algorithms simpler, we assume that constraints only appear in the output gates of constrained circuits. Any circuit can be rewritten into such a normal form [15].

The *restriction* of an assignment τ to a set $G' \subseteq G$ of gates is defined as usual: $\tau|_{G'} = \{\langle g, v\rangle \in \tau \mid g \in G'\}$. Given a non-input gate $g := f(g_1, \ldots, g_n)$ and a value $v \in \{0, 1\}$, a *justification* for the pair $\langle g, v\rangle$ is a partial assignment $\sigma : \{g_1, \ldots, g_n\} \to \{0, 1\}$ to the children of g such that $f(\tau(g_1), \ldots, \tau(g_n)) = v$ holds for all extensions $\tau \supseteq \sigma$. That is, the values assigned by σ to the children of g are enough to force g to take the consistent value v. A gate g is *justified in an assignment* τ if it is assigned, i.e. $\tau(g)$ is defined, and (i) it is an input gate, or (ii) $g := f(g_1, \ldots, g_n) \in C$ and $\tau|_{\{g_1,\ldots,g_n\}}$ is a justification for $\langle g, \tau(g)\rangle$. We denote the set of *unjustified* gates in an assignment τ by $\mathsf{unjust}(C^\alpha, \tau)$.

The *justification cone* $\mathsf{jcone}(C^\alpha, \tau)$ for a constrained circuit C^α under an assignment $\tau \supseteq \alpha$ is defined top-down in the circuit structure, starting from the constrained gates. Intuitively, the cone is the smallest set of gates which includes all constrained gate and, for each justified gate in the set, all the gates that participate in some subset-minimal justification for the gate. More formally, $\mathsf{jcone}(C^\alpha, \tau)$ is the smallest of the sets S of gates which satisfy the following properties.

1. If $\langle g, v\rangle \in \alpha$, then $g \in S$.
2. If $g \in S$ and (i) g is a non-input gate, (ii) g is justified in τ, and (iii) $\langle g_i, v_i\rangle \in \sigma$ for some subset minimal justification σ for $\langle g, \tau(g)\rangle$, then $g_i \in S$.

Notice that by this definition $\mathsf{jcone}(C^\alpha, \tau)$ is unambiguously defined.

As another key concept, the *justification frontier* of C^α under τ, is the "bottom edge" of the justification cone, i.e. those gates in the cone that are not justified:

$$\mathsf{jfront}(C^\alpha, \tau) = \mathsf{jcone}(C^\alpha, \tau) \cap \mathsf{unjust}(C^\alpha, \tau).$$

Notice that we always have $\mathsf{jfront}(C^\alpha, \tau) \subseteq \mathsf{unjust}(C^\alpha, \tau)$, and also that $\mathsf{jfront}(C^\alpha, \tau)$ is unambiguously defined.

A gate g is *interesting* in τ if it belongs to the frontier $\mathsf{jfront}(C^\alpha, \tau)$ or is a descendant of a gate in it; the set of all gates that are interesting in τ is called the *interest set* under τ, and denoted by $\mathsf{interest}(C^\alpha, \tau)$. A gate g is an (*observability*) *don't care* if it is neither interesting nor in the justification cone $\mathsf{jcone}(C^\alpha, \tau)$.

As observed in [14] if the justification frontier $\mathsf{jfront}(C^\alpha, \tau)$ is empty for some complete assignment τ, then the constrained circuit C^α is satisfiable. When $\mathsf{jfront}(C^\alpha, \tau)$ is empty, a satisfying assignment is obtained by (i) restricting τ to the input gates appearing in the justification cone, i.e. to the gate set $\mathsf{jcone}(C^\alpha, \tau) \cap \mathsf{inputs}(C)$, (ii) assigning other input gates (don't cares) arbitrary values, and (iii) recursively evaluating the values of non-input gates. Thus, whenever $\mathsf{jfront}(C^\alpha, \tau)$ is empty, we say that τ *de facto satisfies* C^α.

Finally, given a truth assignment τ and a set of gates $G \subseteq dom(\tau)$, we define

$$\textit{flip}(G,\tau) = \left(\tau \setminus \bigcup_{g\in G} \{\langle g, \tau(g)\rangle\}\right) \cup \bigcup_{g\in G} \{\langle g, 1-\tau(g)\rangle\}.$$

In other words, *flip*(G,τ) is the truth assignment obtained by changing the truth-values of the gates in G, and leaving the rest of the truth-values unchanged.

3 Justification-Based Circuit-Level SLS

In this section we give an overview of the justification-based circuit-level SLS methods BC SLS and CRSAT. Our goal in this paper is to develop structure-based search heuristics for the latter, and to analyze the behavior of the heuristics w.r.t. the behavior of the former method.

3.1 BC SLS – SLS Driven by Justification Frontiers

The first SLS algorithm for constrained Boolean circuits that uses the internal gates to drive the search was proposed by Järvisalo, Junttila and Niemelä in [14], and later improved and analyzed theoretically in [12]. Pseudocode for this algorithm, called BC SLS, is presented as Algorithm 1.

The algorithm starts with a random complete truth assignment τ for C^α, and, as long as the justification frontier $\mathsf{jfront}(C^\alpha,\tau)$ is not empty, picks a random gate g from $\mathsf{jfront}(C^\alpha,\tau)$ (line 6), and proceeds by making one of the following two types of moves.

– In the *downward* move the selected unjustified gate g is justified by flipping some of its children. To achieve this, the algorithm constructs the set Σ of justifications for g and $\tau(g)$, and either picks one justification randomly (this is the *non-greedy downward move* – line 12), or selects a justification that minimizes the size of the interest set after the flip (this is the *greedy downward move* – line 11). As a result of this move, the gate g becomes justified, while some gates in $\mathsf{fanin}(g)$ potentially become unjustified.
– In the *upward* move the truth-value of the selected unjustified gate g itself is flipped (line 14), if possible (if g is constrained then the algorithm resorts to a non-greedy downward move – line 12), thus making it justified, and, potentially, causing one or more gates in $\mathsf{fanout}(g)$ to become unjustified. The name "upward" is used because it (often) pushes the unjustified gate(s) towards the outputs of the circuit.

The results of the empirical evaluation of BC SLS presented in [14, 12] demonstrate that the algorithm is significantly more effective on industrial circuits than SLS-based CNF-level algorithms. This indicates that circuit structure can be used by SLS-based SAT algorithms to significantly accelerate the search for satisfying assignments.

As elaborated in [2], the dynamic behavior of BC SLS is a balance between driving the justification frontier towards the input gates using the downward moves (as eventually the algorithm must flip some of the inputs) and taking recovery steps to fix

Algorithm 1 BC SLS(C^α, p, q, *cutoff*)

```
Input:  $C^\alpha$ – constrained Boolean circuit
        $p$ – the probability of non-greedy move
        $q$ – the probability of non-greedy downward move
        cutoff – the cutoff, that is, the maximum number of steps
Output: $status$ – SAT if a satisfying assignment for $C^\alpha$ is found, UNKNOWN otherwise
        $\tau$ – a de-facto satisfying assignment for $C^\alpha$ if found, $\emptyset$ otherwise
1:  $\tau \leftarrow$ a random truth assignment for $C^\alpha$
2:  steps $\leftarrow 0$
3:  while steps $<$ cutoff do
4:      if jfront$(C^\alpha, \tau) = \emptyset$ then
5:          return $\langle \mathsf{SAT}, \tau \rangle$
6:      $g \leftarrow$ a random element from jfront$(C^\alpha, \tau)$
7:      $\Sigma \leftarrow$ the set of justifications for $\langle g, \tau(g) \rangle$
8:      with-probability $1 - p$ do
9:          $\sigma \leftarrow$ a random justification from the justifications in $\Sigma$      ▷ greedy downward move
                that minimize $|\mathsf{interest}(C^\alpha, \cdot)|$
10:     otherwise
11:         if $g \in dom(\alpha)$ or with probability $q$ do
12:             $\sigma \leftarrow$ random justification from $\Sigma$      ▷ non-greedy downward move
13:         else
14:             $\sigma \leftarrow \{\langle g, 1 - \tau(g) \rangle\}$      ▷ upward move
15:     end with-probability
16:     $G \leftarrow$ set of gates in $\sigma$ that disagree with $\tau$
17:     $\tau \leftarrow flip(G, \tau)$      ▷ flip
18:     steps $\leftarrow$ steps $+ 1$
19: return $\langle \mathsf{UNKNOWN}, \emptyset \rangle$
```

the incorrectly assigned gates in the justification cone (the upwards moves constitute such recovery steps). During this process, there are no hints as to whether a mistake has actually occurred or not: when the justification frontier is "deep" inside the circuit, BC SLS relies on the truth-values in the justification cone to make progress. If these truth-values are set incorrectly, the chances to fix them are small, and get progressively smaller as the justification frontier approaches the inputs. As a result, recovery from mistakes requires potentially a very large number of steps. The algorithm described in the following section attempts to address this deficiency of BC SLS.

3.2 CRSAT – Justification-Based SLS with Limited Forward Propagation

In this section we briefly describe the circuit-level SLS method CRSAT proposed in [4]. The algorithm uses the idea of justification-based steps through the space of complete truth assignments borrowed from BC SLS. However, it does not maintain the justification frontier, and does not perform explicit upward moves – instead, CRSAT uses *limited forward propagation* to control the search.

Limited forward propagation is a restricted form of Boolean Constraint Propagation (BCP) applied on the level of circuits. Pseudocode for the procedure is presented as

Algorithm 2 LBCP-FORWARD(C^α, G, τ)

```
Input: $C^\alpha$ – constrained Boolean circuit;
       $G$ – a set of gates whose truth-value change are to be propagated.
       $\tau$ – a truth assignment for $C^\alpha$;
Output: $\tau'$ – an assignment for $C^\alpha$ which is a result of limited forward propagation of the
    assignment $\tau|_G$.
 1: $\tau' \leftarrow \tau$
 2: $Q$.ENQUEUE($G$)
 3: while $\neg$ $Q$.EMPTY do
 4:     $g \leftarrow Q$.POP_FRONT
 5:     if $g \in G$ then                                          ▷ $g$ is one of the original gates
 6:         $Q$.ENQUEUE(fanout($g$))
 7:     else
 8:         if $g \notin dom(\alpha)$ and $\neg justified(\tau', g)$ then    ▷ $g$ unconstrained and unjustifed
 9:             $\tau' \leftarrow flip(\{g\}, \tau')$
10:             $Q$.ENQUEUE(fanout($g$))
11: return $\tau'$
```

Algorithm 2. The algorithm makes use of a data structure Q which is a priority queue of gates that uses a fixed topological ordering[3] $<_t$ of the set of gates and allows to query the *smallest* gate in this ordering (i.e., "fanin-first") in constant time. The priority queue is backed up by a set, so that duplicate entries are not allowed. More details on this data-structure are given in [2].

The algorithm operates as follows: given a set of gates G (that, presumably, have just changed their truth-value as a result of a previously performed flip), the algorithm starts by enqueuing the gates from G. Then, as long as the queue is not empty, the smallest, according to the topological order $<_t$ gate g is removed from the queue, and

– if $g \in G$, then all gates in $\mathsf{fanout}(g)$ are enqueued, and the algorithm continues,
– if $g \notin G$, then the algorithm checks whether or not g's truth-value is consistent with that of its children. If it is, no action is taken; if it is not, and g is not constrained, the algorithm flips the truth-value of g (so now it is consistent with the truth-values of the gates in $\mathsf{fanin}(g)$), and enqueues all gates in $\mathsf{fanout}(g)$.

The key property of the propagation procedure in the context of justification based search is captured by the following proposition

Proposition 1 ([2], Propositions 5.33 and 5.34). *Let C^α be a constrained Boolean circuit, G a set of gates in C, and τ a truth assignment for C^α. Further, let $\tau' =$ LBCP-FORWARD(C^α, G, τ). Then, the following conditions hold.*

(i) For any gate $g \notin G$, if g is justified in τ and is not justified in τ', then $g \in dom(\alpha)$.
(ii) For any gate $g \in \mathsf{unjust}(C^\alpha, \tau') \setminus G$, we have $\tau'(g) = \tau(g)$.

[3] A topological ordering $<_t$ of the gates in a Boolean circuit is any strict total order that respects the condition "if $g_2 \in \mathsf{fanout}(g_1)$, then $g_1 <_t g_2$".

Thus limited forward propagation does not create new unjustified unconstrained gates, and preserves the truth-values of unconstrained gates that remain unjustified ofter forward propagation.

Pseudocode for CRSAT is presented as Algorithm 3. The algorithm starts by constructing a complete consistent extension of a random truth-value assignment to $\mathsf{inputs}(C^{\alpha})$ — that is, the truth-value of each unconstrained internal gate is set consistently with the truth-values of its children. Then, as long as $\mathsf{unjust}(C^{\alpha}, \tau)$ is not empty (that is, τ is not a satisfying assignment), the algorithm selects a random gate g from $\mathsf{unjust}(C^{\alpha}, \tau)$ (line 6), constructs the set of justifications Σ for $\langle g, \tau(g)\rangle$, and performs one of the following types of moves:

– During the *random walk* a random justification $\sigma \in \Sigma$ is selected, the truth-values of the set G of gates assigned in σ that disagree with τ are flipped, and the effects of the flips are propagated using the procedure for limited forward propagation LBCP-FORWARD, discussed earlier. We denote this combination of flip with limited forward propagation as a *step* of CRSAT from now on. The implementation of the step is encapsulated in the function STEP(C^{α}, G, τ) (lines 17-20) that returns the resulting complete assignment. The random walk is performed with the "walk-probability" wp.
– During the *greedy move* the algorithm selects from the set Σ of justifications for $\langle g, \tau(g)\rangle$ one of the justifications that minimizes the size of $\mathsf{unjust}(C^{\alpha}, \tau)$ after the step (that is after the flip, followed by forward propagation).

It is easy to see that on one hand CRSAT is significantly simpler than BC SLS – it uses the set of all unjustified gates, rather than the justification frontier, to perform steps and to make heuristic decisions. It does not make upward moves, and so has less control parameters. On the other hand, the steps of CRSAT are *crucially* different from those of BC SLS in that every flip is followed by limited forward propagation.

In [2] it is shown that as the result of these modifications, CRSAT is orders of magnitude more efficient than BC SLS on a wide variety of industrial circuits. The absence of upward moves in CRSAT is also justified, both theoretically and empirically.

4 Depth-Based Heuristics for CRSAT

We propose a simple depth-based heuristic for CRSAT, resulting in what we call DEPTH-CRSAT. For introducing the heuristic and for discussing the relationship between DEPTH-CRSAT and BC SLS, we first introduce some additional concepts.

Definition 1. *Let C^{α} be a Boolean circuit. A* path *π in C^{α} is any non-empty sequence of gates $\langle g_1, \ldots, g_n\rangle$, such that for every $1 < i \leq n$, $g_i \in \mathsf{fanin}(g_{i-1})$.*

Note that a path may not be empty but may contain only one gate.

Definition 2. *Let C^{α} be a constrained Boolean circuit, τ a truth assignment for C^{α}, and g a gate in $\mathsf{unjust}(C^{\alpha}, \tau)$. Let $\pi = \langle g_1, \ldots, g_k\rangle$ be a path in C^{α} such that g_1 is constrained, and $g_k = g$. Then, π is a* justification path *for g in C^{α} under τ if*

Algorithm 3 CRSAT(C^α, wp, *cutoff*)

```
Input: $C^\alpha$ – constrained Boolean circuit
       $wp$ – the noise parameter, that is, the probability of random walk
       cutoff – the cutoff, that is, the maximum number of steps
Output: $status$ – SAT if a satisfying assignment for $C^\alpha$ is found, UNKNOWN otherwise
        $\tau$ – a satisfying assignment for $C^\alpha$ if found, $\emptyset$ otherwise
1: $\tau \leftarrow$ a complete consistent extension of a random truth assignment to inputs($C^\alpha$)
2: steps $\leftarrow 0$
3: while steps $<$ cutoff do
4:     if unjust($C^\alpha, \tau$) $= \emptyset$ then
5:         return $\langle$SAT$, \tau\rangle$
6:     $g \leftarrow$ a random element from unjust($C^\alpha, \tau$)
7:     $\Sigma \leftarrow$ the set of justifications for $\langle g, \tau(g)\rangle$
8:     with-probability $wp$ do
9:         $\sigma \leftarrow$ random element of $\Sigma$                              ▷ random walk
10:    otherwise
11:        $\sigma \leftarrow$ a random justification from the justifications in $\Sigma$   ▷ greedy downward move
               that minimize |unjust($C^\alpha, \cdot$)| after step
12:    end with-probability
13:    $G \leftarrow$ set of gates in $\sigma$ that disagree with $\tau$
14:    $\tau \leftarrow$ STEP($C^\alpha, G, \tau$)                       ▷ flip + limited forward propagation
15:    steps $\leftarrow$ steps $+ 1$
16: return $\langle$UNKNOWN$, \emptyset\rangle$

17: function STEP($C^\alpha, G, \tau$)                        ▷ Flips gates in $G$ and propagates forward
18:     $\tau' \leftarrow$ flip($G, \tau$)
19:     $\tau' \leftarrow$ LBCP-FORWARD($C^\alpha, G, \tau'$)
20:     return $\tau'$
```

(i) for every $i < k$, g_i is justified under τ; and
(ii) for every $i < k$, $\langle g_{i+1}, \tau(g_{i+1})\rangle$ is in some subset-minimal justification for $\langle g_i, \tau(g_i)\rangle$.

In other words, $\pi = \langle g_1, \ldots, g_k\rangle$ is a justification path for $g_k \in \mathsf{unjust}(C^\alpha, \tau)$ if for every $i < k$ we have $g_i \in \mathsf{jcone}(C^\alpha, \tau) \setminus \mathsf{unjust}(C^\alpha, \tau)$.

Proposition 2. *Let C^α be a constrained Boolean circuit, τ a truth assignment for C^α, and g a gate in $\mathsf{unjust}(C^\alpha, \tau)$. Then, $g \in \mathsf{jfront}(C^\alpha, \tau)$ if and only if there exists a justification path for g in C^α under τ.*

Proof. Follows from the definition of $\mathsf{jfront}(C^\alpha, \tau)$.

Basically, a justification path for a gate g is a witness of (reason for) the fact that $g \in \mathsf{jfront}(C^\alpha, \tau)$.

Consider now some execution of CRSAT, and assume that the gate $g \in \mathsf{unjust}(C^\alpha, \tau)$ selected on line 6 of the algorithm is such that $g \in \mathsf{jfront}(C^\alpha, \tau)$. Let σ be the justification for $\langle g, \tau(g)\rangle$ selected by the algorithm (either greedily or randomly) by the end of line 12, and G be the set of gates in σ that disagree with τ. Let us consider the effects of the step of the algorithm performed on line 14. Since $g \in \mathsf{jfront}(C^\alpha, \tau)$ by assumption, by Proposition 2 there is a justification path for g under τ. After the step, g becomes justified, and one or more gates in G possibly become unjustified – let us denote the subset of gates in G that become unjustified after the step by G'. Notice that since each gates $g' \in G'$ belongs to some subset minimal justification for g, the justification path for g can now be extended to g'. There is a possibility that g' is part of the *unique* justification path for some other gate in $\mathsf{jfront}(C^\alpha, \tau)$, and hence when g' becomes unjustified it will "cut off" this path. This situation is *not* possible, however, if the gate g selected on line line 6 of the algorithm is taken to be *a gate with the maximum depth among the gates in* $\mathsf{jfront}(C^\alpha, \tau)$, that is, from the set

$$\operatorname*{argmax}_{g \in \mathsf{jfront}(C^\alpha, \tau)} \mathsf{depth}(C^\alpha, g),$$

where

$$\mathsf{depth}(C^\alpha, g) = \begin{cases} 0 & \text{if } g \in \mathsf{outputs}(C) \\ 1 + \max\{\mathsf{depth}(C^\alpha, g') \mid g' \in \mathsf{fanout}(g)\} & \text{otherwise.} \end{cases} \tag{1}$$

With this intuition, combined with the fact that CRSAT focuses on currently unjustified gates (not only gate in the current justification frontier), we denote by DEPTH-CRSAT the version of CRSAT in which the unjustified gate g on line 6 is selected at random from the *current set* $\mathsf{unjust}(C^\alpha, \tau)$ *of unjustified gates at maximum depth according to* (1).

5 Insights into DEPTH-CRSAT

BC SLS uses the justification frontier and searches for *de facto* satisfying assignments, while in CRSAT the search is focused on the set of *all* unjustified gates. Related to this fact, in this section we analyze differences in the behavior of BC SLS and DEPTH-CRSAT. Namely, we show that even by focusing search on unjustified gates with maximum depth as is done in DEPTH-CRSAT, there are examples of configurations under which DEPTH-CRSAT may choose to justify a gate that is not in the current justification frontier. In other words, in general DEPTH-CRSAT does not implicitly maintain the justification frontier during search, something which is done explicitly in BC SLS (with additional cost).

Additionally, and somewhat surprisingly, a proof of CRSAT being probabilistically approximately complete (PAC) relies very much on focusing search on unjustified gates with maximum depth. This fact directly implies that even the more restricted DEPTH-CRSAT is PAC, and hence provides an additional motivation for applying the depth-based heuristics in practice due to more focused search without losing PAC.

5.1 Justification Frontiers and DEPTH-CRSAT

An apparent difference between the methods is that BC SLS maintains an explicit justification frontier during search and focuses search on gates in the frontier, CRSAT relaxes the search to consider any unjustified gates (given that there are additional unjustified gates that are not part of the current justification frontier).

This poses the question of the relationship between the set of unjustified gates (in CRSAT) and the justification frontier (in BC SLS). We now look at this question from the perspective of DEPTH-CRSAT. Especially, consider the following observations.

– Initially, we have $\mathsf{unjust}(C^\alpha, \tau) = \mathsf{jfront}(C^\alpha, \tau)$, since τ is a consistent extension of some truth assignment to $\mathsf{inputs}(C^\alpha)$.
– By focusing search on unjustified gates with maximum depth, as in DEPTH-CRSAT and as discussed in Section 4, justification paths for the gates in $\mathsf{jfront}(C^\alpha, \tau)$ are not cut off.
– By Proposition 1, limited forward propagation does not create new unjustified unconstrained gates.

Taking these observations into account, it appears that there is no danger of "cutting off" existing justification paths during the BCP. Hence, We formalize the following conjecture.

Conjecture 1 *Let C^α be a constrained Boolean circuit, and τ_k be a truth assignment at the end of the kth step of* DEPTH-CRSAT *on C^α. Then, for any $k \geq 0$, it holds that* $\mathsf{unjust}(C^\alpha, \tau_k) = \mathsf{jfront}(C^\alpha, \tau_k)$.

However, *this conjecture turns out to be false*, as demonstrated in the following.

Example 1. Consider the circuit C^α depicted in Figure 1, with $\alpha = \langle g_1, 0 \rangle$. The dashed lines represent edges that have a NOT gate attached to them. Assume that the initial truth assignment, which we denote by τ_1, is a complete consistent extension of the assignment $\{\langle i_1, 1 \rangle, \langle i_2, 1 \rangle, \langle i_3, 1 \rangle\}$ – this assignment is presented in Figure 1(a). The letter "u" (letter "j", respectively) denotes the fact that a gate is unjustified (justified, respectively). We follow the execution of DEPTH-CRSAT on C^α.

1. Figure 1(a): $\mathsf{unjust}(C^\alpha, \tau_1) = \{g_1\}$, and so the gate selected by the algorithm on line 6 is g_1. Assume that the algorithm performs a random walk step and selects the justification $\{\langle g_2, 1 \rangle\}$ for $\langle g_1, 0 \rangle$. The algorithm flips the value of g_2. Then forward propagation does not change any assigned values. The resulting assignment, τ_2, is depicted in Figure 1(b).
2. Figure 1(b): $\mathsf{unjust}(C^\alpha, \tau_2) = \{g_2\}$. Since $\{\langle g_4, 1 \rangle\}$ is the only justification for $\langle g_2, 1 \rangle$, the algorithm flips the value of g_4. Again, forward propagation does not change any assigned values. The assignment τ_3, resulting from flipping g_4, is depicted in Figure 1(c).
3. Figure 1(c): $\mathsf{unjust}(C^\alpha, \tau_3) = \{g_4\}$. Again, $\{\langle g_5, 1 \rangle\}$ is the only justification for $\langle g_4, 1 \rangle$, and the algorithm flips the value of g_5. This time, forward propagation assigns $\langle g_3, 0 \rangle$ and $\langle g_2, 0 \rangle$. Let us denote the resulting assignment, depicted in Figure 1(d), by τ_4.

Note that under τ_4, both of the gates g_1 and g_5 are unjustified, and thus $\mathsf{unjust}(C^\alpha, \tau_4) = \{g_1, g_5\}$. However, $\mathsf{jfront}(C^\alpha, \tau_4) = \{g_1\}$ – that is, the justification path for the gate g_5 got "cut off" by the gate g_1.

Thus, even though the forward propagation does not create new unjustified unconstrained gates, the possibility of it creating new unjustified constrained gates may result in some justification paths being "cut off", which means that Conjecture 1 fails. ■

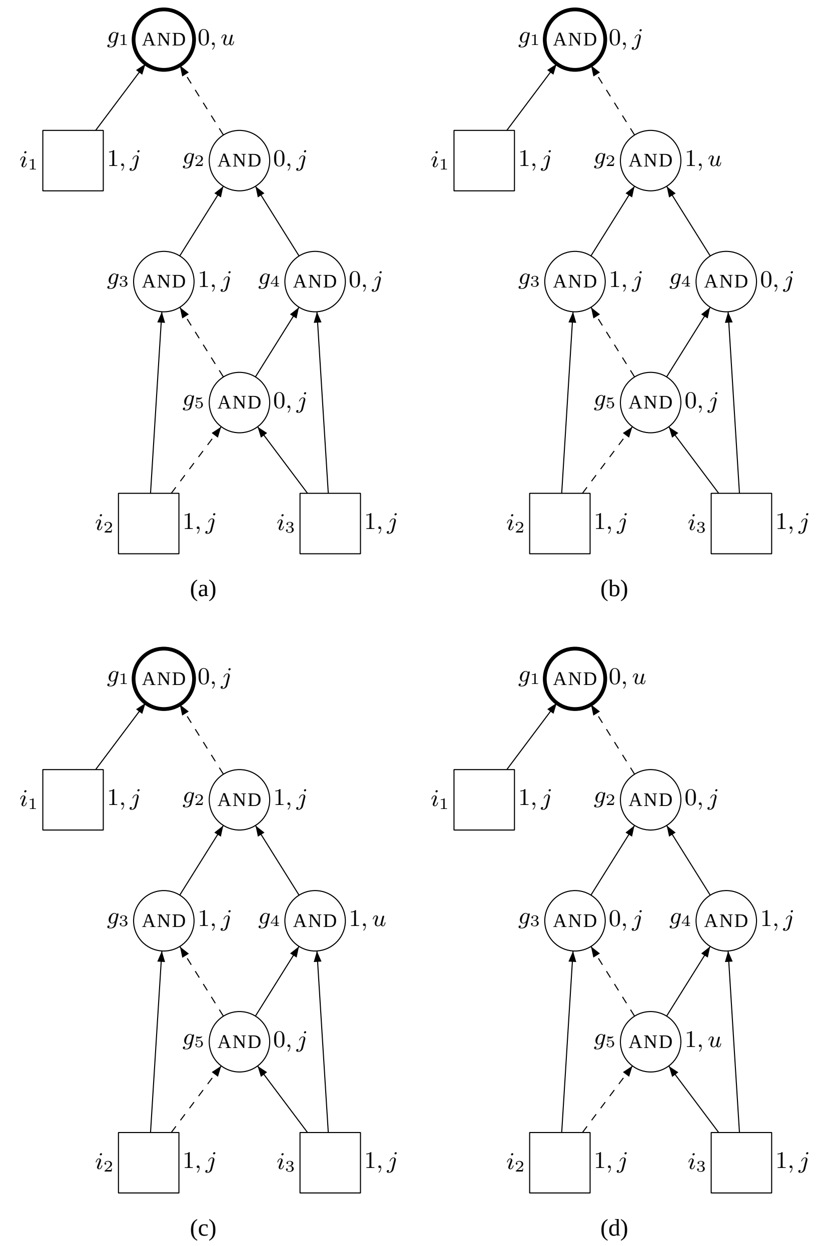

Fig. 1. A counterexample for Conjecture 1: (a) – initial situation; (b) – after the flip of g_2; (c) – after the flip of g_4; (d) – after the flip of g_5 and forward propagation.

Hence we have the following.

Proposition 3. *There are cases in which* DEPTH-CRSAT *may select to justify a gate that is not in the current justification frontier.*

In other words, we always have $\mathsf{unjust}(C^\alpha, \tau_k) \supseteq \mathsf{jfront}(C^\alpha, \tau_k)$, and even for DEPTH-CRSAT there are cases in which $\mathsf{unjust}(C^\alpha, \tau_k) \supset \mathsf{jfront}(C^\alpha, \tau_k)$. On the other hand, it is not clear that restricting search on gates in the justification frontier, as done in the BC SLS method, is always a good idea *in the presence of forward propagation*[4]. In fact, some of the empirical data presented in [2] suggests that this may not be the case.

It should also be noted that, from a practical perspective, maintaining the justification frontier, even incrementally, is in the worst-case rather costly. Indeed, experimental results in [2, 4] imply that CRSAT exhibits dominating performance w.r.t. BC SLS. In the experimental evaluation presented in this paper, we will show that the depth-based heuristic of DEPTH-CRSAT improves the performance of CRSAT further. While, as we demonstrated above, the power of depth-based heuristic in the context of CRSAT does not seem to be related to the justification frontier, it can be explained by the analysis of the PAC property of CRSAT, presented in the following section.

5.2 DEPTH-CRSAT and the PAC property of CRSAT

The notion of *Probabilistic Approximate Completeness (PAC)* of SLS algorithms was introduced by Hoos in [11]. Stated informally, an SLS algorithm is PAC if, for any satisfiable problem instance, the probability of finding a solution is asymptotically 1.

Definition 3 (PAC; adapted from [11]). *A SAT algorithm* A *is* probabilistically approximately complete (PAC) *if for any satisfiable instance F,*

$$\lim_{t\to\infty} P(Runtime_{\mathrm{A},F} \leq t) = 1, \tag{2}$$

where $P(Runtime_{\mathrm{A},F} \leq t)$ denotes the probability that A *finds a satisfying assignment for F in time $\leq t$. Further,* A *is* essentially incomplete *if it is not PAC, that is, there exists a satisfiable instance F such that*

$$\lim_{t\to\infty} P(Runtime_{\mathrm{A},F} \leq t) < 1. \tag{3}$$

Typically, the PAC property of an SLS algorithm is proven by showing that from any given non-satisfying assignment τ there is a non-zero probability p that the algorithm arrives at some fixed satisfying assignment τ^* within a fixed number of steps. Note that when each step takes finite time, we can associate steps with time in this analysis. Then, the sequence of the first n steps taken by the algorithm can be split into fragments of k steps each, and, when no assumptions are made with regard to the initial assignment τ,

[4] As an intriguing complementary observation, we note that the efficiency of DPLL and CDCL has been theoretically shown to notably decrease on *unsatisfiable* formulas in case decision heuristics are restricted using justification frontiers [13].

each fragment can be treated independently. Then, the PAC property follows from the analysis of the resulting Markov chain.

Its easy to see that all CNF-level SLS methods that allow random walk during *any step* are PAC [11] – random walk allows the algorithm to "cheat" the heuristic by giving a non-zero probability to the event that the algorithm will make an improving step by flipping the "correct" variable in a randomly chosen unsatisfied clause (any unsatisfied clause has such a variable). Similar argument, generalized to the setting of constrained Boolean circuits, was used in [12] to show the PAC property of BC SLS. In the case of CRSAT, this approach to the proof of PAC property does not work due to the fact that the deterministic forward propagation makes non-local changes to the current truth assignment. Hence, there is a possibility that by setting the truth-value of one gate correctly, the truth-values of other gates may become incorrect. Nevertheless, CRSAT, is indeed PAC.

Theorem 2 ([2], Theorem 5.48). *The* CRSAT *algorithm with any noise parameter value $wp > 0$ and infinite cutoff is probabilistically approximately complete.*

The proof of Theorem 2 relies on the following two propositions. For detailed proofs we refer the reader to [2]. For the following, given a constrained Boolean circuit C^α, a truth assignment τ is called a *restart assignment* if $\mathsf{unjust}(C^\alpha, \tau) \subseteq \mathit{dom}(\alpha)$.

Lemma 1 ([2], Theorem 5.46). *Starting from any restart assignment, there is a non-zero probability that within a bounded number of steps,* CRSAT *will set all input gates correctly, thus arriving at a satisfying assignment.*

Proof sketch. If on line 6 CRSAT always selects an unjustified gate with maximum depth (note that this event has always a non-zero probability), then, with non-zero probability, at least one input gate will be set correctly after at most $\mathsf{depth}(C^\alpha) = \max_g \mathsf{depth}(C^\alpha, g)$ steps, where g ranges over all gates in C. □

Lemma 2 ([2], Theorem 5.47). *From any truth assignment, there is a non-zero probability that within a bounded number of steps* CRSAT *will reach a restart assignment.*

Proof sketch. If CRSAT selects at each step an unjustified gate with maximum depth on line 6, then after a bounded number of steps all unconstrained unjustified gates will become justified. When this is the case, the truth assignment is the restart assignment. □

Proof sketch (of Theorem 2). Consider any execution of CRSAT on a circuit C^α, let τ_i be a truth assignment on step i of this execution, and let X_i be a random variable defined as follows:

$$X_i = \begin{cases} 0 & \text{if } \tau_i \text{ is a non-satisfying restart assignment} \\ 1 & \text{if } \tau_i \text{ is a non-satisfying non-restart assignment} \\ 2 & \text{if } \tau_i \text{ is a satisfying assignment.} \end{cases}$$

Let k_1 and k_2 be the bounds in Lemmas 1 and 2, respectively, and consider the sequence of random variables

$$\langle X_0, X_{k_1}, X_{k_1+k_2}, X_{2k_1+k_2}, X_{2k_1+2k_2}, \ldots \rangle.$$

It is not difficult to see that this is a Markov chain. Furthermore, using Lemmas 1 and 2, one can show that the state 2 is the only persistent state. □

It is clear that depth plays an essential role in establishing the PAC property of CRSAT. Notably, the same reasoning establishes the PAC of DEPTH-CRSAT. Furthermore, since DEPTH-CRSAT selects a deepest unjustified gate with the probability 1, the bounds on the number of steps used in Lemmas 1 and 2 are significantly smaller. Hence, in theory, DEPTH-CRSAT converges to a satisfying assignment significantly faster. The results of the empirical evaluation of DEPTH-CRSAT presented in the next section confirm that this is also the case in practice.

6 Experiments

To evaluate the effectiveness of the depth-based heuristic for CRSAT presented in this paper we compare empirically the performance of CRSAT with that of DEPTH-CRSAT. As also suggested in [14] in the context of BC SLS, in our implementations of CRSAT and DEPTH-CRSAT justifications used for flipping truth-values are selected from the set of subset minimal justifications for the gate value; in other words, for a false AND-gate the value of a single child is flipped, and for a true AND-gate the values of all its children flipped. This strategy gave positive results in preliminary experiments. Additionally, we note that both of the implementations apply circuit-level Boolean constraint propagation as a preprocessing step before search.

We follow the methodology for comparative performance analysis of SLS-based SAT algorithms described in [10]. According to this methodology, the comparative analysis is performed by obtaining the empirical run-time and run-length distributions (RTDs/RLDs) of the solvers that implement the algorithms on a variety of benchmark circuits. In our experiments, the RTDs/RLDs were obtained from 100 runs on each benchmark circuit. The near-optimal setting of the walk probability parameter wp was determined in the preliminary set of experiments. Since in this paper we are concerned with large-scale effects of the heuristics, we only compare the medians of the obtained distributions. The medians are compared by employing the Mann-Whitney U-test (see [26] for example). Here we consider a difference between the median of the run-times/run-length of two solvers *statistically significant* when the U-test rejects the null hypothesis at the p-value of at most 0.05. All experiments were performed under Linux (kernel version 2.6.9) using a Intel Core 2 Duo 3.00-GHz processor and 4 GB of RAM.

Before presenting the results, we shortly describe the benchmarks used in the experiments.

6.1 Benchmarks

The evaluation of the effectiveness of the depth-based heuristic in CRSAT was performed on over 400 benchmark circuits from four different benchmark classes. These circuits come directly from various industrial applications of SAT.

hwmcc08-sat This is the set of 204 satisfiable benchmarks representing BMC problems (with step bound $k = 45$) obtained from all the sequential Hardware Model Checking Competition HWMCC 2008 problems. The `aigbmc` tool [5] was used for the time frame expansion.

smtqfbv-sat These AIG circuits were obtained by bit-blasting instances from the `QF_BV` category (theory of bit-vectors) used in the SMT competition 2009 [1]. For the bit-blasting we used the `Boolector` SMT solver [8]. We used 61 satisfiable benchmarks from this set for the experiments.

sss-sat-1.0 These circuits by Velev originate from "formal verification of buggy variants of a dual-issue superscalar microprocessor with exceptions, multicycle functional units, and branch prediction" [30]. The benchmarks were originally in the `ISCAS` format, and we converted them to AIGs using `ABC` [7]. We used 96 of these circuits in our experiments.

vliw-sat-1.1 This is another suite of circuits by Velev originating from "formal verification of buggy variants of a VLIW microprocessor" [30]. As with the previous set, the benchmarks were converted from the `ISCAS` format to AIGs using `ABC`. We used 98 of these circuits in our experiments.

6.2 Results

To evaluate the effectiveness of the depth-based selection of unjustified gates empirically, we implemented two SAT solvers: the SAT solver `crsat` that implements the CRSAT algorithm described in Algorithm 3, and the SAT solver `crsat-d` that implements the depth-based variant DEPTH-CRSAT.

In order to allow for the retrieval of the deepest unjustified gate in `crsat-d`, the set $\mathsf{unjust}(C^\alpha, \tau)$ of unjustified gates is kept in a heap datastructure. The datastructure allows to retrieve the deepest unjustified gate in constant time. However, there is a performance penalty for insertions of the order of $\log(|\mathsf{unjust}(C^\alpha, \tau)|)$.

Table 1 summarizes the results of the empirical comparison between the performance of `crsat` and `crsat-d`. It is clear that overall `crsat-d` takes significantly fewer steps than `crsat` to find a solution. Despite the fact that the use of a heap incurs a run-time penalty, the differences in the number of search steps translate into the improvement in the run-time. As a result, `crsat-d` often solves more problems from the benchmark set than `crsat`. Figure 2 deserves special attention as it demonstrates that the performance improvements in `crsat-d` are more pronounced on difficult problems than on the easy ones. The depth-based justification selection seems to have particularly large impact on medium to hard problems from the `hwmcc08-sat` and `smtqfbv-sat` benchmark sets.

Interestingly, as the plots in Figure 3 demonstrate, the relative performance improvements are not necessarily directly related either to the depth of instances or their size. Thus, we conjecture that some additional structural properties of circuits are at play during the search for satisfying assignments in CRSAT.

Table 1. Performance comparison between `crsat` and `crsat-d`. The second column shows the number of instances from each benchmark class used for the comparison. The third and fourth columns show the number of instances that were solved (i.e., success rate over 50%) within the allocated 300 seconds per try, by each solver. The remaining four columns compare the total median number of steps and the total median run-time in CPU seconds taken by each solver on those instances that were solved by *both* solvers.

Benchmark class	Instances from this class	Instances solved `crsat-d`	Instances solved `crsat`	Total steps on solved by both `crsat-d`	Total steps on solved by both `crsat`	Total time on solved by both `crsat-d`	Total time on solved by both `crsat`
`hwmcc08-sat`	204	**137**	103	**7,227,753**	63,962,348	**197.41**	1289.12
`smtqfbv-sat`	61	**53**	38	**2,124,984**	15,242,248	**164.02**	651.50
`sss-sat-1.0`	96	**79**	74	**11,197,448**	25,020,062	**600.16**	1284.55
`vliw-sat-1.1`	98	94	**95**	**8,984,087**	10,084,139	**2014.17**	2057.44

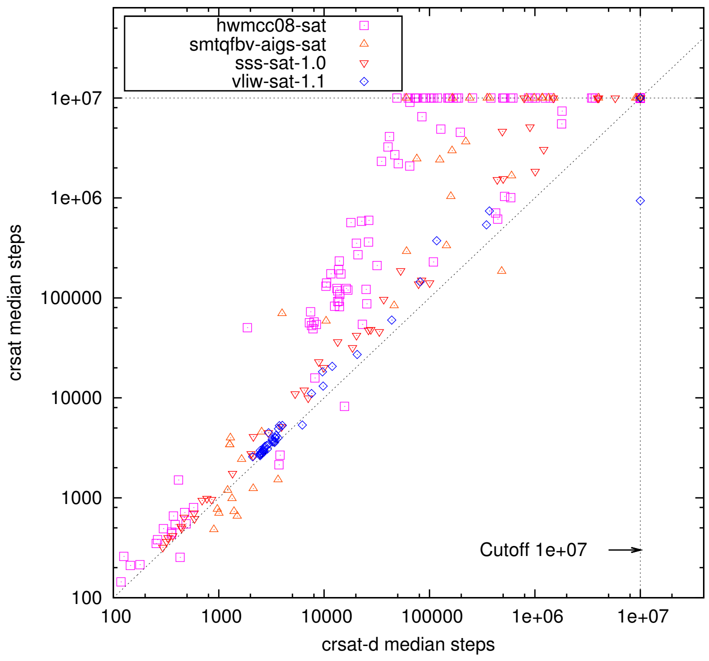

Fig. 2. Performance comparison between `crsat` and `crsat-d`. The plot, drawn on the logarithmic scale, depicts the correlation between median number of steps required by the solvers to find solutions – each median was measured from 100 tries for each instance, per solver. Those instances where the performance differences are not statistically significant (p-value of U-test > 0.05) are not shown.

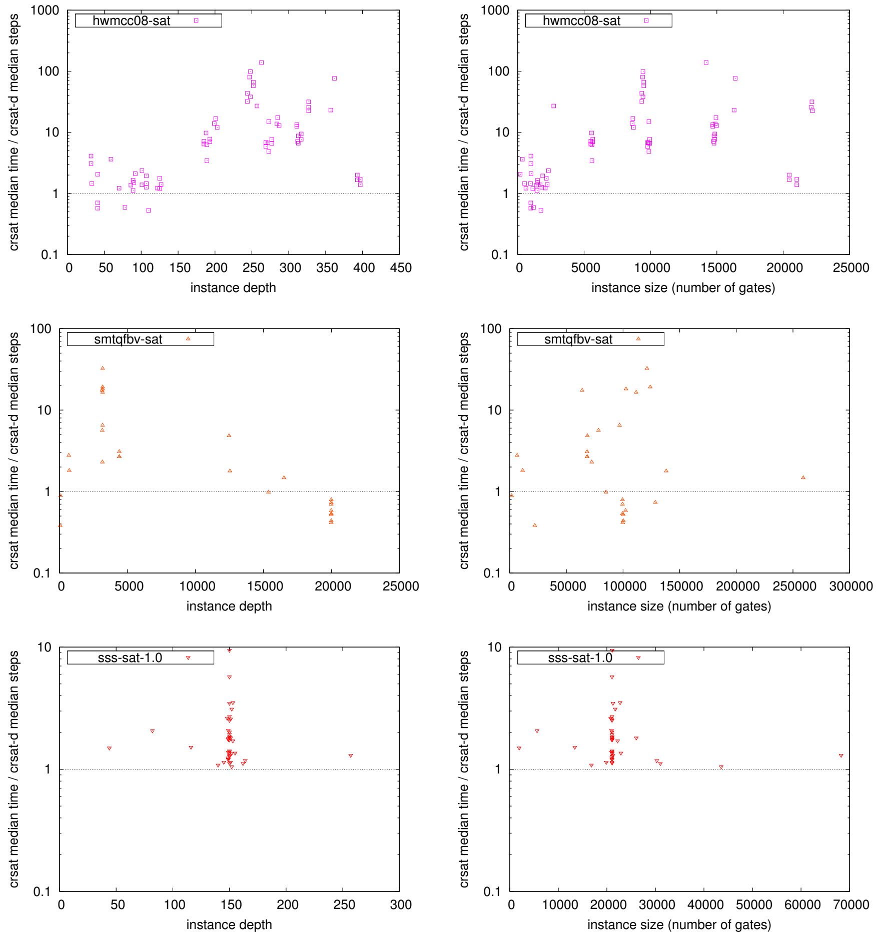

Fig. 3. The correlation between the performance improvement in `crsat-d`, as opposed to `crsat`, with the depth and the size of the circuits solvable by both solvers from three benchmarks sets. Those instances where the performance differences are not statistically significant (p-value of U-test > 0.05) are not shown.

7 Conclusions

We proposed a new, simple structure-based heuristic for the justification-based circuit-level SLS method CRSAT that combines justification-based SLS with forward propagation. Resulting in the variant DEPTH-CRSAT, the proposed depth-based heuristic increases the efficiency of CRSAT on various structural real-world circuit benchmark classes. We also showed that the concept of gate depth plays a central role in the proof of the fact that CRSAT is probabilistically approximately complete (PAC). This immediately implies that even the restricted variant DEPTH-CRSAT is also PAC, which further motivates the depth-based heuristic. This also implies that focusing search based on gate depth is in a sense a natural choice when combined with limited forward propagation. Additionally, one should notice that there are cases when the depth-based heuristic focuses search also on gates that do not appear in the justification frontier, in contrast to the BC SLS method.

The structural property of gate depth, while quite a simple one, yields good results as a heuristic in practice when applied in conjunction with limited forward propagation. This raises the question of whether there are more intricate structural gate properties—or even *combinations* of different properties—harnessing which even more effective and focused search heuristics could be developed. Indeed, developing a deeper understanding of the central structural properties for enhancing circuit-level SLS, especially in conjunction with forward propagation, remains an important part of further work. In addition to the important factor of practical efficiency, deeper insights into the interplay between problem structure and search algorithms are required.

References

1. The Satisfiability Modulo Theories Competition (SMT-COMP) 2009. `http://www.smtcomp.org/2009/`
2. Belov, A.: Stochastic Local Search for Non-clausal and Circuit Satisfiability. Ph.D. thesis, York University, Toronto, Canada (2010), in preparation
3. Belov, A., Stachniak, Z.: Improving variable selection process in stochastic local search for propositional satisfiability. In: Kullmann, O. (ed.) Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing (SAT 2009). Lecture Notes in Computer Science, vol. 5584, pp. 258–264. Springer (2009)
4. Belov, A., Stachniak, Z.: Improved local search for circuit satisfiability. In: Strichman, O., Szeider, S. (eds.) Proceedings of the 13th International Conference on Theory and Applications of Satisfiability Testing (SAT 2010). Lecture Notes in Computer Science, vol. 6175, pp. 293–299. Springer (2010)
5. Biere, A.: The AIGER library and set of utilities for And-Inverter Graphs (AIGs). `http://fmv.jku.at/aiger/`
6. Biere, A., Heule, M.J.H., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185. IOS Press (2009)
7. Brayton, R.K., Mishchenko, A.: ABC: An academic industrial-strength verification tool. In: Touili, T., Cook, B., Jackson, P. (eds.) Proceedings of the 22nd International Conference on Computer Aided Verification (CAV 2010). Lecture Notes in Computer Science, vol. 6174, pp. 24–40. Springer (2010)

8. Brummayer, R., Biere, A.: Boolector: An efficient SMT solver for bit-vectors and arrays. In: Kowalewski, S., Philippou, A. (eds.) Proceedings of the 15th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2009). Lecture Notes in Computer Science, vol. 5505, pp. 174–177. Springer (2009)
9. Darwiche, A., Pipatsrisawat, K.: Complete algorithms. In: Biere, A., Heule, M.J.H., van Maaren, H., Walsh, T. (eds.) Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185, chap. 3, pp. 75–98. IOS Press (2009)
10. Hoos, H., Stutzle, T.: Stochastic Local Search: Foundations and Applications. Elsevier / Morgan Kaufmann, San Francisco, CA, USA (2004)
11. Hoos, H.H.: On the run-time behaviour of stochastic local search algorithms for SAT. In: Proceedings of the 16th National Conference on Artificial Intelligence (AAAI 1999). pp. 661–666. AAAI Press (1999)
12. Järvisalo, M., Junttila, T., Niemelä, I.: Justification-based local search with adaptive noise strategies. In: Cervesato, I., Veith, H., Voronkov, A. (eds.) Proceedings of the 15th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR 2008). Lecture Notes in Computer Science, vol. 5330, pp. 31–46. Springer (2008)
13. Järvisalo, M., Junttila, T.: On the power of top-down branching heuristics. In: Fox, D., Gomes, C.P. (eds.) Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI 2008). pp. 304–309. AAAI Press (2008)
14. Järvisalo, M., Junttila, T.A., Niemelä, I.: Justification-based non-clausal local search for SAT. In: Ghallab, M., Spyropoulos, C.D., Fakotakis, N., Avouris, N.M. (eds.) Proceedings of the 18th European Conference on Artificial Intelligence (ECAI 2008). Frontiers in Artificial Intelligence and Applications, vol. 178, pp. 535–539. IOS Press (2008)
15. Junttila, T., Niemelä, I.: Towards an efficient tableau method for Boolean circuit satisfiability checking. In: Lloyd, J., Dahl, V., Furbach, U., Kerber, M., Lau, K.K., Palamidessi, C., Pereira, L.M., Sagiv, Y., Stuckey, P.J. (eds.) Proceedings of the 1st International Conference on Computational Logic (CL 2000). Lecture Notes in Artificial Intelligence, vol. 1861, pp. 553–567. Springer (2000)
16. Kautz, H., McAllester, D., Selman, B.: Exploiting variable dependency in local search. In: IJCAI poster session (1997), `http://www.cs.rochester.edu/u/kautz/papers/dagsat.ps`
17. Kautz, H., Selman, B.: The state of SAT. Discrete Applied Mathematics 155(12), 1514–1524 (2007)
18. Kautz, H., Sabharwal, A., Selman, B.: Incomplete algorithms. In: Biere, A., Heule, M.J.H., van Maaren, H., Walsh, T. (eds.) Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185, chap. 6, pp. 185–204. IOS Press (2009)
19. Kuehlmann, A., Ganai, M., Paruthi, V.: Circuit-based Boolean reasoning. In: Proceedings of the 38th Design Automation Conference (DAC 2001). pp. 232–237. ACM (2001)
20. Kuehlmann, A., Paruthi, V., Krohm, F., Ganai, M.K.: Robust Boolean reasoning for equivalence checking and functional property verification. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 21(12), 1377–1394 (2002)
21. Marques-Silva, J., Lynce, I., Malik, S.: CDCL solvers. In: Biere, A., Heule, M.J.H., van Maaren, H., Walsh, T. (eds.) Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185, chap. 4, pp. 99–154. IOS Press (2009)
22. Muhammad, R., Stuckey, P.J.: A stochastic non-CNF SAT solver. In: Yang, Q., Webb, G.I. (eds.) Proceedings of the 9th Pacific Rim International Conference on Artificial Intelligence (PRICAI 2006). Lecture Notes in Computer Science, vol. 4099, pp. 120–129. Springer (2006)
23. Pham, D., Thornton, J., Sattar, A.: Building structure into local search for SAT. In: Veloso, M.M. (ed.) Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007). pp. 2359–2364 (2007)

24. Prestwich, S.D.: Variable dependency in local search: Prevention is better than cure. In: Marques-Silva, J., Sakallah, K.A. (eds.) Proceedings of the 10th International Conference on Theory and Applications of Satisfiability Testing (SAT 2007). Lecture Notes in Computer Science, vol. 4501, pp. 107–120. Springer (2007)
25. Sebastiani, R.: Applying GSAT to non-clausal formulas. Journal of Artificial Intelligence Research 1, 309–314 (1994)
26. Sheskin, D.: Handbook of Parametric and Nonparametric Statistical Procedures. CRC Press (1997)
27. Stachniak, Z.: Going non-clausal. In: SAT 2002, Fifth International Symposium on the Theory and Applications of Satisfiability Testing, May 6-9, 2002 Cincinnati, Ohio, USA (2002), `http://gauss.ececs.uc.edu/Conferences/SAT2002/Abstracts/stachniak.ps`
28. Stachniak, Z., Belov, A.: Speeding-up non-clausal local search for propositional satisfiability with clause learning. In: Büning, H.K., Zhao, X. (eds.) Proceedings of the 11th International Conference on Theory and Applications of Satisfiability Testing (SAT 2008). Lecture Notes in Computer Science, vol. 4996, pp. 257–270. Springer (2008)
29. Thiffault, C., Bacchus, F., Walsh, T.: Solving non-clausal formulas with DPLL search. In: Wallace, M. (ed.) Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming (CP 2004). Lecture Notes in Computer Science, vol. 3258, pp. 663–678. Springer (2004)
30. Velev, M.: Miroslav Velev's SAT Benchmarks. `http://www.miroslav-velev.com/sat_benchmarks.html`

Aalto University

UNIVERSITY OF HELSINKI

We propose a structure-exploiting heuristic for the justification-based stochastic local search (SLS) method CRSat for Boolean circuit satisfiability. Experimental evaluation shows that the proposed depth-based heuristic significantly improved the performance of CRSat on structural instances arising from industrial applications. A proof of probabilistically approximate completeness (PAC) of CRSat relies on the same kind of variable ordering as the one imposed by the depth-based heuristic, and hence provides a theoretical motivation for the heuristic. Furthermore, we compare the behavior of (depth-based) CRSat to that of the justification-based SLS method BC SLS driven by justification frontiers.

HIIT Technical Reports 2010-2

ISBN 978-952-60-3415-7 (electronic)
ISSN 1458-9478 (electronic)