Depth-Driven Circuit-Level Stochastic Local Search for SAT

Anton Belov[*]
University College Dublin
Ireland

Matti Järvisalo[†]
University of Helsinki
Finland

Zbigniev Stachniak
York University
Canada

Abstract

We develop a novel circuit-level stochastic local search (SLS) method D-CRSat for Boolean satisfiability by integrating a structure-based heuristic into the recent CRSat algorithm. D-CRSat significantly improves on CRSat on real-world application benchmarks on which other current CNF and circuit-level SLS methods tend to perform weakly. We also give an intricate proof of probabilistically approximate completeness for D-CRSat, highlighting key features of the method.

1 Introduction

Today, Boolean satisfiability (SAT) solvers are routinely used to solve hard problem instances arising from AI research and various industrial applications. The most efficient SAT-based approach to solving such real-world instances is typically based on conflict-driven clause learning (CDCL). In contrast, stochastic local search (SLS) for SAT is often considered effective mainly on random SAT instances. Only recently some research effort has been directed towards making SLS a noteworthy alternative for solving real-world instances. This work contributes substantially to these efforts by developing novel structure-based SLS techniques that can lift the performance of SLS closer to that of CDCL on real-world application instances.

A major challenge in improving the performance of SLS on real-world application instances is in developing efficient techniques that exploit *variable dependencies* [Kautz and Selman, 2007]. The results in this paper indicate that a key to solving this challenge is to exploit non-clausal formula representations (such as *Boolean circuits*) instead of focusing on the customary approach of first translating formulas into the flat conjunctive normal form (CNF) format and then applying CNF-level SLS methods.

Most SLS methods previously proposed for non-clausal formulas focus search on truth assignments over *independent* (or *input*) variables [Sebastiani, 1994; Kautz *et al.*, 1997; Stachniak, 2002; Pham *et al.*, 2007; Muhammad and Stuckey, 2006; Stachniak and Belov, 2008; Belov and Stachniak, 2009]. An alternative approach to circuit-level SLS was first proposed in the BC SLS method [Järvisalo *et al.*, 2008b; 2008a]. In contrast to searching in a bottom-up mode as in circuit-level SLS methods focusing on input variables, search in BC SLS is driven top-down in the overall structure of the circuit by utilizing so-called *justification frontiers*. BC SLS also introduced the concept of *justification-based* SLS, in which search steps aim at correcting local inconsistencies within a circuit by *justifying* inconsistently assigned (*unjustified*) gates. The more recent circuit-level SLS method CRSat builds on the concept of justification-based search [Belov and Stachniak, 2010]. The key novel feature of CRSat is the incorporation of limited constraint propagation into the search. Additionally, CRSat relaxes justification-based search to consider all unjustified gates. CRSat was shown to significantly outperform BC SLS on problem instances from real-world application domains [Belov and Stachniak, 2010].

In this work we develop a novel *depth-driven* circuit-level SLS method D-CRSat. D-CRSat exploits the explicit circuit-level instance structure through a symbiosis of *limited forward propagation* and a structure-based heuristic based on *gate depth information* within justification-based search. We show that D-CRSat significantly improves the performance of CRSat on a wide range of industrial application benchmarks on which both bottom-up mode circuit-level SLS and current CNF-level SLS methods (using a standard CNF translation for the latter) tend to perform weakly. In fact, D-CRSat compares in some cases favourably even with a modern circuit-level CDCL solver. Complementing these major practical improvements, we give a proof of *probabilistically approximate completeness* (PAC) [Hoos, 1999] for D-CRSat. While also interesting in its own right, the proof reveals that gate depth information is indeed an important search parameter. Compared to typical PAC proofs, the case of D-CRSat is more involved due to non-local changes caused by the constraint propagation mechanism. The proof also reveals that D-CRSat (and CRSat) has an intrinsic ability to autonomously restart search without explicitly being forced to do so. An additional side-product of the proof is that CRSat is also PAC.

2 Preliminaries

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

[*]Partially supported by SFI PI grant BEACON (09/IN.1/I2618).
[†]Financially supported by Academy of Finland (grant 132812).

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 $\langle G, E(C)\rangle$, where $E(C) = \{\langle g', g\rangle \in G \times G \mid g = f(\ldots, g', \ldots) \in C\}$, 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 its 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 fanin^* and fanout^* are 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). A gate with no children (resp. no parents) is an *input gate* (resp. 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 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. The *domain* of τ, i.e., the set of gates assigned in τ, is denoted by $dom(\tau)$. Two assignments, τ and τ', *disagree* on a gate $g \in dom(\tau) \cap dom(\tau')$ if $\tau(g) \neq \tau'(g)$. Furthermore, we identify $\langle g, v\rangle \in \tau$ with $\tau(g) = v$.

A *constrained Boolean circuit* C^α consists of a Boolean circuit C and an assignment α for C. 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 assignment satisfies C^α, then C^α is *satisfiable*. A circuit that is not satisfiable is *unsatisfiable*. Without loss of generality, we assume that constraints are imposed only on output gates.

The *restriction* $\tau|_{G'}$ of an assignment τ to a set $G' \subseteq G$ of gates is defined as $\{\langle g, v\rangle \in \tau \mid g \in G'\}$. Given a gate equation $g = f(g_1, \ldots, g_n)$ 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. For example, the justifications for $\langle g, 0\rangle$, where $g = \text{AND}(g_1, g_2)$, are $\{\langle g_1, 0\rangle\}$, $\{\langle g_2, 0\rangle\}$, and $\{\langle g_1, 0\rangle, \langle g_2, 0\rangle\}$, out of which the first two are *subset-minimal*. 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 $\mathsf{unjust}(C^\alpha, \tau)$.

For a truth assignment τ and set of gates $G \subseteq dom(\tau)$, let

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

In other words, $\mathit{flip}(G, \tau)$ is the truth assignment obtained by changing the values of the gates in G, and leaving the rest of τ unchanged.

3 Justification-Based Circuit-Level SLS

This section provides an overview of the justification-based circuit-level SLS methods BC SLS and CRSAT. We start with BC SLS which introduced the idea of justification-based SLS [Järvisalo *et al.*, 2008b] that was later adopted in CRSAT [Belov and Stachniak, 2010]

3.1 BC SLS

In BC SLS search is driven by the dynamically updated *justification frontier* $\mathsf{jfront}(C^\alpha, \tau)$ of the constrained circuit C^α based on the current assignment τ. For a given τ, consider the smallest set S of gates which includes all constrained gates and, for each justified gate g in S, all the gates that participate in some subset-minimal justification for g. The justification frontier $\mathsf{jfront}(C^\alpha, \tau)$ is the "bottom edge" of S, consisting of those gates in S that are not justified. By definition, we always have $\mathsf{jfront}(C^\alpha, \tau) \subseteq \mathsf{unjust}(C^\alpha, \tau)$.

BC SLS exploits the fact that when the justification frontier $\mathsf{jfront}(C^\alpha, \tau)$ is empty, the constrained circuit C^α is satisfiable. Starting with a random complete assignment τ for C^α, and as long as the justification frontier $\mathsf{jfront}(C^\alpha, \tau)$ is not empty, the algorithm removes a gate g from $\mathsf{jfront}(C^\alpha, \tau)$ at random, and performs either a *downward move*—which is central in justification-based search—or an *upward move*.

In a *downward* move the gate g is justified by choosing a justification σ for $\langle g, \tau(g)\rangle$, and flipping the values of gates on which σ and τ disagree. As a result, some gates in $\mathsf{fanin}(g)$ may become unjustified and are hence added to the updated justification frontier. To choose σ, the set Σ of all justifications for $\langle g, \tau(g)\rangle$ is constructed, and one element is selected from this set either at random, or greedily with the objective of minimizing the size of the set of gates in $\mathsf{jfront}(C^\alpha, \tau)$ and their descendants after the move.

In an *upward* move the value of g itself is flipped (requiring that g is unconstrained). As a result, g becomes justified, and, potentially, some gates in $\mathsf{fanout}(g)$ become unjustified.

Intuitively, BC SLS balances between moving the justification frontier towards the inputs with downward moves with the upward moves that, in essence, assign to the chosen gate the value that is consistent with the assignment on its children, hence pushing the frontier towards the outputs.

3.2 CRSAT

The CRSAT algorithm also applies the idea of justification-based search. However, in contrast to BC SLS, CRSAT does not maintain the justification frontier or perform explicit upward moves. Instead, CRSAT incorporates a more direct *limited forward propagation* mechanism—restricted bottom-up circuit-level constraint propagation—within the search.

Pseudo-code for CRSAT is presented as Algorithm 1. First, a complete extension of a random value assignment to $\mathsf{inputs}(C^\alpha)$ is constructed, i.e., the value of each unconstrained internal gate is set consistently with the values of its children. Then, as long as $\mathsf{unjust}(C^\alpha, \tau)$ is not empty (i.e., τ is not a satisfying assignment), the algorithm selects a gate g from $\mathsf{unjust}(C^\alpha, \tau)$ at random (line 6), and, similarly to the downward moves in BC SLS, justifies g by choosing a justification σ for $\langle g, \tau(g)\rangle$ and flipping the values of gates on which σ and τ disagree. However, CRSAT additionally propagates the consequences of the flip using limited forward propagation. These two actions form a *step* of CRSat (see function $\textsc{Step}(C^\alpha, G, \tau)$ on lines 19-22).

The justification σ used to make a step is selected from the set Σ of all justifications for $\langle g, \tau(g)\rangle$ either at random (with probability wp), or greedily with the objective of minimizing the number of unjustified gates after the step.

Algorithm 1 CRSAT(C^α, wp, *cutoff*)

Input: C^α – constrained Boolean circuit
wp – noise parameter ,i.e., probability of random walk
cutoff – cutoff, i.e., maximum number of steps
Output: $status$ – SAT if a satisfying assignment for C^a is found, UNKNOWN otherwise
τ – a satisfying assignment for C^α if found, $\emptyset$ otherwise

1: $\tau \leftarrow$ a complete extension of a random assignment to $\mathsf{inputs}(C^\alpha)$
2: *steps* $\leftarrow 0$
3: **while** *steps* $<$ *cutoff* **do**
4: **if** $\mathsf{unjust}(C^\alpha, \tau) = \emptyset$ **then**
5: **return** $\langle \mathtt{SAT}, \tau \rangle$
6: $g \leftarrow$ a random element from $\mathsf{unjust}(C^\alpha, \tau)$
7: $\Sigma \leftarrow$ the set of justifications for $\langle g, \tau(g) \rangle$
8: **with-probability** wp **do**
9: $\sigma \leftarrow$ random element of Σ ▷ random walk
10: **otherwise**
11: $\sigma \leftarrow$ a random justification from the justifications in Σ
12: that minimize $|\mathsf{unjust}(C^\alpha, \cdot)|$ after step
13: ▷ greedy downward move
14: **end with-probability**
15: $G \leftarrow$ set of gates in σ that disagree with τ
16: $\tau \leftarrow$ STEP(C^α, G, τ) ▷ flip + limited forward propagation
17: *steps* $\leftarrow$ *steps* $+ 1$
18: **return** $\langle \mathtt{UNKNOWN}, \emptyset \rangle$

19: **function** STEP(C^α, G, τ)
20: $\tau' \leftarrow$ *flip*(G, τ)
21: $\tau' \leftarrow$ LBCP-FORWARD(C^α, G, τ')
22: **return** τ'

3.3 Limited Forward Propagation

We now provide details on how to implement the limited forward propagation mechanism (Algorithm 2) that is one of the key techniques in both CRSAT and the method developed in this paper. More details can be found in [Belov, 2010].

The propagation algorithm uses a priority queue Q of gates (without duplicates) that allows to query the *smallest* gate according to a topological order in constant time. Recall that a topological order on the set of gates in a circuit is any strict total order $<$ that respects the condition "if $g_1 \in \mathsf{fanin}(g_2)$, then $g_1 < g_2$".

Given a set of gates G (that presumably have just changed their value as a result of a flip), the algorithm starts by inserting the gates into Q. Then, for each gate g removed from Q, the algorithm queues all gates in $\mathsf{fanout}(g)$ if either $g \in G$, or g is unjustified and not constrained. In the latter case the value of g is flipped, and so g becomes justified. Thus, informally, given the set of gates G the algorithm propagates their values towards the outputs of the circuit, but only as long as some gates change their values.

The following proposition captures a key property of the forward propagation procedure applied within CRSAT.

Proposition 1 *Let C^α be a constrained circuit, τ an assignment for C^α, and G be a set of gates constructed on line 15 of* CRSAT. *Let $\tau' =$ STEP(C^α, G, τ). Then:*

(i) For all $g \notin G \cup dom(\alpha)$, if $g \in \mathsf{unjust}(C^\alpha, \tau')$, then $g \in \mathsf{unjust}(C^\alpha, \tau)$.

Algorithm 2 LBCP-FORWARD(C^α, G, τ)

Input: C^α – constrained Boolean circuit;
G – a set of gates whose value changes are to be propagated.
τ – an assignment for C^a;
Output: τ' – an assignment for C^α 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($\mathsf{fanout}(g)$)
7: **else**
8: **if** $g \in \mathsf{unjust}(C^\alpha, \tau') \setminus dom(\alpha)$ **then**
9: ▷ g unconstrained and unjustified
10: $\tau' \leftarrow$ *flip*$(\{g\}, \tau')$
11: Q.ENQUEUE($\mathsf{fanout}(g)$)
12: **return** τ'

(ii) For all $g \notin G$, if $g \in \mathsf{unjust}(C^\alpha, \tau')$, then $\tau'(g) = \tau(g)$.

Thus, a step of CRSAT does not create new unjustified gates beside, possibly, those in G and $dom(\alpha)$. Furthermore, any gate not in G that is unjustified after a step has the same value as before the step.

4 D-CRSAT: Depth-Based CRSAT

The efficiency of justification-based search depends critically on how gates are selected for justification during search. In [Belov and Stachniak, 2010] no explicit gate selection heuristic for CRSAT was proposed. On the other hand, the search heuristic of BC SLS is tightly bound to the justification frontier, causing a single move of BC SLS to be quite expensive in practice. In this section we introduce a structure-based gate selection heuristic which, when incorporated into CRSAT, results in significant performance improvement with only small overhead. We call the resulting method D-CRSAT. In the following sections we provide theoretical justifications and experimental evidence for D-CRSAT's good performance.

Given a constrained Boolean circuit C^α, for each gate g in C^α we define the *depth* of g in C^α as $\mathsf{depth}(C^\alpha, g) = 0$ if $g \in \mathsf{outputs}(C)$ and, otherwise,

$$\mathsf{depth}(C^\alpha, g) = 1 + \max\{\mathsf{depth}(C^\alpha, g') \mid g' \in \mathsf{fanout}(g)\}.$$

We denote by D-CRSAT the version of CRSAT in which the unjustified gate g on line 6 is selected at random from the set of gates in $\mathsf{unjust}(C^\alpha, \tau)$ that are *at maximum depth* according to $\mathsf{depth}(C^\alpha, g)$. In other words, D-CRSAT always selects the unjustified gate g from the set of gates

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

Proposition 1 provides some intuition as to why the combination of this depth-based heuristic and limited forward propagation may be appealing. Namely, forward propagation does not create new unjustified unconstrained gates, and hence can only make currently unjustified gates justified. Therefore, choosing a gate for justification from the set of unjustified

gates *at maximum depth* provides forward propagation with more opportunities to justify gates at smaller depths.

The depth-based heuristic makes D-CRSAT a more focused refinement of CRSAT (in CRSAT gate selection is done at random), but even so, we establish that D-CRSAT is probabilistically approximately complete. In fact, the proof of this fact shows that D-CRSAT may find a satisfying assignment using a significantly smaller number of steps than CRSAT.

5 PAC and Restarts

An SLS algorithm is *probabilistically approximately complete (PAC)* [Hoos, 1999] if the probability of finding a solution to any satisfiable instance is asymptotically 1.

Definition 1 *A SAT algorithm* A *is* probabilistically approximately complete (PAC) *if for any satisfiable instance F, $\lim_{t\to\infty} P(RT_{\mathrm{A},F} \leq t) = 1$, where $P(RT_{\mathrm{A},F} \leq t)$ 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.*

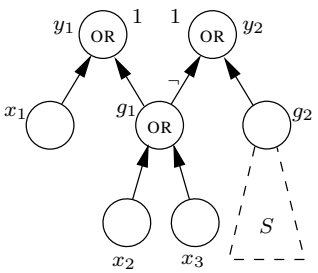
Our *main theoretical result* is the following.

Theorem 1 D-CRSAT *with any noise parameter value $wp > 0$ and infinite cutoff is PAC.*

For full versions of the proofs in this section, see [Belov, 2010]. Before proceedings with a proof of Theorem 1, we show that the condition $wp > 0$ is indeed necessary.

Theorem 2 D-CRSAT *with $wp = 0$ and infinite cutoff is essentially incomplete.*

Proof. Consider the circuit C^α



where $\alpha = \{\langle y_1, 1\rangle, \langle y_2, 1\rangle\}$ and g_2 is the output gate of a sub-circuit S such that $\tau^*(g_2) = 0$ for any satisfying assignment τ^* for C^α. Assume that the initial assignment is $\tau_0 = \alpha \cup \{\langle g_1, 1\rangle, \langle g_2, 0\rangle, \langle x_1, 0\rangle, \langle x_2, 1\rangle, \langle x_3, 1\rangle, \ldots\}$ and that every gate in S is justified under τ_0. Now $\mathsf{unjust}(C^\alpha, \tau_0) = \{y_2\}$, and D-CRSAT greedily flips the value of g_2 to 1, making g_2 unjustified. Since S is unsatisfiable when g_2 is assigned to 1, D-CRSAT will either get stuck inside S, or will return to the assignment τ_0 and greedily flip g_2 again. □

It is easy to show that any CNF-level SLS algorithm that allows random walk during *any step* is PAC [Hoos, 1999]: random walk guarantees that there is a non-zero probability of the event that the Hamming distance from the current assignment τ to some fixed satisfying assignment τ^* is decreased. This is because any unsatisfied clause c must have at least one variable set in disagreement with τ^* and so with the probability $1/|c|$, this particular variable will be flipped during the random walk. Similar argument, generalized to the setting of constrained Boolean circuits, establishes the PAC property of BC SLS [Järvisalo *et al.*, 2008a].

For D-CRSAT this approach to proving PAC does not work, since D-CRSAT cannot flip the unjustified gate selected on line 6. Thus, if during a step all unjustified gates are assigned in disagreement with τ^*, then D-CRSAT may be unable to decrease the Hamming distance to τ^* in this step. This situation could be avoided if D-CRSAT could start with an assignment in which all unjustified gates were assigned as in τ^*, and could always choose justifications that agree with τ^*; by Proposition 1, all unjustified gates would then always be assigned as in τ^*. Observe that for any assignment τ with $\mathsf{unjust}(C^\alpha, \tau) \subseteq dom(\alpha)$ (we call such τ's *restart assignments*), the unjustified gates *are* assigned as in τ^*.

Additional difficulty in proving PAC is caused by the fact that forward propagation makes non-local changes to the current assignment. Hence, even when the value of one gate is set as in τ^* during a step, forward propagation may cause other gates in the circuit to be set in disagreement with τ^*. The solution is to monitor the changes in the Hamming distance between the current assignment τ and the selected satisfying assignment τ^*, both restricted to the *input gates*.

Lemma 1 *Let C^α be a satisfiable constrained Boolean circuit, and let $m \geq 0$ be such that the assignment τ_m at the beginning of the m-th iteration of* D-CRSAT *is a restart assignment. Then, there is a constant k, where*

$$0 \leq k \leq \mathsf{depth}(C^\alpha) \cdot |\mathsf{inputs}(C^\alpha)|, \tag{1}$$

such that the probability of the event that the assignment τ_{m+k} at the beginning of the $(m+k)$-th iteration is satisfying is at least

$$\left(\frac{wp}{2^f - 1}\right)^k, \tag{2}$$

where f is the maximum size of fanin among the gates in C^α.

Proof sketch. Assume that τ_m is not satisfying (otherwise $k = 0$), and let τ^* be some satisfying assignment for C^α. Consider those executions of D-CRSAT in which the justification σ selected on line 14 is such that $\tau^*|_{dom(\sigma)} = \sigma$. At each step, the probability of selecting such a justification is at least $wp/(2^f - 1)$ (i.e., the probability of taking a random walk, and selecting the justification that agrees with τ^* from the $2^f - 1$ possibilities). Thus, the probability of selecting justifications according to τ^* during any k consecutive steps is at least (2). By Proposition 1, for any k, τ_{m+k} agrees with τ^* on the values of all unjustified gates. Since τ_m is not satisfying, at least one constrained gate is unjustified, and, for each unjustified gate g, $\mathsf{fanin}^*(g)$ must contain at least one input gate that disagrees with τ^*. In the worst case all $|\mathsf{inputs}(C^\alpha)|$ input gates disagree with τ^*. D-CRSAT has to make at most $\mathsf{depth}(C^\alpha)$ steps to assign values as in τ^* to such gates: D-CRSAT always selects a gate of maximal depth for justification and, hence, some input gate will be flipped after no more than $\mathsf{depth}(C^\alpha)$ steps. Since input gates are *not* affected by forward propagation, once an input gate is assigned according to τ^* it will not change its value in the future. Hence the number k of steps required to assign all inputs according to τ^* satisfies (1). Hence $\tau_{m+k}|_{\mathsf{inputs}(C^\alpha)} = \tau^*|_{\mathsf{inputs}(C^\alpha)}$, and therefore $\tau_{m+k} = \tau^*$ (otherwise there would be a gate unjustified under τ_{m+k} whose assignment disagrees with τ^*). □

We observe that D-CRSAT starts the search from a restart assignment (line 1), and, in fact, will modify any assignment into a restart assignment in a bounded number of steps:

Lemma 2 *Let C^α be a constrained Boolean circuit, and let τ_m be the assignment at the beginning of the m-th iteration of* D-CRSAT. *Then, there is a constant k, where*

$$0 \leq k \leq |G|^2 \cdot \mathsf{depth}(C^\alpha), \tag{3}$$

such that the assignment τ_{m+k} at the beginning of the $(m+k)$-th iteration is a restart assignment.

Proof sketch. Assume that τ_m is not a restart assignment; otherwise we are done. Let $g_m \notin dom(\alpha)$ be the unjustified gate selected at the m-th iteration. All gates in $\mathsf{fanin}^*(g_m)$ are justified. To justify g_m without making any gate in $\mathsf{fanin}^*(g_m)$ unjustified, the algorithm needs to

(i) follow a path from g_m down to some input gate and justify every gate along the path (at most $\mathsf{depth}(C^\alpha)$ steps);

(ii) select the next unjustified gate and repeat (i). This gate has to be in $\mathsf{fanin}^*(g_m)$ since we always select unjustified gates at maximum depth, and by Proposition 1 the only new unjustified gates resulting from step (i) are in $\mathsf{fanin}^*(g_m) \cup dom(\alpha)$; and

(iii) repeat (ii) for all gates in $\mathsf{fanin}^*(g_m)$.

Since $|\mathsf{fanin}^*(g_m)| \leq |G|$, g_m and all gates in $\mathsf{fanin}^*(g_m)$ will be justified in at most $|G| \cdot \mathsf{depth}(C^\alpha)$ steps. Again, by Proposition 1 the only new unjustified gates that will be created during this process are in $dom(\alpha)$. Since at iteration m there are at most $|G|$ unjustified gates outside of $dom(\alpha)$, after at most $|G|^2 \cdot \mathsf{depth}(C^\alpha)$ steps all such unjustified gates will be justified, resulting in a restart assignment. □

With Lemmas 1 and 2 we can prove Theorem 1.

Proof sketch of Theorem 1. Consider any execution of D-CRSAT. Let τ_i be an assignment on step i of this execution, and let X_i be a random variable with

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

Let k_1 and k_2 be the bounds (1) and (3), respectively. The sequence $\langle X_0, X_{k_1}, X_{k_1+k_2}, X_{2k_1+k_2}, X_{2k_1+2k_2}, \dots \rangle$ is a Markov chain. Using Lemmas 1 and 2, one can show that state 2 is the only persistent state. □

The proof of Theorem 1 also applies to CRSAT as, with non-zero probability, it can always justify a gate at maximum depth. However, compared to D-CRSAT, the expected number of steps (recall Lemmas 1 and 2) can be significantly larger, i.e., in theory D-CRSAT converges to a satisfying assignment significantly faster. The experimental results presented next confirm that this is also the case in practice.

Furthermore, Lemma 2 reveals another intriguing property of both D-CRSAT and CRSAT. Namely, the algorithms are always bound to eventually return to a restart assignment. Since, by definition, both of the algorithms start the search from some restart assignment, Lemma 2 shows that D-CRSAT and CRSAT have the intrinsic ability to *dynamically restart without explicitly forced restarts*.

6 Experiments

We compare the performance of D-CRSAT to that of CRSAT (using our implementations `d-crsat` and `crsat`) and also to other circuit and CNF-level SLS methods. For this, we also implemented a circuit-level method `inputLS` that searches over assignments to input gates in the style of [Pham *et al.*, 2007] (the authors were unable to provide us with their implementation). We also used the CNF-level methods `TNM` and `slstc` that were one of the best SLS solvers in SAT Competition 2009 on random and application instances, resp., on the standard *Tseitin* CNF encodings of the benchmark circuits.

In both `d-crsat` and `crsat`), a justification at each step is selected from the set of subset minimal justifications for the selected gate. This is due to positive results in preliminary experiments for both methods. For retrieving the unjustified gates of maximum depth in `d-crsat`, the set $\mathsf{unjust}(C^\alpha, \tau)$ of unjustified gates is kept in a heap data-structure that allows to retrieve such a gate in constant time, but incurs a $\mathcal{O}(\log(|\mathsf{unjust}(C^\alpha, \tau)|))$ penalty for insertions.

For each solver, we obtained the empirical run-time and run-length distributions from 100 runs on each benchmark. The near-optimal random walk probability values were determined experimentally beforehand. The experiments were run under Linux on a Intel Core 2 Duo 3.00-GHz processor. As benchmarks, we used over 450 And-Inverted circuits (AIGs) from four different industrial application domains.

hwmcc08-sat 204 AIGs obtained from Hardware Model Checking Competition 2008 (http://fmv.jku.at/hwmcc08/) problems using aigtobmc (http://fmv.jku.at/aiger) with step bound $k = 45$ for time frame expansion.

smtqfbv-sat 61 AIGs generated using Boolector (http://fmv.jku.at/boolector/) to bit-blast `QF_BV` (theory of bit-vectors) instances of the SMT Competition 2009 (http://www.smtcomp.org/2009/)

sss-sat-1.0 98 AIGs from "formal verification of buggy variants of a dual-issue superscalar microprocessor" (http://www.miroslav-velev.com/sat_benchmarks.html) converted to AIGs with ABC (http://www.eecs.berkeley.edu/~alanmi/abc/).

vliw-sat-1.1 98 AIGs from "formal verification of buggy variants of a VLIW microprocessor", in the same fashion as sss-sat-1.0.

6.1 Results

Table 1 summarizes the results. The time and steps ratios are calculated from the median running times and number of steps. Overall, `d-crsat` takes significantly fewer steps than `crsat`. Despite the fact that the heap incurs a run-time penalty, the differences in the number of search steps translate into improvements in run-times. Improvements are pronounced on difficult problems (Figure 1, upper). `d-crsat` solves significantly more instances than the circuit-level method `inputLS` and the CNF-level methods `TNM` and `slstc`. It also solves a vast majority of those instances solved by the other solvers significantly faster.

We also compare `d-crsat` to `NoClause`, a circuit-level conflict-driven clause learning solver [Thiffault *et al.*, 2004] with many modern CDCL solver techniques in the style of zChaff (including VSIDS, 1-UIP learning and backjumping, watched literals, etc.); such CDCL techniques are at the center of state-of-the-art SAT solvers for industrial applications.

Table 1: Performance of d-crsat compared to other SLS-based SAT solvers. Here an instance is considered "solved" by a solver if the success rate on 100 tries with 300 stimeout per try is over 50%. The "time ratio" (resp. "step ratio") column for a solver shows the ratio of total time (resp. steps) taken by the solver to that of d-crsat on instances solved by *both* solvers.

Benchmark class (# instances)	d-crsat solved	crsat			inputLS		slstc		TNM	
		solved	time ratio	steps ratio	solved	time ratio	solved	time ratio	solved	time ratio
hwmcc08-sat (204)	**137**	103	6.58x	8.84x	17	0.31x	81	23.69x	50	331.83x
smtqfbv-sat (61)	**53**	38	3.97x	7.17x	25	10.32x	2	181.87x	1	1.00x
sss-sat-1.0 (96)	**79**	74	2.14x	2.23x	15	377.64x	64	4.12x	3	489.29x
vliw-sat-1.1(98)	94	**95**	1.02x	1.22x	68	23.90x	9	597.78x	0	n/a

Figure 1 (lower) shows that d-crsat compares in cases favourably even with NoClause, especially on the vliw-sat-1.1 family (although one should notice that NoClause is not as efficient as the currently most efficient CDCL solvers).

7 Conclusions

We developed a circuit-level SLS method D-CRSat that combines justification-based SLS with structure-based heuristics and limited reasoning by forward propagation. We showed experimentally that D-CRSat outperforms CRSat on various classes of real-world circuit benchmarks, and dominates other recent circuit and CNF-level SLS methods, including an implementation of circuit-level SLS focusing on input variables. In some cases, CRSat compares favourably even with a circuit-level conflict-driven clause learning solver. This indicates that further advances in SLS-based techniques could make SLS a viable alternative to CDCL-based algorithms for solving instances from real-world application domains. Complementing our experimental results, the presented intricate PAC proof for D-CRSAT provides key insights into the proposed gate selection heuristic, highlighting the gate depth as an important search parameter and revealing the intrinsic ability of D-CRSAT to dynamically restart search.

References

[Belov and Stachniak, 2009] A. Belov and Z. Stachniak. Improving variable selection process in stochastic local search for propositional satisfiability. In *Proc. SAT*, volume 5584 of *LNCS*, pages 258–264. Springer, 2009.

[Belov and Stachniak, 2010] A. Belov and Z. Stachniak. Improved local search for circuit satisfiability. In *Proc. SAT*, volume 6175 of *LNCS*, pages 293–299. Springer, 2010.

[Belov, 2010] A. Belov. *Stochastic Local Search for Non-clausal and Circuit Satisfiability*. PhD thesis, York University, 2010.

[Hoos, 1999] Holger H. Hoos. On the run-time behaviour of stochastic local search algorithms for SAT. In *Proc. AAAI*, pages 661–666. AAAI Press, 1999.

[Järvisalo *et al.*, 2008a] M. Järvisalo, T. Junttila, and I. Niemelä. Justification-based local search with adaptive noise strategies. In *Proc. LPAR*, volume 5330 of *LNCS*, pages 31–46. Springer, 2008.

[Järvisalo *et al.*, 2008b] M. Järvisalo, T.A. Junttila, and I. Niemelä. Justification-based non-clausal local search for SAT. In *Proc. ECAI*, pages 535–539. IOS Press, 2008.

[Kautz and Selman, 2007] H.A. Kautz and B. Selman. The state of SAT. *Discrete Applied Mathematics*, 155(12):1514–1524, 2007.

[Kautz *et al.*, 1997] H. Kautz, D. McAllester, and B. Selman. Exploiting variable dependency in local search. In *IJCAI*, 1997.

[Muhammad and Stuckey, 2006] R. Muhammad and P.J. Stuckey. A stochastic non-CNF SAT solver. In *Proc. PRICAI*, volume 4099 of *LNCS*, pages 120–129. Springer, 2006.

[Pham *et al.*, 2007] D.N. Pham, J. Thornton, and A. Sattar. Building structure into local search for SAT. In *Proc. IJCAI*, pages 2359–2364, 2007.

[Sebastiani, 1994] R. Sebastiani. Applying GSAT to non-clausal formulas. *J. Artif. Intell. Res.*, 1:309–314, 1994.

[Stachniak and Belov, 2008] Z. Stachniak and A. Belov. Speeding-up non-clausal local search for propositional satisfiability with clause learning. In *Proc. SAT*, volume 4996 of *LNCS*, pages 257–270. Springer, 2008.

[Stachniak, 2002] Z. Stachniak. Going non-clausal. In *SAT*, 2002.

[Thiffault *et al.*, 2004] C. Thiffault, F. Bacchus, and T. Walsh. Solving non-clausal formulas with DPLL search. In *Proc. CP*, volume 3258 of *LNCS*, pages 663–678. Springer, 2004.

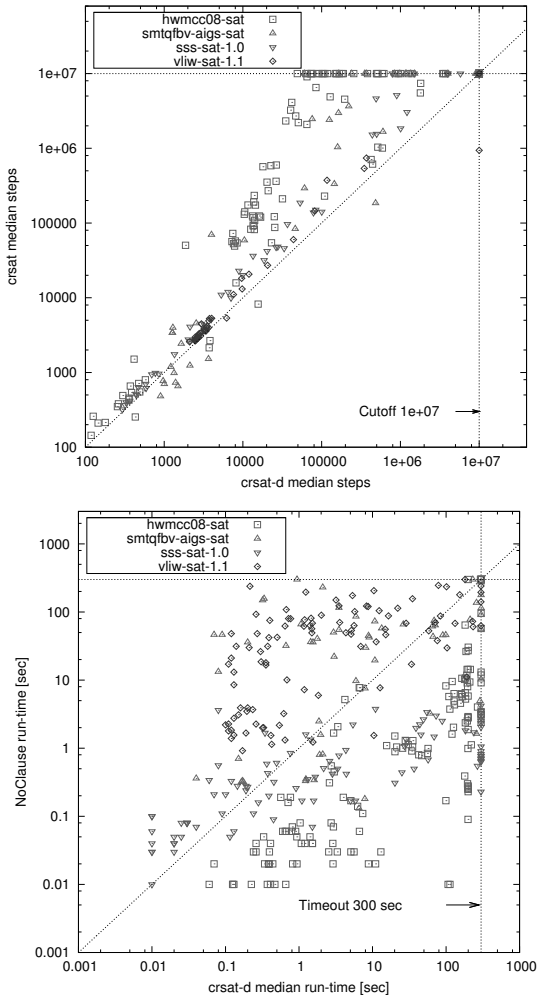


Figure 1: Upper: steps taken by crsat and d-crsat; lower: runtimes of d-crsat (median) and NoClause.