# Justification-Based Non-Clausal Local Search for SAT

**Matti Järvisalo** and **Tommi Junttila** and **Ilkka Niemelä**[1]

**Abstract.** While stochastic local search (SLS) techniques are very efficient in solving hard randomly generated propositional satisfiability (SAT) problem instances, a major challenge is to improve SLS on structured problems. Motivated by heuristics applied in complete circuit-level SAT solvers in electronic design automation, we develop novel SLS techniques by harnessing the concept of justification frontiers. This leads to SLS heuristics which concentrate the search into relevant parts of instances, exploit observability don't cares and allow for an early stopping criterion. Experiments with a prototype implementation of the framework presented in this paper show up to a four orders of magnitude decrease in the number of moves on real-world bounded model checking instances when compared to WalkSAT on the standard CNF encodings of the instances.

## 1 INTRODUCTION

Advances in propositional satisfiability (SAT) testing have established SAT based methods as a competitive way of solving combinatorial problems in various domains. Stochastic local search (SLS) methods, such as [16, 15, 10, 3], are very efficient especially in solving randomly generated SAT instances. However, for structural real-world SAT instances *complete* DPLL based SAT solvers seem to dominate SLS solvers (see, e.g., results of the latest SAT competitions at `http://www.satcompetition.org/`). Further work on improving SLS techniques for structural problems is needed and, in particular, developing techniques for handling variable dependencies efficiently has been identified a major challenge [7].

One problem in developing efficient techniques for handling variable dependencies is that typically the most efficient SLS solvers work on the flat CNF input format. Some techniques for CNF level SLS solvers have been developed to utilize propagation during search [2]. However, there seems to be room for novel structure-based SLS techniques exploiting variable dependencies more directly. Indeed, in SAT based approaches, direct CNF encodings of a problem domain are rarely used: the problem at hand is typically encoded with a structure-preserving general propositional formula $\phi$ which can then be translated into an equi-satisfiable CNF formula by introducing additional variables for the subformulas of $\phi$. There are also SAT solvers which—instead of demanding CNF translation before solving—work directly on general formulas. Such solvers use *Boolean circuits* [11] as the compact representation for a general propositional formula in a DAG-like structure. However, such solvers are typically complete DPLL style non-clausal algorithms [5, 8, 9, 17]. Only a few SLS methods have been proposed for general propositional formulas [14, 6, 12]. Common to these SLS approaches is that they attempt to explicitly exploit variable dependencies through *independent* (or *input*) *variables*, i.e., sets of variables such that a truth value assignment for them uniquely determines the truth values of all other variables, by focusing the search on truth assignments of input variables.

In this paper we develop a novel non-clausal SLS method for structural SAT problems from a different starting point. Our aim is to bring structure-exploiting techniques into local search for SAT in order to lift the performance of local search SAT solving especially on structural real-world problem domains. We employ Boolean circuits as the representation of general propositional formulas. Motivated by *justification frontier* heuristics (see e.g. [9]) applied in complete circuit-level SAT solvers in electronic design automation, our search technique looks for a *justification* for the Boolean circuit instead of focusing on finding a satisfying truth assignment. The idea is to be able to drive local search more top-down in the overall structure of the circuit rather than in a bottom-up mode as is done in local search techniques focusing on input variables. This is achieved by guiding the search using justification frontiers that enable exploiting *observability don't cares* (see e.g. [13]), drive the search to relevant parts of the circuit, and offer early stopping criteria which allow to end the search when the circuit is *de facto satisfiable* even if no concrete satisfying truth assignment has been found. Experiments with a prototype implementation of the framework presented in this paper show up to a four orders of magnitude decrease in the number of moves on real-world bounded model checking instances when compared to WalkSAT on the standard CNF encodings of the instances.

The rest of this paper is organized as follows. First, Boolean circuits and related central concepts are defined (Sect. 2). The proposed justification-based non-clausal SLS method is then described (Sect. 3) and analyzed w.r.t. both CNF level and previous non-clausal methods (Sect. 4). Initial experiments are presented in Sect. 5.

## 2 CONSTRAINED BOOLEAN CIRCUITS

Boolean circuits offer a natural non-clausal representation for propositional formulas in a compact DAG-like structure with *subformula sharing*. Rather than translating circuits to CNF for solving the resulting SAT instance by local search, in this work we will work directly on the Boolean circuit representation.

A Boolean circuit over a finite set $G$ of *gates* is a set $\mathcal{C}$ of equations of form $g := f(g_1, \ldots, g_n)$, where $g, g_1, \ldots, g_n \in G$ and $f : \{\mathbf{f}, \mathbf{t}\}^n \to \{\mathbf{f}, \mathbf{t}\}$ 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 $\mathcal{C}$, and (ii) the underlying directed graph

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

is acyclic. The set of gates in a circuit $\mathcal{C}$ is denoted by $G(\mathcal{C})$. If $\langle g', g \rangle \in E(\mathcal{C})$, then $g'$ is a *child* of $g$ and $g$ is a *parent* of $g'$. The

---

[1] Helsinki University of Technology, Dept. Information and Comp. Sci., Finland. Emails: {matti.jarvisalo,tommi.junttila,ilkka.niemela}@tkk.fi. Research supported by Academy of Finland (#122399 (MJ,IN) and #112016 (TJ)). MJ additionally acknowledges support from the HeCSE graduate school, Emil Aaltonen Foundation, Jenny and Antti Wihuri Foundation, Nokia Foundation, and Foundation for Technology Promotion TES.

*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 $\mathcal{C}$, then $g$ is an $f$-gate (or of type $f$), otherwise it is an *input gate*. The set of input gates in $\mathcal{C}$ is denoted by $\mathsf{inputs}(\mathcal{C})$. A gate with no parents is an *output gate*.

An assignment for $\mathcal{C}$ is a (possibly partial) function $\tau : G \to \{\mathbf{f}, \mathbf{t}\}$. A total assignment $\tau$ is *consistent with* $\mathcal{C}$ if $\tau(g) = f(\tau(g_1), \ldots, \tau(g_n))$ for each $g := f(g_1, \ldots, g_n)$ in $\mathcal{C}$.

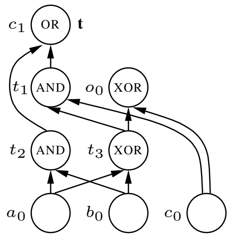
A *constrained Boolean circuit* $\mathcal{C}^\alpha$ is a pair $\langle \mathcal{C}, \alpha \rangle$, where $\mathcal{C}$ is a circuit and $\alpha$ is an assignment for $\mathcal{C}$. Each $\langle g, v \rangle \in \alpha$ is called a *constraint* where $g$ is *constrained* to $v$ (typically used for setting an output gate to a truth value). A total assignment $\tau$ for $\mathcal{C}$ *satisfies* $\mathcal{C}^\alpha$ if (i) $\tau$ is consistent with $\mathcal{C}$, and (ii) respects the constraints: $\tau \supseteq \alpha$. If some total assignment satisfies $\mathcal{C}^\alpha$, then $\mathcal{C}^\alpha$ is *satisfiable* and otherwise *unsatisfiable*. In this work we consider Boolean circuits in which the following Boolean functions are available as gate types.

- $\text{NOT}(v)$ is $\mathbf{t}$ iff $v$ is $\mathbf{f}$.
- $\text{OR}(v_1, \ldots, v_n)$ is $\mathbf{t}$ iff at least one of $v_1, \ldots, v_n$ is $\mathbf{t}$.
- $\text{AND}(v_1, \ldots, v_n)$ is $\mathbf{t}$ iff all $v_1, \ldots, v_n$ are $\mathbf{t}$.
- $\text{XOR}(v_1, v_2)$ is $\mathbf{t}$ iff exactly one of $v_1, v_2$ is $\mathbf{t}$.

However, notice that the techniques developed in this paper can be adapted for a wider range of types such as equivalence and cardinality gates. 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 by using the rules in [5].

**Example 1** *Figure 1 shows a Boolean circuit for a full-adder with the constraint that the carry-out bit $c_1$ is* $\mathbf{t}$. *One satisfying total assignment for the circuit is*

$$\{\langle c_1, \mathbf{t}\rangle, \langle t_1, \mathbf{t}\rangle, \langle o_0, \mathbf{f}\rangle, \langle t_2, \mathbf{f}\rangle, \langle t_3, \mathbf{t}\rangle, \langle a_0, \mathbf{t}\rangle, \langle b_0, \mathbf{f}\rangle, \langle c_0, \mathbf{t}\rangle\}. \quad (1)$$




$$\begin{aligned} \mathcal{C} \;=\; \{ c_1 &:= \text{OR}(t_1, t_2) \\ t_1 &:= \text{AND}(t_3, c_0) \\ o_0 &:= \text{XOR}(t_3, c_0) \\ t_2 &:= \text{AND}(a_0, b_0) \\ t_3 &:= \text{XOR}(a_0, b_0) \} \end{aligned}$$

$$\alpha = \{\langle c_1, \mathbf{t}\rangle\}$$

**Figure 1.** A constrained Boolean circuit $\mathcal{C}^\alpha$.

The *restriction* of an assignment $\tau$ 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 $v \in \{\mathbf{f}, \mathbf{t}\}$, a *justification* for the pair $\langle g, v \rangle$ is a partial assignment $\sigma : \{g_1, \ldots, g_n\} \to \{\mathbf{f}, \mathbf{t}\}$ to the children of $g$ such that $f(\tau(g_1), \ldots, \tau(g_n)) = v$ holds for all extensions $\tau \supset \sigma$. That is, the values assigned by $\sigma$ to the children of $g$ are enough to force $g$ to have the value $v$. A gate $g$ is *justified in an assignment* $\tau$ if it is assigned, i.e. $\tau(g)$ is defined, and (i) it is an input gate, or (ii) $g := f(g_1, \ldots, g_n) \in \mathcal{C}$ and $\tau|_{\{g_1, \ldots, g_n\}}$ is a justification for $\langle g, \tau(g)\rangle$. For example, consider the gate $t_1$ in Fig. 1. The possible justifications for $\langle t_1, \mathbf{f}\rangle$ are $\{\langle t_3, \mathbf{f}\rangle\}$, $\{\langle t_3, \mathbf{f}\rangle, \langle c_0, \mathbf{t}\rangle\}$, $\{\langle t_3, \mathbf{f}\rangle, \langle c_0, \mathbf{f}\rangle\}$, $\{\langle c_0, \mathbf{f}\rangle\}$, and $\{\langle t_3, \mathbf{t}\rangle, \langle c_0, \mathbf{f}\rangle\}$; the first and fourth are subset minimal ones. Gate $t_1$ is justified in the assignment (1).

Given a constrained circuit $\mathcal{C}^\alpha$ and an assignment $\tau \supseteq \alpha$ for $\mathcal{C}$, the *justification cone* of $\mathcal{C}^\alpha$ under $\tau$, denoted by $\mathsf{jcone}(\mathcal{C}^\alpha, \tau)$, is the minimal set of gates satisfying the following requirements.

1. All constrained gates belong to the cone. That is, if $\langle g, v\rangle \in \alpha$, then $g \in \mathsf{jcone}(\mathcal{C}^\alpha, \tau)$.
2. If a justified gate belongs to the cone, then all the gates that participate in some subset minimal justification for the gate are also in the cone. Formally, if $g \in \mathsf{jcone}(\mathcal{C}^\alpha, \tau)$ and (i) $g$ is a non-input gate, (ii) $g$ is justified in $\tau$, and (iii) $\langle g_i, v_i\rangle \in \sigma$ for some subset minimal justification $\sigma$ for $\langle g, \tau(g)\rangle$, then $g_i \in \mathsf{jcone}(\mathcal{C}^\alpha, \tau)$. In principle it would be sufficient to consider only one, arbitrarily chosen subset minimal justification. However, such a formalization would make $\mathsf{jcone}(\mathcal{C}^\alpha, \tau)$ ambiguously defined.

The *justification frontier* of $\mathcal{C}^\alpha$ under $\tau$ is the "bottom edge" of the justification cone, i.e. those gates in the cone that are not justified:

$$\mathsf{jfront}(\mathcal{C}^\alpha, \tau) = \{g \in \mathsf{jcone}(\mathcal{C}^\alpha, \tau) \mid g \text{ is not justified in } \tau\}.$$

A gate $g$ is *interesting* in $\tau$ if it belongs to the frontier $\mathsf{jfront}(\mathcal{C}^\alpha, \tau)$ or is a descendant of a gate in it; the set of all gates interesting in $\tau$ is denoted by $\mathsf{interest}(\mathcal{C}^\alpha, \tau)$. A gate $g$ is an (*observability*) *don't care* if it is neither interesting nor in the justification cone $\mathsf{jcone}(\mathcal{C}^\alpha, \tau)$. For instance, consider the constrained circuit $\mathcal{C}^\alpha$ in Fig. 1. Under the assignment $\tau = \{\langle c_1, \mathbf{t}\rangle, \langle t_1, \mathbf{t}\rangle, \langle o_0, \mathbf{f}\rangle, \langle t_2, \mathbf{f}\rangle, \langle t_3, \mathbf{t}\rangle, \langle a_0, \mathbf{f}\rangle, \langle b_0, \mathbf{f}\rangle, \langle c_0, \mathbf{t}\rangle\}$, the justification cone $\mathsf{jcone}(\mathcal{C}^\alpha, \tau)$ is $\{c_1, t_1, t_3, c_0\}$, the justification frontier $\mathsf{jfront}(\mathcal{C}^\alpha, \tau)$ is $\{t_3\}$, $\mathsf{interest}(\mathcal{C}^\alpha, \tau) = \{t_3, a_0, b_0\}$, and the gates $t_2$ and $o_0$ are don't cares.

**Proposition 1** *If the justification frontier* $\mathsf{jfront}(\mathcal{C}^\alpha, \tau)$ *is empty for some total assignment $\tau$, then the constrained circuit $\mathcal{C}^\alpha$ is satisfiable.*

When $\mathsf{jfront}(\mathcal{C}^\alpha, \tau)$ is empty, a satisfying assignment can be obtained by (i) restricting $\tau$ to the input gates appearing in the justification cone, i.e. to the gate set $\mathsf{jcone}(\mathcal{C}^\alpha, \tau) \cap \mathsf{inputs}(\mathcal{C})$, (ii) assigning other input gates arbitrary values, and (iii) recursively evaluating the values of non-input gates. Thus, whenever $\mathsf{jfront}(\mathcal{C}^\alpha, \tau)$ is empty, we say that $\tau$ *de facto satisfies* $\mathcal{C}^\alpha$. As an example, the assignment $\tau = \{\langle c_1, \mathbf{t}\rangle, \langle t_1, \mathbf{f}\rangle, \langle o_0, \mathbf{f}\rangle, \langle t_2, \mathbf{t}\rangle, \langle t_3, \mathbf{t}\rangle, \langle a_0, \mathbf{t}\rangle, \langle b_0, \mathbf{t}\rangle, \langle c_0, \mathbf{t}\rangle\}$ de facto satisfies the constrained circuit $\mathcal{C}^\alpha$ in Fig. 1. Also note that if a total truth assignment $\tau$ satisfies $\mathcal{C}^\alpha$, then $\mathsf{jfront}(\mathcal{C}^\alpha, \tau)$ is empty.

**Translating Circuits to CNF.** Each constrained Boolean circuit $\mathcal{C}^\alpha$ can be translated into an equi-satisfiable CNF formula $\mathsf{cnf}(\mathcal{C}^\alpha)$ by applying the standard "Tseitin translation". In order to obtain a small CNF formula, the idea is to introduce a variable $\tilde{g}$ for each gate $g$ in the circuit, and then to describe the functionality of each gate with a set of clauses. For instance, an AND-gate $g := \text{AND}(g_1, \ldots, g_n)$ is translated into the clauses $(\neg\tilde{g} \vee \tilde{g}_1)$,..., $(\neg\tilde{g} \vee \tilde{g}_n)$, and $(\tilde{g} \vee \neg\tilde{g}_1 \vee \ldots \vee \neg\tilde{g}_n)$. The constraints are translated into unit clauses: $\langle g, \mathbf{t}\rangle \in \alpha$ introduces the unit clause $(\tilde{g})$ and $\langle g, \mathbf{f}\rangle \in \alpha$ the negated unit clause $(\neg\tilde{g})$.

**A Note on Negations.** As usual in many SAT algorithms, we will implicitly ignore NOT-gates of form $g := \text{NOT}(g_1)$; $g$ and $g_1$ are always assumed to have the opposite values. Thus NOT-gates are, for instance, (i) "inlined" in the cnf translation by substituting $\neg\tilde{g}_1$ for $\tilde{g}$, and (ii) never counted in an interest set $\mathsf{interest}(\mathcal{C}^\alpha, \tau)$.

## 3 JUSTIFICATION-BASED NON-CLAUSAL SLS

In contrast to typical local search algorithms for SAT, which work on CNF formulas, we develop justification-based non-clausal stochastic local search techniques. As typical in clausal SLS, a configuration is described by a total truth assignment. However, our method works directly on general propositional formulas represented as Boolean circuits, and hence a configuration is a total assignment on the gates of the Boolean circuit at hand. In contrast to typical local search for SAT, we exploit—motivated by successful implementations of complete circuit SAT solving techniques (see, e.g., [9])–techniques for detecting *justification-based don't cares* within our Boolean circuit SAT local search (BC SLS) framework. This is based on justification frontiers, which guide the search heuristics to concentrate on relevant parts of the instance and, moreover, provide an alternative, early stopping criterion for the search.

We demonstrate the novel approach by developing a *WalkSAT type* algorithm [15] that exploits justification frontiers in guiding search. In the clausal WalkSAT local moves are based on randomly selecting a clause falsified by the current truth assignment. In our algorithm the role of the falsified clauses is played by the gates in the justification front, i.e., the gates in the justification cone not justified by the current assignment. WalkSAT flips one of the variables in the chosen clause in the greedy move to maximize the decrease in the number of falsified clauses. In our case a greedy move selects a justification for the chosen gate to minimize the number of interesting gates.

The resulting method is presented as Algorithm 1. Given a constraint circuit $\mathcal{C}^\alpha$ and a noise parameter $p \in [0, 1]$ (with $p = 0$ only greedy moves are made), the algorithm performs local search over the assignment space of *all* the gates in $\mathcal{C}$ (inner loop on lines 3-13).

**Algorithm 1** BC SLS

**Input:** constrained Boolean circuit $\mathcal{C}^\alpha$, parameter $p \in [0, 1]$
**Output:** a *de facto* satisfying assignment for $\mathcal{C}^\alpha$ or "*don't know*"
**Explanations:**
$\tau$: current truth assignment on all gates with $\tau \supseteq \alpha$
$\delta$: next move (a partial assignment)

1: **for** $try := 1$ to $\textsc{MaxTries}(\mathcal{C}^\alpha)$ **do**
2: $\quad\tau :=$ pick an assignment over all gates in $\mathcal{C}$ s.t. $\tau \supseteq \alpha$
3: $\quad$**for** $move := 1$ to $\textsc{MaxMoves}(\mathcal{C}^\alpha)$ **do**
4: $\quad\quad$**if** $\mathsf{jfront}(\mathcal{C}^\alpha, \tau) = \emptyset$ **then return** $\tau$
5: $\quad\quad$Select a random gate $g \in \mathsf{jfront}(\mathcal{C}^\alpha, \tau)$
6: $\quad\quad$**with** probability $(1 - p)$ **do** *%greedy move*
7: $\quad\quad\quad\delta :=$ a random justification from those justifications for $\langle g, v\rangle \in \tau$ that minimize $\mathsf{cost}(\tau, \cdot)$
8: $\quad\quad$**otherwise** *%non-greedy move (with probability p)*
9: $\quad\quad\quad$**if** $g$ is unconstrained in $\alpha$
10: $\quad\quad\quad\quad\delta := \{\langle g, \neg v\rangle\}$ where $\langle g, v\rangle \in \tau$
11: $\quad\quad\quad$**else**
12: $\quad\quad\quad\quad\delta :=$ a random justification for $\langle g, v\rangle \in \tau$
13: $\quad\quad\tau := (\tau \setminus \{\langle g, \neg w\rangle \mid \langle g, w\rangle \in \delta\}) \cup \delta$
14: **return** "*don't know*"

We will next describe the inner loop of BC SLS in more detail.

### 3.1 Stopping Criterion

Similar to typical CNF level SLS methods, one could terminate the search in BC SLS by applying the *standard* stopping criterion: when all gates are justified in the current configuration $\tau$, then $\tau$ is in itself a satisfying truth assignment for the circuit. However, the justification frontier allows for an early stopping criterion by Proposition 1: when the current front $\mathsf{jfront}(\mathcal{C}^\alpha, \tau)$ is empty (line 4), the current configuration $\tau$ *de facto* satisfies $\mathcal{C}^\alpha$. Thus we can obtain from $\tau$ a satisfying assignment after the search is terminated by simply evaluating the unconstrained gates in $\mathcal{C}^\alpha$ by using the values for input gates in $\tau$. This is a *stronger* stopping criterion than the standard one, since the front is empty whenever the standard one holds, but the opposite does not necessarily hold: the front can be empty even if there are gates in the circuit which are not justified in $\tau$.

### 3.2 Making Moves

For each of the $\textsc{MaxTries}(\mathcal{C}^\alpha)$ runs of the inner loop, $\textsc{MaxMoves}(\mathcal{C}^\alpha)$ moves are made. The moves exploit structural information and semantics of individual gates for finding a *justification* for the currently assigned value of a chosen gate (lines 6-12).

Given the current configuration $\tau$, we concentrate on making moves on gates in $\mathsf{jfront}(\mathcal{C}^\alpha, \tau)$ by randomly picking a gate $g$ from this set. For a gate $g$ and its current value $v$ in $\tau$, the possible *greedy moves* are induced by the justifications for $\langle g, v\rangle$. The idea is to minimize the *size of the interest set*. In other words, the value of the cost function for a move (justification) $\delta$ is

$$\mathsf{cost}(\tau, \delta) = \left|\mathsf{interest}(\mathcal{C}^\alpha, \tau')\right|,$$

where $\tau' = (\tau \setminus \{\langle g, \neg w\rangle \mid \langle g, w\rangle \in \delta\}) \cup \delta$. That is, the cost of a move $\delta$ is given by the size of the interest set in the configuration $\tau'$ where for the gates mentioned in $\delta$ we use the values in $\delta$ instead of those in $\tau$. The move is then selected randomly from those justifications $\delta$ for $\langle g, v\rangle$ for which the value $\mathsf{cost}(\tau, \delta)$ is smallest over all justifications for $\langle g, v\rangle$.

During a *non-greedy move* (lines 9-12, executed with probability $p$), we invert the value of *the gate $g$ itself* whenever this is possible, i.e., when $g$ is not constrained in $\alpha$. The idea here is to try to escape from possible local minima by more radically changing the justification front, most likely upwards in the circuit structure. In the case that we may not invert the value of $g$ (since it is constrained), the move is chosen randomly from the set of all justifications for $\langle g, v\rangle \in \tau$.

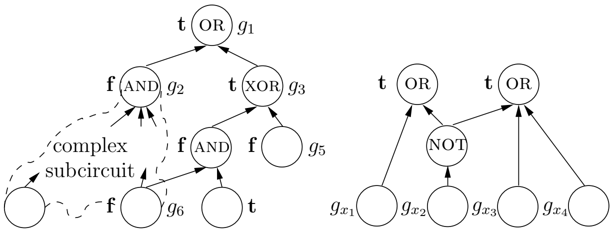
## 4 ANALYSIS

### 4.1 Interest Set Size Driven Greedy Moves

Considering greedy moves, the objective function under minimization in BC SLS is $\mathsf{cost}(\tau, \cdot)$. Alternatively, one could use the objective of minimizing $|\mathsf{jfront}(\mathcal{C}^\alpha, \tau')|$, since (i) flipping is concentrated on gates in $\mathsf{jfront}(\mathcal{C}^\alpha, \tau)$ and (ii) the stopping criterion $\mathsf{jfront}(\mathcal{C}^\alpha, \tau) = \emptyset$ is used. The reasoning behind choosing to minimize the number of gates in $\mathsf{interest}(\mathcal{C}^\alpha, \tau')$ is that it gives a *better progress measure* than minimizing the number of gates in the justification front. First, notice that the justification front cannot become empty before it reaches a subset of the input gates, since only input gates are justified by default. Now, the size of the interest set gives an upper bound on the number of gates that still need to be justified (the descendants of the gates in the front). Following this intuition, by minimizing the size of the interest set the greedy moves drive the search towards the input gates.

### 4.2 Comparison with Clausal Methods

One of the main advantages of the proposed BC SLS method over clausal local search methods is that BC SLS can exploit observability don't cares. As an example, consider the circuit in Fig. 2(a), where the gate $g_1$ is constrained to true and the other **t** and **f** symbols depict the current configuration $\tau$. All the gates, except $g_6$, in the complex subcircuit rooted at the gate $g_2$ are don't cares under $\tau$. Therefore BC SLS can ignore the subcircuit and terminate after flipping the input gate $g_5$ as the justification front becomes empty. However, assume that we translate the circuit into a CNF formula by using the Tseitin translation $\mathsf{cnf}$ given in Sect. 2. If we apply a clausal SLS algorithm such as WalkSAT on the CNF formula, observability don't cares are no longer available in the sense that the algorithm must find a total truth assignment that simultaneously satisfies all the clauses originating from the subcircuit. This can be a very complex task.




(a) Exploiting don't cares. (b) A CNF circuit.

**Figure 2.** Example circuits

We can also analyze how BC SLS behaves on flat clausal input. To do this, we associate a CNF formula $F = C_1 \wedge \ldots \wedge C_k$ with a *constrained CNF circuit* $\mathsf{ccirc}(F) = \langle \mathcal{C}, \alpha \rangle$ as follows. Take an input gate $g_x$ for each variable $x$ occurring in $F$. Now $\mathcal{C} = \{g_{C_i} := \text{OR}(g_{l_1}, ..., g_{l_m}) \mid C_i = (l_1 \vee \ldots \vee l_m)\} \cup \left\{g_{\neg x} := \text{NOT}(g_x) \mid \neg x \in \cup_{i=1}^{k} C_i\right\}$ and the constraints force each "clause gate" $g_{C_i}$ to true: $\alpha = \{\langle g_{C_i}, \mathbf{t}\rangle \mid 1 \leq i \leq k\}$. This is illustrated in Fig. 2(b) for $F = (x_1 \vee \neg x_2) \wedge (\neg x_2 \vee x_3 \vee x_4)$.

When BC SLS is run on a CNF circuit, it can only flip input variables. If input gates were excluded from the set $\mathsf{interest}(\mathcal{C}^\alpha, \tau)$ of interesting gates, then $|\mathsf{interest}(\mathcal{C}^\alpha, \tau)|$ would equal to the number of unjustified clause gates in the configuration $\tau$. Thus the greedy move cost function $\mathsf{cost}(\tau, \cdot)$ would equal to that applied in WalkSAT measuring the number of clauses fixed/broken during a flip. Since input gates are included in $\mathsf{interest}(\mathcal{C}^\alpha, \tau)$, the BC SLS cost function also measures, in CNF terms, the number of variables occurring in unsatisfied clauses.

### 4.3 Comparison with Non-Clausal Methods

SLS techniques working directly on non-clausal problems closest to our work include [14, 6, 12]. They are all based on the idea of limiting flipping to input (independent) variables whereas we allow flipping all gates (subformulas) of the problem instance. Moreover, in these approaches the greedy part of the search is driven by a cost function which is substantially different from the justification-based cost function that we employ. Sebastiani [14] generalizes the GSAT heuristic to general propositional formulas and defines the cost function by (implicitly) considering the CNF form *cnf*($\phi$) of the general formula $\phi$: the cost for a truth assignment is the number of clauses in *cnf*($\phi$) falsified by the assignment. The approaches of Kautz and Selman [6] and Pham et al. [12] both use a Boolean circuit representation of the problem and employ a cost function which, given a truth assignment for the input gates, counts the number of constrained output gates falsified by the assignment. This cost function provides limited guidance to greedy moves in cases where there are few constrained output gates or they are far from the input gates. A worst-case scenario occurs when the Boolean circuit given as input has *a single output gate* implying that the cost function can only have the values 0 or 1 for any flip under any configuration. Such a cost function does not offer much direction for the greedy flips towards a satisfying truth assignment. Our cost function appears to be less sensitive to the number of output gates or their distance from the input gates. This is because the search is based on the concept of a justification frontier which is able to distribute the requirements implied by the constrained output gates deeper in the circuit.

## 5 EXPERIMENTS

In order to evaluate the ideas behind the BC SLS framework, we have implemented a prototype on top of the bc2cnf Boolean circuit simplifier/CNF translator [4]. The computation of justification cone is implemented directly by the definition. When making greedy and random moves, justifications are selected from the set of subset minimal justifications for the gate value; for a true OR-gate and false AND-gate, the value of a single child is inverted, and for a true OR-gate and false AND-gate the values of all children are inverted. As structural benchmarks we use a set of Boolean circuits encoding bounded model checking of asynchronous systems for deadlocks [1], available at `http://www.tcs.hut.fi/~mjj/benchmarks/`. Although rather easy for current DPLL solvers, these benchmarks are challenging for typical SLS methods.

Since our implementation is at present a very preliminary *non-incremental* one, we will compare the number of moves made by WalkSAT and our prototype.[2] We use WalkSAT, since the current prototype—as explained also in Sect. 3—can be basically seen as a justification-based variation of WalkSAT. For running WalkSAT, we apply exactly the same Boolean circuit level simplification in bc2cnf to the circuits as in our prototype (including, e.g., circuit level propagation that is equivalent to unit propagation), and then translate the simplified circuit to CNF with the Tseitin-style translation implemented in bc2cnf for running WalkSAT. We run both WalkSAT and our prototype implementation with the default noise value $p = 0.5$ (that is, 50%). To make a fair evaluation (not favoring our prototype), we allow WalkSAT $10^8$ moves and limit our implementation to a maximum of $10^6$ moves. Each instance is run 9 times without restarts. The number of gates in the simplified circuits (column **#gates**), and the number of variables (**#vars**) and clauses (**#clauses**) resulting from the standard CNF translation, are given in Table 1. Furthermore, the minimum (**min**), median (**med**), and maximum (**max**) number of moves for each instance is presented. The number of runs without a satisfying truth assignment is given in the column **max** in parentheses. Additionally, we give the ratio of the number of moves made by our prototype and WalkSAT for the minimum, median, and maximum number of moves done by the solvers. For example, the **max/max** ratio of 533.43 for the instance speed_1.fsa-b10-s means that the maximum number of moves made by WalkSAT over the nine runs was 533.43 times as large as the maximum number of moves done by our implementation on the instance.

To sum up, the experiments demonstrate potential of our novel approach when solving structural (non-clausal) SAT instances. A promising observation is that our justification frontier based technique seems to keep the search rather focused when the size of the instance grows as witnessed by the modestly increasing number of moves. In particular, this compares favorably to WalkSAT which typically exceeds the cutoff of $10^8$ moves as the instance sizes grow.

[2] The prototype computes the justification front and cone repeatedly in a global, non-incremental way. This naive implementation makes around 80-250 times fewer flips per second (fps) than WalkSAT on instances with 1000-2500 gates. By careful re-implementation a very substantial increase is expected in the fps rate by incrementally computing the front and cone.

**Table 1.** Comparison of a prototype implementation of BC SLS with WalkSAT

| | CNF | | | BC SLS #moves | | | WalkSAT #moves | | | relative gain in #moves | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Instance** | **#gates** | **#vars** | **#clauses** | **min** | **med** | **max** | **min** | **med** | **max** | **min/min** | **med/med** | **max/max** |
| speed_1.fsa-b6-s | 836 | 688 | 2087 | 965 | 965 | 4358 | 2252 | 5805 | 11368 | 2.33 | 6.02 | 2.61 |
| speed_1.fsa-b7-s | 1142 | 943 | 2875 | 2266 | 2578 | 5077 | 6255 | 20915 | 38237 | 2.76 | 8.11 | 7.53 |
| speed_1.fsa-b8-s | 1448 | 1198 | 3660 | 1633 | 1849 | 5518 | 9266 | 62497 | 95837 | 5.67 | 33.80 | 17.37 |
| speed_1.fsa-b9-s | 1754 | 1453 | 4444 | 5029 | 6695 | 12616 | 25911 | 330321 | 1643032 | 5.15 | 49.34 | 130.23 |
| speed_1.fsa-b10-s | 2060 | 1708 | 5226 | 5089 | 11313 | 28423 | 1511045 | 4376285 | 15161778 | 296.92 | 386.84 | 533.43 |
| speed_1.fsa-b12-s | 2672 | 2218 | 6786 | 6899 | 41379 | 141700 | - | - | - (9) | >14494.85 | >2416.68 | >705.72 |
| speed_1.fsa-b13-s | 2978 | 2473 | 7563 | 31384 | 139921 | 415601 | - | - | - (9) | >3186.34 | >714.69 | >240.62 |
| speed_1.fsa-b14-s | 3284 | 2728 | 8338 | 43690 | 179967 | 587184 | - | - | - (9) | >2288.85 | >555.66 | >170.30 |
| speed_1.fsa-b15-s | 3590 | 2983 | 9111 | 33647 | 321554 | - (1) | - | - | - (9) | >2972.03 | >310.99 | - |
| speed_1.fsa-b6-p | 687 | 577 | 1666 | 1129 | 1129 | 1129 | 1342 | 1851 | 7706 | 1.19 | 1.64 | 6.83 |
| speed_1.fsa-b7-p | 1022 | 863 | 2528 | 2148 | 2777 | 7614 | 4636 | 10916 | 25955 | 2.16 | 3.93 | 3.41 |
| speed_1.fsa-b8-p | 1359 | 1149 | 3387 | 4338 | 8248 | 22294 | 10991 | 40833 | 278042 | 2.53 | 4.95 | 12.47 |
| speed_1.fsa-b9-p | 1696 | 1435 | 4245 | 5176 | 10610 | 27500 | 33752 | 76864 | 288506 | 6.52 | 7.24 | 10.49 |
| speed_1.fsa-b10-p | 2033 | 1721 | 5101 | 7249 | 30846 | 60009 | 2043613 | 4638369 | 10800631 | 281.92 | 150.37 | 179.98 |
| speed_1.fsa-b12-p | 2703 | 2289 | 6793 | 45304 | 144787 | 735228 | - | - | - (9) | >2207.31 | >690.67 | >136.01 |
| speed_1.fsa-b13-p | 3040 | 2575 | 7643 | 34363 | 328346 | 709696 | - | - | - (9) | >2910.11 | >304.56 | >140.91 |
| dp_12.fsa-b5-s | 1579 | 1339 | 4148 | 8880 | 27519 | 36421 | 14469 | 37361 | 81102 | 1.63 | 1.36 | 2.23 |
| dp_12.fsa-b6-s | 2060 | 1748 | 5418 | 23740 | 52975 | 106542 | 123249 | 790190 | 2299552 | 5.19 | 14.92 | 21.58 |
| dp_12.fsa-b7-s | 2541 | 2157 | 6688 | 28289 | 69029 | 170824 | 397887 | 28360757 | - (1) | 14.07 | 410.85 | >585.40 |
| dp_12.fsa-b8-s | 3022 | 2566 | 7958 | 33935 | 91764 | 461459 | - | - | - (9) | >2946.81 | >1089.75 | >216.70 |
| dp_12.fsa-b9-s | 3503 | 2975 | 9228 | 83107 | 162137 | 446453 | - | - | - (9) | >1203.27 | >616.76 | >223.99 |
| dp_12.fsa-b5-p | 1267 | 1111 | 3210 | 4838 | 85808 | 411793 | 4619 | 12545 | 46037 | 0.95 | 0.15 | 0.11 |
| dp_12.fsa-b6-p | 1844 | 1616 | 4673 | 38563 | 118477 | 221461 | 17961 | 85344 | 145830 | 0.47 | 0.72 | 0.66 |
| dp_12.fsa-b7-p | 2421 | 2121 | 6136 | 22545 | 69040 | 214360 | 113932 | 244863 | 406876 | 5.05 | 3.55 | 1.90 |
| dp_12.fsa-b8-p | 2998 | 2626 | 7599 | 73826 | 132431 | 576672 | 379112 | 14101664 | 69496990 | 5.14 | 106.48 | 120.51 |
| dp_12.fsa-b9-p | 3575 | 3131 | 9062 | 50227 | 148409 | 425594 | - | - | - (9) | >1990.96 | >673.81 | >234.97 |
| elevator_1-b4-p | 264 | 230 | 649 | 171 | 171 | 171 | 1176 | 3041 | 10801 | 6.88 | 17.78 | 63.16 |
| elevator_1-b4-s | 534 | 447 | 1363 | 869 | 869 | 1723 | 2450 | 51226 | 270317 | 2.82 | 58.95 | 156.89 |
| elevator_1-b5-s | 841 | 704 | 2163 | 2543 | 3632 | 4788 | 19472 | 202139 | 391216 | 7.66 | 55.66 | 81.71 |
| elevator_1-b6-s | 1307 | 1093 | 3388 | 5073 | 57305 | 116572 | 305888 | 1317650 | 4433915 | 60.30 | 22.99 | 38.04 |
| elevator_2-b6-p | 896 | 775 | 2308 | 1789 | 4376 | 15621 | 16898 | 1134779 | 2824590 | 9.45 | 259.32 | 180.82 |
| elevator_2-b7-p | 1606 | 1379 | 4214 | 6221 | 18601 | 91691 | 492869 | 2756750 | 13232933 | 79.23 | 148.20 | 144.32 |
| elevator_2-b6-s | 1582 | 1339 | 4157 | 6776 | 16702 | - (1) | 13576544 | - | - (8) | 2003.62 | >5987.31 | - |
| elevator_2-b7-s | 2448 | 2070 | 6495 | 7940 | 28524 | 72247 | - | - | - (9) | >12594.46 | >3505.82 | >1384.14 |
| mmgt_2.fsa-b6-p | 903 | 777 | 2285 | 1220 | 1220 | 34132 | 170454 | 620821 | 1260101 | 139.72 | 508.87 | 36.92 |
| mmgt_2.fsa-b7-p | 1283 | 1113 | 3278 | 5671 | 32236 | 86944 | 873901 | 4289501 | 16896746 | 154.10 | 133.07 | 194.34 |
| mmgt_2.fsa-b6-s | 1421 | 1188 | 3722 | 3051 | 10831 | 38780 | 8586379 | 67686412 | - (3) | 2814.28 | 6249.32 | >2578.65 |
| mmgt_3.fsa-b7-p | 1953 | 1692 | 5034 | 5136 | 7264 | 48315 | 6886854 | - | - (6) | 1340.90 | >13766.52 | >2069.75 |
| mmgt_3.fsa-b7-s | 3079 | 2600 | 8260 | 39796 | 178191 | 833128 | - | - | - (9) | >2512.82 | >561.20 | >120.03 |

Considering the input flipping SLS methods in the literature (recall Sect. 4.3), we were, unfortunately, unable at the moment to obtain implementations of these methods for comparison. Comparing input flipping methods to our current framework remains thus an important aspect of future work. We did also investigate the performance of AdaptNovelty+ [3] on the benchmarks. We omit the precise results here due to space reasons. On the whole, although AdaptNovelty+ does find satisfying truth assignments for more instances than WalkSAT using the cutoff of $10^8$ moves, our prototype shows typically a one-to-three orders of magnitude reduction in the number of moves compared to AdaptNovelty+ — rather similarly as when compared to WalkSAT.

## 6 CONCLUSIONS

Motivated by techniques applied in circuit-level SAT solvers in electronic design automation, we present a novel approach to solving structural SAT problems with local search on non-clausal level. By incorporating justification frontiers, we develop SLS heuristics which concentrate the search into relevant parts of instances, exploit observability don't cares and allow for an early stopping criterion. Encouraged by the potential witnessed by low move counts of a prototype implementation, we see various directions for further work. We plan to replace the prototype with a proper solver implementation with specialized data structures. For achieving self-tuning of the greediness parameter for effectively escaping from local minima, developing adaptive noise mechanisms [3] for non-clausal SLS is a topic for further work. Another aspect is to investigate the effect of adding local consistency checking (on the circuit level, extending studies on adding propagation to CNF level SLS [2]) into the framework, and possibly even conflict learning.

## REFERENCES


[1] K. Heljanko, 'Bounded reachability checking with process semantics', in *CONCUR*, volume 2154 of *LNCS*, pp. 218–232. Springer, (2001).

[2] E.A. Hirsch and A. Kojevnikov, 'UnitWalk: A new SAT solver that uses local search guided by unit clause elimination', *Annals of Mathematics and Artificial Intelligence*, **43**(1), 91–111, (2005).

[3] H.H. Hoos, 'An adaptive noise mechanism for WalkSAT', in *AAAI*, pp. 655–660, (2002).

[4] T. Junttila. The BC package and a file format for constrained Boolean circuits˙ `http://www.tcs.hut.fi/~tjunttil/bcsat/`.

[5] T. Junttila and I. Niemelä, 'Towards an efficient tableau method for Boolean circuit satisfiability checking', in *CL 2000*, volume 1861 of *LNAI*, pp. 553–567. Springer, (2000).

[6] H. Kautz, D. McAllester, and B. Selman, 'Exploiting variable dependency in local search', in *IJCAI poster session*, (1997). `http://www.cs.rochester.edu/u/kautz/papers/dagsat.ps`.

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

[8] A. Kuehlmann, M.K. Ganai, and V. Paruthi, 'Circuit-based Boolean reasoning', in *DAC*, pp. 232–237. ACM, (2001).

[9] A. Kuehlmann, V. Paruthi, F. Krohm, and M. K. Ganai, 'Robust Boolean reasoning for equivalence checking and functional property verification', *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, **21**(12), 1377–1394, (2002).

[10] D. McAllester, B. Selman, and H. Kautz, 'Evidence for invariants in local search', in *AAAI*, pp. 321–326, (1997).

[11] C. Papadimitriou, *Computational Complexity*, Addison-Wesley, 1995.

[12] D.N. Pham, J. Thornton, and A. Sattar, 'Building structure into local search for SAT', in *IJCAI*, pp. 2359–2364, (2007).

[13] S. Safarpour, A. Veneris, R. Drechsler, and J. Lee, 'Managing don't cares in Boolean satisfiability', in *DATE'04*. IEEE, (2004).

[14] R. Sebastiani, 'Applying GSAT to non-clausal formulas', *Journal of Artificial Intelligence Research*, **1**, 309–314, (1994).

[15] B. Selman, H.A. Kautz, and B. Cohen, 'Noise strategies for improving local search', in *AAAI*, pp. 337–343, (1994).

[16] B. Selman, H. Levesque, and D. Mitchell, 'A new method for solving hard satisfiability problems', in *AAAI*, pp. 440–446, (1992).

[17] C. Thiffault, F. Bacchus, and T. Walsh, 'Solving non-clausal formulas with DPLL search', in *CP*, volume 3258 of *LNCS*, pp. 663–678. Springer, (2004).