Justification-Based Local Search with Adaptive Noise Strategies

Matti Järvisalo, Tommi Junttila, and Ilkka Niemelä

Helsinki University of Technology (TKK)
Department of Information and Computer Science, P.O. Box 5400, FI-02015 TKK, Finland
{matti.jarvisalo,tommi.junttila,ilkka.niemela}@tkk.fi

Abstract. We study a framework called BC SLS for a novel type of stochastic local search (SLS) for propositional satisfiability (SAT). Aimed specifically at solving real-world SAT instances, the approach works directly on a non-clausal structural representation for SAT. This allows for don't care detection and justification guided search heuristics in SLS by applying the circuit-level SAT technique of justification frontiers. In this paper we extend the BC SLS approach first by developing generalizations of BC SLS which are probabilistically approximately complete (PAC). Second, we develop and study adaptive noise mechanisms for BC SLS, including mechanisms based on dynamically adapting the waiting period for noise increases. Experiments show that a preliminary implementation of the novel adaptive, PAC generalization of the method outperforms a well-known CNF level SLS method with adaptive noise (AdaptNovelty+) on a collection of structured real-world SAT instances.

1 Introduction

While stochastic local search techniques (SLS) such as [1–5] are very efficient in solving hard randomly generated SAT instances, a major challenge is to improve SLS on structural problems by efficiently handling variable dependencies [6]. In this paper we extend a recent non-clausal stochastic local search (SLS) method, BC SLS [7], which applies similar ideas as typical in clausal SLS methods but differs in many crucial aspects. In particular, BC SLS combines techniques from structure-based complete DPLL style non-clausal algorithms [8–11]. Aimed specifically at solving real-world SAT instances, BC SLS works directly on a non-clausal structural representation for SAT. This allows for adopting don't cares [12] and justification guided search heuristics in SLS by applying ideas from the circuit-level SAT technique of justification frontiers [10]. For a discussion of the relationship between the basic BC SLS method and both CNF level and other non-clausal SLS methods, such as [13–15], see [7].

In this work we adopt the basic ingredients of local search—the notions of a configuration and a move, the objective function, and the stopping criterion—from BC SLS, and extend the approach. In more detail, we develop generalizations of BC SLS which (i) are *probabilistically approximately complete* (PAC) [16], and which (ii) *exploit adaptive noise mechanisms* within the framework.

It has been observed that the performance of CNF level SLS methods, such as those in the WalkSAT family, varies greatly depending on the chosen fixed noise parameter setting [3, 4]. We show that the same phenomenon is present also in BC SLS. In the case of CNF level SLS, in order to avoid manual noise tuning this has led to the development of automatic noise level mechanisms based on probing techniques for selecting a fixed noise parameter setting before actual search [17], or by adaptively tuning the noise level during search [4]. Here we adapt latter techniques to the BC SLS framework. However, we discover that compared to the parameter values for adapting noise used in CNF level SLS methods, radically different settings are required in BC SLS. We then show how to adjust this technique for BC SLS for better performance. In addition to the adaptive noise mechanism based on a static waiting period for noise increments, we suggest an alternative based on dynamic waiting periods that depend more on the current state of the search. While maintaining similar performance, the application of dynamic waiting periods gives the possibility of dismissing the fixed constant used in the typical adaptive noise mechanism based on a static waiting periods.

Applying a novel adaptive noise strategy for BC SLS, we show experimentally that a preliminary implementation of an adaptive PAC variant of the BC SLS method outperforms a fine-tuned implementation of the CNF level SLS method AdaptNovelty+ on a collection of structured real-world SAT instances.

This paper is organized as follows. First we define Boolean circuits and central concepts related to justifications and don't cares (Sect. 2). The justification-based nonclausal SLS framework is described in Sect. 3, with analysis of probabilistically approximately completeness of different variants of the method (Sect. 3.1). Section 4 is focused on developing adaptive noise mechanisms for the framework.

2 Constrained Boolean Circuits

Boolean circuits offer a natural non-clausal representation for *arbitrary* 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

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

is acyclic. If $\langle g', g\rangle \in E(\mathcal{C})$, then g' is a *child* of g and g is a *parent* of g'. 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 $\mathcal{C}$, 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 τ for $\mathcal{C}$ is *consistent* if $\tau(g) = f(\tau(g_1), \ldots, \tau(g_n))$ for each $g := f(g_1, \ldots, g_n)$ in $\mathcal{C}$. A circuit $\mathcal{C}$ has $2^{|\mathsf{inputs}(\mathcal{C})|}$ consistent total assignments.

A *constrained Boolean circuit* $\mathcal{C}^\alpha$ is a pair $\langle \mathcal{C}, \alpha\rangle$, where $\mathcal{C}$ is a Boolean circuit and α is an assignment for $\mathcal{C}$. With respect to a constrained circuit $\mathcal{C}^\alpha$, each $\langle g, v\rangle \in \alpha$ is a *constraint*, and g is *constrained* to v if $\langle g, v\rangle \in \alpha$. A total assignment τ for $\mathcal{C}$ *satisfies* $\mathcal{C}^\alpha$ if (i) τ 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.

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

However, notice that the techniques developed 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 by using the rules in [8].

Figure 1 shows a Boolean circuit for a full-adder with the constraint that the carry-out bit c_1 is **t**. Formally the circuit is defined as $\mathcal{C} = \{c_1{:=}\mathrm{OR}(t_1, t_2), t_1{:=}\mathrm{AND}(t_3, c_0), o_0{:=}\mathrm{XOR}(t_3, c_0), t_2{:=}\mathrm{AND}(a_0, b_0), t_3{:=}\mathrm{XOR}(a_0, b_0)\}$, and $\alpha = \{\langle c_1, \mathbf{t}\rangle\}$. A satisfying total assignment for it is $\{\langle c_1, \mathbf{t}\rangle, \langle t_1, \mathbf{t}\rangle, \langle o_0, \mathbf{f}\rangle, \langle t_2, \mathbf{f}\rangle, \langle t_3, \mathbf{t}\rangle, \langle a_0, \mathbf{t}\rangle, \langle b_0, \mathbf{f}\rangle, \langle c_0, \mathbf{t}\rangle\}$.

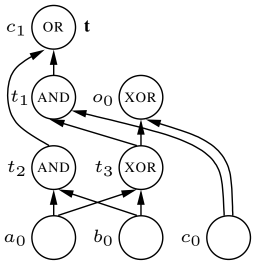

Fig. 1: A constrained circuit

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 $v \in \{\mathbf{f}, \mathbf{t}\}$, 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 $\tau \supseteq \sigma$. That is, the values assigned by σ to the children of g are enough to force g to have the 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) $g := f(g_1, \ldots, g_n) \in \mathcal{C}$ 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 $\{\langle t_3, \mathbf{t}\rangle, \langle c_0, \mathbf{f}\rangle\}$; of these the first and fourth one are subset minimal ones. The gate t_1 is justified in the assignment $\tau = \{\langle c_1, \mathbf{t}\rangle, \langle t_1, \mathbf{f}\rangle, \langle o_0, \mathbf{t}\rangle, \langle t_2, \mathbf{t}\rangle, \langle t_3, \mathbf{f}\rangle, \langle a_0, \mathbf{t}\rangle, \langle b_0, \mathbf{t}\rangle, \langle c_0, \mathbf{t}\rangle\}$.

A key concept in BC SLS is the *justification cone* $\mathsf{jcone}(\mathcal{C}^\alpha, \tau)$ for a constrained circuit $\mathcal{C}^\alpha$ under an assignment $\tau \supseteq \alpha$. The justification cone is defined recursively top-down in the circuit structure, starting from the constrained gates. Intuitively, the cone is the smallest set of gates which includes all constrained gates and, for each justified gate in the set, all the gates that participate in some subset minimal justification for the gate. More formally, $\mathsf{jcone}(\mathcal{C}^\alpha, \tau)$ is the smallest one of those 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 $\mathsf{jcone}(\mathcal{C}^\alpha, \tau)$ is unambiguously defined.

As another key concept, the *justification frontier* of $\mathcal{C}^\alpha$ under τ, 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 τ 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 that are interesting in τ 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.

As observed in [7] if the justification frontier $\mathsf{jfront}(\mathcal{C}^\alpha, \tau)$ is empty for some total assignment τ, 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 τ 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 τ *de facto satisfies* $\mathcal{C}^\alpha$. For example, the assignment $\{\langle c_1, \mathbf{t}\rangle, \langle t_1, \mathbf{f}\rangle, \langle o_0, \mathbf{f}\rangle, \langle t_2, \mathbf{t}\rangle, \langle t_3, \mathbf{t}\rangle, \langle a_0, \mathbf{t}\rangle, \langle b_0, \mathbf{t}\rangle, \langle c_0, \mathbf{t}\rangle\}$ de facto satisfies the constrained circuit $\mathcal{C}^\alpha$ in Fig. 1; a satisfying assignment obtained by the procedure above is $\{\langle c_1, \mathbf{t}\rangle, \langle t_1, \mathbf{f}\rangle, \langle o_0, \mathbf{f}\rangle, \langle t_2, \mathbf{t}\rangle, \langle t_3, \mathbf{f}\rangle, \langle a_0, \mathbf{t}\rangle, \langle b_0, \mathbf{t}\rangle, \langle c_0, \mathbf{f}\rangle\}$. Also note that if a total truth assignment τ 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: introduce the clause $(\tilde{g})$ for $\langle g, \mathbf{t}\rangle \in \alpha$, and the clause $(\neg\tilde{g})$ for $\langle g, \mathbf{f}\rangle \in \alpha$.

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 the non-clausal method BC SLS [7] a configuration is described by a total truth assignment as in typical clausal SLS methods. However, the non-clausal 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. Moreover, the key elements of an SLS method – the notion of moves, the objective function, and the stopping criterion – are substantially different from the corresponding elements in clausal SLS methods.

In typical SLS methods for SAT the moves consist of individual flips on variable values in the current configuration. In BC SLS structural knowledge is exploited for making moves on gates: a typical move on a gate g flips the values of a subset of g's children so that g becomes locally justified under the new truth assignment. Moreover, moves are focused on a particular subset of the gates, the justification frontier, which guides the search to concentrate on relevant parts of the instance exploiting *observability don't cares*. In typical clausal SLS methods the objective function measures the number of clauses that are falsified by the current truth assignment. In BC SLS the objective function is based on the concept of justification frontier and uses the set of interesting gates. The notion of a justification frontier leads to a early stopping criterion where the search can be halted when the circuit has been shown to be de facto satisfiable which often occurs before a total satisfying truth assignment has been found.

In this work we extend BC SLS in order to (i) achieve a *probabilistically approximately complete* (PAC) generalization of the method, and to (ii) *exploit adaptive noise mechanisms* within the framework. The resulting generalized framework is described as Algorithm 1. Given a constrained Boolean circuit $\mathcal{C}^\alpha$ the algorithm performs structural local search over the assignment space of *all* the gates in $\mathcal{C}$ (inner loop on lines 3–13). As typical, the *noise parameter* $p \in [0, 1]$ controls the probability of making non-greedy moves (with $p = 0$ only greedy moves are made). Here we introduce an additional parameter $q \in [0, 1]$ which leads to PAC variants of BC SLS. We will consider adaptive noise mechanisms for controlling the value of p during the search in Sect. 4.

Algorithm 1 Generalized BC SLS

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

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

For each of the $\textsc{MaxTries}(\mathcal{C}^\alpha)$ runs, $\textsc{MaxMoves}(\mathcal{C}^\alpha)$ moves are made. As the *stopping criterion* we use the condition that the justification frontier $\mathsf{jfront}(\mathcal{C}^\alpha, \tau)$ is empty. As discussed in Section 2 if $\mathsf{jfront}(\mathcal{C}^\alpha, \tau)$ is empty, then $\mathcal{C}^\alpha$ is satisfiable and a satisfying truth assignment can be computed from τ. Notice that typically this stopping criterion is reached before all gates are justified in the current configuration τ.

Given the current configuration τ, 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 τ, 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 objective function for a move (justification) δ is $\mathsf{cost}(\tau, \delta) = |\mathsf{interest}(\mathcal{C}^\alpha, \tau')|$, 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 δ is the size of the interest set in the configuration τ' where for the gates mentioned in δ we use the values in δ instead of those in τ. The move is then selected randomly from those justifications δ for $\langle g, v\rangle$ for which $\mathsf{cost}(\tau, \delta)$ is smallest over all justifications for $\langle g, v\rangle$.

During a *non-greedy move* (lines 9–12, executed with probability p), we introduce a new parameter q for guaranteeing the PAC property (for PAC proofs, see Section 3.1). For non-greedy moves, the control parameter q defines the probability of justifying the selected gate g by a randomly chosen justification from the set of all justifications for the value of g (this is a *non-greedy downward move*). With probability $(1 - q)$ the non-greedy move consists of inverting the value of *the gate g itself* (a *non-greedy upward move*). The idea in upward moves is to try to escape from possible local minima by more radically changing the justification front. In the special case when g is constrained, a random downward move is done with probability 1.

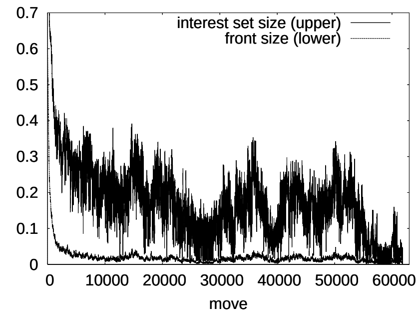

Fig. 2: Comparison of dynamics: sizes of interest set and justification frontier

Notice that 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 applying the objective function of minimizing the size of the interest set, the greedy moves drive the search towards the input gates. Alternatively, one could use the objective of minimizing the size of the justification frontier since moves are concentrated on gates in the frontier and since the search is stopped when the frontier is empty. However, we notice that the size of the interest set is more responsive to quantifying the changes in the configuration than the size of the justification frontier, as exemplified in Fig. 2. The size of the frontier typically drops rapidly close to zero percents from its starting value (the y axis is scaled to $[0, 1]$ in the figure), and after this remains quite stable until a solution is found. This is very similar to the typical behavior observed for objective functions based on the number of unsatisfied clauses in CNF level SLS methods [18]. In contrast, the size of the interest set can vary significantly without visible changes in the size of the justification frontier. Using the size of the interest set rather than the size of the justification frontier also resulted in better performance in preliminary experiments.

3.1 On the PAC Property in BC SLS

We now analyze under which conditions BC SLS is PAC (*probabilistically approximately complete*) [16]. A CNF-level SLS SAT method S is PAC if, for any satisfiable CNF SAT instance F and any initial configuration τ, the probability that S *eventually* finds a satisfying truth assignment for F starting from τ is 1 *without using restarts*, i.e., the number of allowed flips is set to infinity and the number of tries to one. A non-PAC SLS method is *essentially incomplete*. Examples of PAC CNF level SLS methods include GWSAT (with non-zero random walk probability) and UnitWalk, while GSAT, WalkSAT/TABU and Novelty (for arbitrary noise parameter setting) are essentially incomplete [16, 19]. Here we adapt the definition of PAC to the context of BC SLS.

Definition 1. *BC SLS is PAC using fixed parameters p, q if, for any satisfiable constrained circuit $\mathcal{C}^\alpha$ and any initial configuration τ, the probability that BC SLS eventually finds a de facto satisfying assignment for $\mathcal{C}^\alpha$ starting from τ is 1 when setting* $\textsc{MaxTries}(\mathcal{C}^\alpha) = 1$ *and* $\textsc{MaxMoves}(\mathcal{C}^\alpha) = \infty$.

It turns out that for a PAC variant of BC SLS, both upward and downward non-greedy moves are needed.

Theorem 1. *The variant of BC SLS where non-greedy downward moves are allowed with probability q, where $0 < q < 1$, is PAC for any fixed noise parameter $p > 0$.*

Proof. Assume that $\mathcal{C}^\alpha$ is satisfiable, the current assignment is τ, and $\mathsf{jfront}(\mathcal{C}^\alpha, \tau) \neq \emptyset$. We show that by executing the inner loop (lines 3–13) at most $|G|$ times the algorithm reaches a de facto satisfying assignment with probability of at least

$$\left(\frac{1}{|G|} \cdot p \cdot \min(q \cdot \frac{1}{2^{|G|}}, 1 - q)\right)^{|G|}.$$

First, take any satisfying assignment $\tau^\star$ for $\mathcal{C}^\alpha$. Recall that $\mathsf{jfront}(\mathcal{C}^\alpha, \tau^\star) = \emptyset$ by definition. Repeat the following until $\mathsf{jfront}(\mathcal{C}^\alpha, \tau) = \emptyset$.

1. If there is a gate g in the frontier $\mathsf{jfront}(\mathcal{C}^\alpha, \tau)$ such that $\tau(g) \neq \tau^\star(g)$, execute the line 12 that flips the value $\tau(g)$ to $\tau^\star(g)$. Note that g is not constrained by α as both $\tau, \tau^\star \supseteq \alpha$. Thus this step happens with the probability of at least $\frac{1}{|G|} \cdot p \cdot (1 - q)$.
2. Otherwise the current assignment τ is such that all the gates in the justification cone and frontier under τ have the same values as in the satisfying truth assignment $\tau^\star$. Take a gate g in the frontier $\mathsf{jfront}(\mathcal{C}^\alpha, \tau)$. Now there is at least one child of g whose value differs in τ and $\tau^\star$. Execute the line 10 in a way that only flips the values of children of g whose values differ in τ and $\tau^\star$; the value of at least one such child is flipped. This step happens with the probability of at least $\frac{1}{|G|} \cdot p \cdot q \cdot \frac{1}{2^{|G|}}$, where the term $\frac{1}{2^{|G|}}$ comes from the fact that a gate always has less than $|G|$ children, and thus the probability of picking the desired justification is at least $\frac{1}{2^{|G|}}$.

As both steps above (i) flip the value of at least one gate to one in $\tau^\star$ and (ii) never flip a gate whose value already is the same as in $\tau^\star$, they are executed at most $|G|$ times: after this $\tau = \tau^\star$ and thus $\mathsf{jfront}(\mathcal{C}^\alpha, \tau) = \mathsf{jfront}(\mathcal{C}^\alpha, \tau^\star) = \emptyset$. Naturally, it may happen that $\mathsf{jfront}(\mathcal{C}^\alpha, \tau) = \emptyset$ earlier and the process terminates in fewer than $|G|$ steps; now τ is not necessary equal to $\tau^\star$ but is de facto satisfying anyway. Therefore, executing the lines 3–13 $|G|$ times transforms the current assignment into a de facto satisfying assignment with probability of at least $\left(\frac{1}{|G|} \cdot p \cdot \min(q \cdot \frac{1}{2^{|G|}}, 1 - q)\right)^{|G|}$. Since this is non-zero when $p > 0$ and $0 < q < 1$, BC SLS finds a satisfying assignment with probability one as $\textsc{MaxMoves}(\mathcal{C}^\alpha)$ approaches infinity. □

Interestingly, at least for the gate types considered here, downward non-greedy moves can be restricted to *minimal* justifications without affecting Theorem 1.

However, if non-greedy moves are only allowed either (i) upwards or (ii) downwards, then BC SLS becomes essentially incomplete.

Theorem 2. *The variant of BC SLS where non-greedy moves are done only upwards (i.e. when $q = 0$) is essentially incomplete for any fixed noise parameter p.*

Proof. Consider the constrained circuit $\mathcal{C}^\alpha$ in Fig. 3; the subcircuit C_f is such that the gate d can evaluate both to **t** or **f**, depending on the values of the input gates, while C_g is a subcircuit that only allows the gate e to evaluate to **f**. Therefore the gate d must have the value **t** in any (de facto or standard) satisfying assignment. Furthermore, assume that the subcircuit C_g has fewer gates than C_f.

Assume that the current assignment τ assigns the gate d to **f** and that $\langle d, \mathbf{f}\rangle$ is not justified under τ. Now if $\tau(b) =$ **f**, the gate b cannot be in the frontier, and the inner loop (lines 3–13) of BC SLS cannot change the value of d to **t**. If $\tau(b) =$ **t** (and thus b is in the justification cone), either (i) $\tau(e) =$ **t** implying that d is a don't care and thus its value cannot be changed in the inner loop, or (ii) $\tau(e) =$ **f** implying that b is in the frontier and the inner loop can pick an interest set size minimizing justification for b on line 7 (but random justification on line 10 is not in use as $q = 0$).

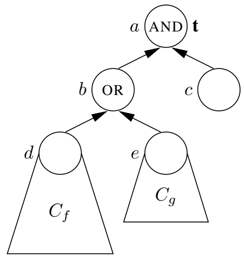

Fig. 3: A circuit

In case (ii), as C_g has fewer gates than C_f and $\langle d, \mathbf{f}\rangle$ is not justified in τ, the greedy move will flip the value of e to **t** and leave d intact because the whole subcircuit C_f becomes a don't care and is removed from the interest set. To sum up, when $q = 0$ the inner loop cannot change the value of d and never finds a de facto satisfying assignment. □

Theorem 3. *The variant of BC SLS where non-greedy moves are done only downwards (i.e. when $q = 1$) is essentially incomplete for any fixed noise parameter p.*

Proof. Consider again the constrained circuit $\mathcal{C}^\alpha$ in Fig. 3 with the assumption that the subcircuit C_f is such that the gate d can evaluate both to **t** or **f**, depending on the values of the input gates, while C_g is a subcircuit that only allows the gate e to evaluate to **f**. Suppose that the current assignment τ assigns b to **t**, d to **f**, and e to **t**. Now the gate b is not in the frontier. Because of this and the fact that the line 12 is never executed when $q = 1$, the (incorrect) value of e cannot be changed in the inner loop (lines 3–13) of BC SLS. Thus b never appears in the frontier and the (incorrect) value of the gate d cannot be changed during the execution of the inner loop. Thus a de facto satisfying assignment is never found. □

3.2 Experiments with non-PAC and PAC variant with Fixed Noise Parameter

Before developing adaptive noise mechanisms for BC SLS (Sect. 4), we look at the performance of BC SLS with the fixed noise parameter setting $p = 0.5$. We experiment with a prototype which is a relatively straightforward implementation of BC SLS

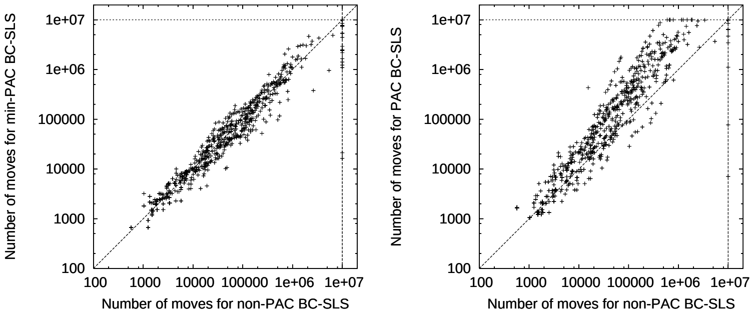

Fig. 4. Non-PAC vs min-PAC BC SLS (left), non-PAC vs PAC BC SLS (right)

constructed on top of the bc2cnf Boolean circuit simplifier/CNF translator [20]. In the implementation, only subset minimal justifications are considered for greedy moves. In all the experiments of this paper we use as main benchmarks a set of Boolean circuits encoding the problem of bounded model checking of various asynchronous systems for deadlocks using the encoding in [21] (as listed in Table 1). Although rather easy for current DPLL solvers, these benchmarks are challenging for typical SLS methods. We limit the number of moves (cutoff) for the variants of BC SLS to 10^7, and run each instance 15 times without restarts. When comparing BC SLS to CNF level SLS procedures, we apply exactly the same Boolean circuit level simplification in bc2cnf to the circuits as in our prototype implementation of BC SLS, and then translate the simplified circuit to CNF with the standard "Tseitin-style" translation.

As the first experiment we compare the essentially incomplete ("non-PAC") version where non-greedy moves are only done upwards ($q = 0$) to two PAC variants (as detailed in Section 3.1): in "min-PAC" 1% of non-greedy moves are randomly selected from the set of *minimal* justifications, while in "PAC" 1% of non-greedy moves are randomly selected from the set of *all* justifications (that is, in both cases we set $q = 0.01$ so that the downward non-greedy moves do not become dominating).

It turns out that the variants "non-PAC" and "min-PAC" have quite similar performance (left in Fig. 4) except that "non-PAC" exceeds the cutoff more often. Surprisingly, the "PAC" version, where also non-minimal random justifications are allowed, does not perform as well as the other two variants (right in Fig. 4). With this evidence, we will in all the following experiments apply the "min-PAC" variant of BC SLS.

In the following experiments, we concentrate on evaluating adaptive noise mechanisms for BC SLS, and compare the resulting methods to adaptive clausal SLS methods. We note that a comparison of ("non-PAC") BC SLS using fixed noise parameter setting with WalkSAT is provided in [7] with the results that BC SLS exhibits typically a one-to-four-decade reduction in the number of moves compared to WalkSAT.

4 Adaptive Noise Strategies for BC SLS

Considering CNF level SLS methods for SAT, it has been noticed that SLS performance can vary critically depending on the chosen noise setting [4], and the optimal noise setting can vary from instance to instance and within families of similar instances. The same phenomenon is present also in BC SLS. The average number of moves over 100 runs of BC SLS with different noise parameter settings is shown in Fig. 5 for two different families of increasingly difficult Boolean circuit instances. This observation has led to the development of an *adaptive noise mechanism* for CNF level SLS in the solver AdaptNovelty+ [4], dismissing the requirement of a pre-tuned noise parameter. This idea has been successfully applied in other SLS solvers as well [22]. We now consider strategies for adapting noise in BC SLS.

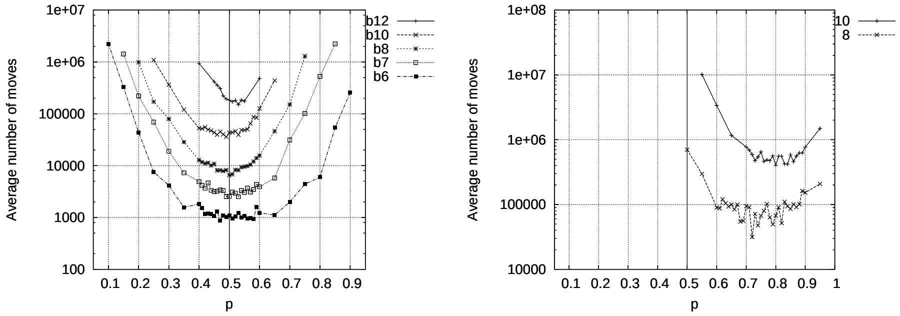

Fig. 5. Average number of moves for BC SLS with different noise parameter settings; left: LTS BMC instance family speed-p, right: factoring instance family braun (see http://www.tcs.tkk.fi/Software/genfacbm/)

4.1 Adaptive Noise in the Context of BC SLS

Following the general idea presented in [4], a generic adaptive noise mechanism for BC SLS is presented as Algorithm 2. Starting from $p = 0$, the noise setting is tuned

---

Algorithm 2 Generic Adaptive Noise Mechanism

---

p: noise (initially $p = 0$)
adapt_score: score at latest noise change
adapt_step: step of latest noise change
1: **if** score $<$ adapt_score **then** *%% noise decrease*
2: $\quad p := p - \frac{\phi}{2} \cdot p$
3: $\quad$ adapt_step := step
4: $\quad$ adapt_score := score
5: **else**
6: $\quad$ **if** (step $-$ adapt_step) $>$ WAITINGPERIOD() **then** *%% noise increase*
7: $\qquad p := p + \phi \cdot (1 - p)$
8: $\qquad$ adapt_step := step
9: $\qquad$ adapt_score := score

---

during search based on the development of the objective function value. *Every time* the objective function value is improved, noise is decreased according to line 2. If no improvement in the objective function value has been observed during the last WAITINGPERIOD() steps, the noise is increased according to line 7, where $\phi \in\ ]0, 1[$ controls the relative amount of noise increase. Each time the noise setting is changed, the current objective function value is then stored for the next comparison.

Hoos [4] suggests, reporting generally good performance, to use $\phi = \frac{1}{5}$ and the *static* function $\theta \cdot C$ for WAITINGPERIOD(), where $\theta = \frac{1}{6}$ is a constant and C denotes the number of clauses in the CNF instance at hand. These parameter values have been applied also in other CNF level SLS solvers [22].

For BC SLS, as the first step we fix ϕ accordingly to $\frac{1}{5}$, and focus on investigating the effect of applying different waiting periods for noise increases in the context of BC SLS. First we investigate using as $\textsc{WaitingPeriod}()$ a static linear function $\theta \cdot U$, where the number U of unconstrained gates is multiplied by a constant θ. In fact, opposed to reported experience with CNF level SLS, it turns out that for BC SLS $\theta = \frac{1}{6}$ is too large: by decreasing θ we can increase the performance of BC SLS. As shown in Fig. 6 (left), by decreasing θ to $\frac{1}{24}$ we witness an evident overall gain in performance against $\theta = \frac{1}{6}$ (left), and again by decreasing θ from $\frac{1}{24}$ to $\frac{1}{96}$ (right).

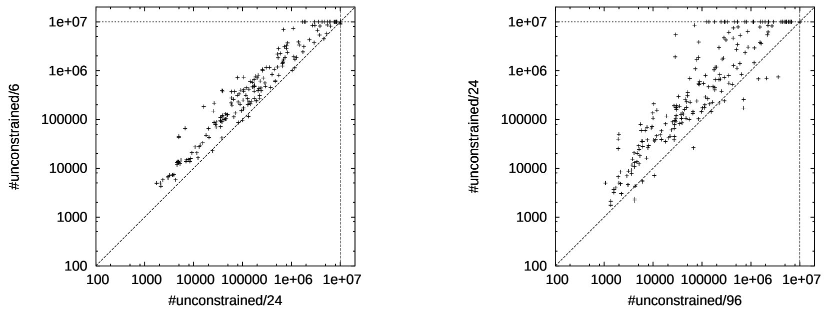

Fig. 6. Comparison of number of moves; left: $\theta = \frac{1}{24}$ vs $\theta = \frac{1}{6}$, right: $\theta = \frac{1}{96}$ vs $\theta = \frac{1}{24}$

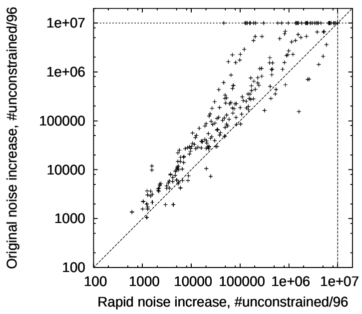

Fig. 7: Comparison of number of moves: rapidly increasing vs original noise mechanism

However, we noticed that changing the overall scheme in the original adaptive noise mechanism leads to even better performance for BC SLS. In the novel scheme, which we call *rapidly increasing*, when the waiting period is exceeded, the noise level is increased after *each* step until we see the first one-step improvement in the objective function. This can be implemented by removing line 8 in Algorithm 2. An example of the resulting improvement is shown in Fig. 7, in which the original and rapidly increasing noise mechanism are compared using $\theta = \frac{1}{96}$. In the following, we will apply the rapidly increasing noise mechanism for BC SLS.

We next compare BC SLS with $\theta = \frac{1}{96}$ to AdaptNovelty+ [23]. Our current prototype of BC SLS does compute the effect of moves on the justification cone and interest set incrementally but is otherwise relatively unoptimized. The results shown in Table 1 are encouraging: BC SLS usually makes much fewer moves and is able to solve more instances in the given time limit than AdaptNovelty+. Although making moves is slower in our BC SLS prototype (around 100000 moves per second on average) than in Adapt-

Novelty+ (2.5 million per second), BC SLS is very competitive also in running times on these instances as less moves are usually needed for finding a solution.

It it interesting to look at how the noise level fluctuates during a run with different values of θ. An example is provided in Fig. 8 where, using instance `dp_12.fsa-b6-s`, the development of p is shown for $\theta = \frac{1}{6}, \frac{1}{24}, \frac{1}{96}$ (from top to bottom) on runs of similar length. It appears that with larger θ, a significant portion of moves are wasted on plateaus, from which we can escape only with a strong noise increase. On the other hand, for small values, such as $\frac{1}{96}$, the noise level seems to thrash heavily, not focusing on a specific noise range. From another viewpoint, we observed that lowering the value of θ basically raises the average noise level.

Table 1. Comparison of AdaptNovelty+ and BC SLS (static adaptive noise mechanism, $\theta = \frac{1}{96}$): 101 runs for each instance, 5–minute time limit for each run. succ %: percent of succesful runs.

Instance			**BC SLS $\theta = \frac{1}{96}$**					**AdaptNovelty+**				
name	**vars**	**clauses**	**succ %**	**time**		**#moves**		**succ %**	**time**		**#moves**	
				min	**med**	**min**	**med**		**min**	**med**	**min**	**med**
dp_12.fsa-b5-p.bc	953	2966	100	0.1	1.0	4272	149287	100	0.1	0.1	4105	10012
dp_12.fsa-b6-p.bc	1362	4236	100	0.1	0.7	7996	79106	100	0.1	0.1	11006	29010
dp_12.fsa-b7-p.bc	1771	5506	100	0.1	0.6	11504	67705	100	0.1	0.1	23519	72153
dp_12.fsa-b8-p.bc	2180	6776	100	0.2	1.5	21143	142100	100	0.1	0.1	48525	215934
dp_12.fsa-b9-p.bc	2589	8046	100	0.1	4.6	18056	376007	100	0.1	0.3	109929	817996
dp_12.fsa-b5-s.bc	1337	4146	100	0.1	0.1	6234	17642	100	0.1	0.1	9240	22320
dp_12.fsa-b6-s.bc	1746	5416	100	0.1	0.3	9119	37626	100	0.1	0.1	27853	58083
dp_12.fsa-b7-s.bc	2155	6686	100	0.1	1.0	18480	86447	100	0.1	0.1	40098	136157
dp_12.fsa-b8-s.bc	2564	7956	100	0.1	3.1	19857	247490	100	0.1	0.1	60910	369385
dp_12.fsa-b9-s.bc	2973	9226	100	0.3	9.5	38487	730250	100	0.1	2.1	170040	5212785
elevator_1-b4-s.bc	439	1343	100	0.1	0.1	394	1707	100	0.1	0.1	2866	81606
elevator_1-b5-s.bc	698	2149	100	0.1	0.1	1365	3844	100	0.1	0.5	7961	1254582
elevator_1-b6-s.bc	1087	3374	100	0.1	0.8	2507	60052	100	1.4	15.5	3693776	42037729
elevator_2-b6-p.bc	682	2115	100	0.1	0.1	982	4366	100	0.1	5.5	149405	15053510
elevator_2-b7-p.bc	1253	3952	100	0.1	0.7	4120	37607	93	1.3	82.3	3406967	220184348
elevator_2-b6-s.bc	1333	4143	100	0.1	0.2	4389	17761	82	0.3	122.3	832838	329714970
elevator_2-b7-s.bc	2063	6478	100	0.2	1.1	11526	65931	6	36.7	-	94059483	-
elevator_2-b8-s.bc	3123	9919	67	1.7	179.9	79857	7254453	0	-	-	-	-
mmgt_2.fsa-b6-p.bc	654	2036	100	0.1	0.1	569	12878	100	0.1	0.1	11902	308130
mmgt_2.fsa-b7-p.bc	928	2895	100	0.1	0.2	3027	26968	100	0.1	0.3	80656	1468861
mmgt_2.fsa-b8-p.bc	1317	4119	94	0.1	74.3	6293	6395263	100	0.1	34.0	70058	102384691
mmgt_2.fsa-b6-s.bc	1182	3708	100	0.1	0.1	3148	12644	95	1.8	89.2	4798784	239335425
mmgt_2.fsa-b7-s.bc	1723	5429	100	0.1	6.0	8989	347129	0	-	-	-	-
mmgt_2.fsa-b8-s.bc	2381	7530	100	1.2	29.1	60339	1315753	0	-	-	-	-
mmgt_3.fsa-b7-p.bc	1421	4459	100	0.1	0.4	3456	44913	100	0.1	0.1	26370	377011
mmgt_3.fsa-b9-p.bc	2596	8184	100	0.3	29.4	23771	1759402	27	4.8	-	12129665	-
mmgt_3.fsa-b7-s.bc	2588	8226	100	0.2	2.8	11575	154457	0	-	-	-	-
speed_1.fsa-b6-p.bc	498	1514	100	0.1	0.1	385	1159	100	0.1	0.1	1327	26923
speed_1.fsa-b7-p.bc	758	2319	100	0.1	0.1	902	2935	100	0.1	0.1	7364	132024
speed_1.fsa-b8-p.bc	1021	3132	100	0.1	0.1	2125	7914	100	0.1	0.3	43042	919969
speed_1.fsa-b9-p.bc	1284	3944	100	0.1	0.2	3482	17454	100	0.1	2.6	46186	6812540
speed_1.fsa-b10-p.bc	1547	4754	100	0.1	0.4	5382	46156	100	0.4	18.5	1000674	48965683
speed_1.fsa-b12-p.bc	2073	6368	100	0.2	4.8	20250	499851	15	24.3	-	57759838	-
speed_1.fsa-b13-p.bc	2336	7172	100	1.2	40.8	123031	4332369	0	-	-	-	-
speed_1.fsa-b14-p.bc	2599	7974	34	7.0	-	744191	-	0	-	-	-	-
speed_1.fsa-b6-s.bc	666	2026	100	0.1	0.1	603	1278	100	0.1	0.1	2326	13049
speed_1.fsa-b7-s.bc	920	2811	100	0.1	0.1	1238	2409	100	0.1	0.1	6308	47237
speed_1.fsa-b8-s.bc	1175	3596	100	0.1	0.1	2025	4185	100	0.1	0.1	12134	98165
speed_1.fsa-b9-s.bc	1430	4380	100	0.1	0.1	2820	8629	100	0.1	0.1	29602	237623
speed_1.fsa-b10-s.bc	1685	5162	100	0.1	0.2	3514	14860	100	0.1	0.3	52643	790049
speed_1.fsa-b12-s.bc	2195	6722	100	0.1	1.2	8500	100027	100	0.3	6.6	723313	17287780
speed_1.fsa-b13-s.bc	2450	7499	100	0.4	3.7	30637	311209	84	1.5	135.4	3814440	354742108
speed_1.fsa-b14-s.bc	2705	8274	100	0.2	12.3	17063	1072742	15	1.8	-	4647662	-
speed_1.fsa-b15-s.bc	2960	9047	92	1.2	67.9	102953	6013459	3	0.4	-	982942	-

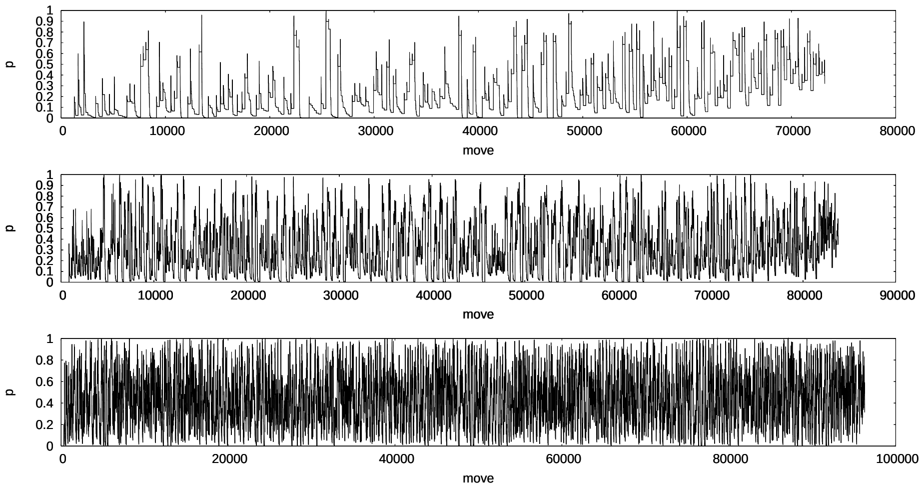

Fig. 8. Noise level fluctuations during a run using $\theta = \frac{1}{6}, \frac{1}{24}, \frac{1}{96}$ (from top to bottom)

Now, the original motivation behind developing adaptive noise mechanisms comes from the fact that the optimal noise level is instance-specific (recall Fig. 5). Apparently a sufficient amount of noise is needed, which can be achieved by lowering the fixed value of θ, but then the hence shortened waiting period for noise increases results in unfocused fluctuations of the noise level. That is, by employing the adaptive noise mechanism based on static waiting periods, we may have only changed the problem of finding the optimal static noise level parameter p into the problem of finding an instance-specific optimal value for θ. This motivates us to consider, opposed to a static waiting period controlled by the addition parameter θ, *dynamic waiting periods* based on the state of search, with the possibility of dismissing the otherwise required constant θ.

We consider two dynamic alternatives: $\textsc{WaitingPeriod}() = \mathsf{jfront}(\mathcal{C}^{\alpha}, \tau)$ (the size of the current justification frontier), and $\textsc{WaitingPeriod}() = \mathsf{interest}(\mathcal{C}^{\alpha}, \tau)$ (the size of the current interest set). The intuition behind using front is that since the gate at each step is selected from the justification frontier, the size of the frontier gives us an estimate on the number of possible greedy moves in order to improve the objective function value before increasing the possibility of non-greedy moves (increasing noise). On the other hand, the size of the interest set is precisely the objective function value. Intuitively, the greater the objective function value is, the further we are from a solution, and thus more effort is allowed on finding a good greedy move.

Fig. 9 gives a comparison of performance using the static waiting period with $\theta = \frac{1}{96}$ with the performance resulting from using dynamic waiting period based on frontier size (left) and interest set size (right). The dynamic waiting period results in comparable performance than the static one, although we notice that with the dynamic approach based on frontier size seems to behave more similarly to the static one than the dynamic approach based on interest set size.

This difference is highlighted by looking at the fluctuations of the noise level for the dynamic waiting periods (exemplified in Fig. 10). Especially the noise level fluctuation resulting from the interest set size approach seems to be more focused than when using the static waiting period with $\theta = \frac{1}{96}$ (recall Fig. 8 (bottom)), avoiding some of the observed thrashing behavior without needing to choose a specific value for θ. The

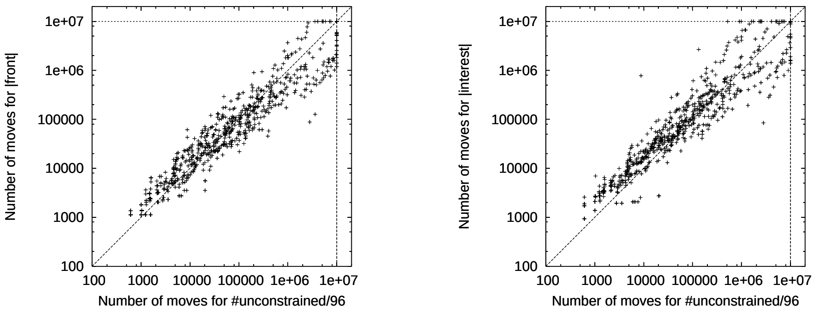

Fig. 9. $\theta = \frac{1}{96}$ vs front (left); $\theta = \frac{1}{96}$ vs interest (right).

question of to what extend thrashing may affect performance is an interesting aspect of further work.

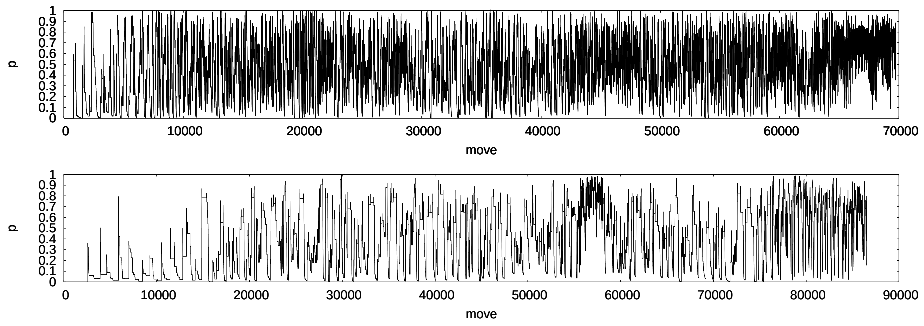

Fig. 10. Noise level fluctuation during a run using front (top) and interest set size (bottom)

5 Conclusions

We extend a recent framework BC SLS [7] for a novel type of stochastic local search (SLS) for SAT. We analyze in detail under which conditions the extended framework is probabilistically approximately complete and under which essentially incomplete. We develop and study adaptive noise mechanisms for BC SLS. The results suggest that, compared to the parameter values for adapting noise used in CNF level SLS methods, radically different settings are required in BC SLS. As more fundamental changes to the CNF level noise mechanism, we demonstrate improvements in performance for BC SLS by introducing the *rapidly increasing* noise mechanism, and show that there is promise for dismissing the static waiting period constant θ required in current CNF level noise mechanisms by *dynamically* adapting the waiting period for noise increases. Compared to well-known CNF level SLS methods, a prototype implementation of the framework performs favorably w.r.t. the number of moves, showing promise for more optimized implementations of the procedure. An interesting question regarding dynamic waiting periods is whether CNF level SLS methods can gain from similar mechanisms.

Acknowledgements. Research supported by Academy of Finland (grants #122399 and #112016). Järvisalo additionally acknowledges financial support from HeCSE graduate school, Emil Aaltonen Foundation, Jenny and Antti Wihuri Foundation, Nokia Foundation, and Finnish Foundation for Technology Promotion.

References

1. Selman, B., Levesque, H., Mitchell, D.: A new method for solving hard satisfiability problems. In: AAAI. (1992) 440–446
2. Selman, B., Kautz, H., Cohen, B.: Noise strategies for improving local search. In: AAAI. (1994) 337–343
3. McAllester, D., Selman, B., Kautz, H.: Evidence for invariants in local search. In: AAAI. (1997) 321–326
4. Hoos, H.: An adaptive noise mechanism for WalkSAT. In: AAAI. (2002) 655–660
5. Braunstein, A., Mézard, M., Zecchina, R.: Survey propagation: An algorithm for satisfiability. Random Structures and Algorithms **27**(2) (2005) 201–226
6. Kautz, H., Selman, B.: The state of SAT. Discr. Appl. Math. **155**(12) (2007) 1514–1524
7. Järvisalo, M., Junttila, T., Niemelä, I.: Justification-based non-clausal local search for SAT. In: ECAI. Volume 178 of Frontiers in AI and Applications., IOS Press (2008) 535–539
8. Junttila, T., Niemelä, I.: Towards an efficient tableau method for Boolean circuit satisfiability checking. In: CL 2000. Volume 1861 of LNAI., Springer (2000) 553–567
9. Kuehlmann, A., Ganai, M., Paruthi, V.: Circuit-based Boolean reasoning. In: DAC, ACM (2001) 232–237
10. Kuehlmann, A., Paruthi, V., Krohm, F., Ganai, M.K.: Robust Boolean reasoning for equivalence checking and functional property verification. IEEE T-CAD **21**(12) (2002) 1377–1394
11. Thiffault, C., Bacchus, F., Walsh, T.: Solving non-clausal formulas with DPLL search. In: CP. Volume 3258 of LNCS., Springer (2004) 663–678
12. Safarpour, S., Veneris, A., Drechsler, R., Lee, J.: Managing don't cares in Boolean satisfiability. In: DATE'04, IEEE (2004)
13. Sebastiani, R.: Applying GSAT to non-clausal formulas. J.Artif.Intell.Res. **1** (1994) 309–314
14. 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`.
15. Pham, D., Thornton, J., Sattar, A.: Building structure into local search for SAT. In: IJCAI. (2007) 2359–2364
16. Hoos, H.H.: On the run-time behaviour of stochastic local search algorithms for SAT. In: AAAI. (1999) 661–666
17. Patterson, D.J., Kautz, H.: Auto-Walksat: A self-tuning implementation of Walksat. In: SAT, 4th Workshop on Theory and Application of Satisfiability Testing. (2001)
18. Selman, B., Kautz, H.: An empirical study of greedy local search for satisfiability testing. In: AAAI. (1993) 46–51
19. Hirsch, E., Kojevnikov, A.: UnitWalk: A new SAT solver that uses local search guided by unit clause elimination. Ann. Math. Artif. Intell. **43**(1) (2005) 91–111
20. Junttila, T.: The BC package and a file format for constrained Boolean circuits `http://www.tcs.hut.fi/~tjunttil/bcsat/`.
21. Heljanko, K.: Bounded reachability checking with process semantics. In: CONCUR. Volume 2154 of LNCS., Springer (2001) 218–232
22. Li, C., Wei, W., Zhang, H.: Combining adaptive noise and look-ahead in local search for SAT. In: SAT. Volume 4501 of LNCS., Springer (2007) 121–133
23. Tompkins, D., Hoos, H.: UBCSAT: An implementation and experimentation environment for SLS algorithms for SAT and MAX-SAT. In: SAT 2004 Revised Selected Papers. Volume 3542 of LNCS., Springer (2005) 306–320