Boosting local search thanks to CDCL

Gilles Audemard, Jean-Marie Lagniez, Bertrand Mazure, Lakhdar Saïs

Boosting local search thanks to CDCL

Gilles Audemard, Jean-Marie Lagniez, Bertrand Mazure, and Lakhdar Saïs

Université Lille-Nord de France
CRIL - CNRS UMR 8188
Artois, F-62307 Lens
{audemard,lagniez,mazure,sais}@cril.fr

Abstract. In this paper, a novel hybrid and complete approach for propositional satisfiability, called SATHYS (*Sat Hybrid Solver*), is introduced. It efficiently combines the strength of both local search and CDCL based SAT solvers. Considering the consistent partial assignment under construction by the CDCL SAT solver, local search is used to extend it to a model of the Boolean formula, while the CDCL component is used by the local search one as a strategy to escape from a local minimum. Additionally, both solvers heavily cooperate thanks to relevant information gathered during search. Experimentations on SAT instances taken from the last competitions demonstrate the efficiency and the robustness of our hybrid solver with respect to the state-of-the-art CDCL based, local search and hybrid SAT solvers.

1 Introduction

The SAT problem, namely the issue of checking whether a Boolean formula in Conjunctive Normal Form (CNF) is satisfiable or not, is a central issue in many artificial intelligence and computer science domains, including hardware and software verification, planning, cryptography and bioinformatics. These last two decades, many approaches have been proposed to solve large application SAT instances, based on logically complete or incomplete methods. Both stochastic local search (SLS) techniques [34, 33, 20] and elaborate variants of the DPLL procedure [7], commonly named modern SAT solvers [30, 9], can now solve many families of hard SAT instances. Based on different paradigms, these two kinds of approaches admit complementary behavior in terms of performances. Modern SAT or CDCL (Conflict Driven Clause Learning) solvers are particularly efficient on application benchmarks while local search performs better on random SAT instances. Stochastic Local search (Algorithm 2) and modern SAT solvers (Algorithm 1) are recognized as two important search paradigms. Their differences arise in the way the search space is explored. In SLS, the algorithm explores the search space in a non systematic way starting from a complete assignment and moving to another complete assignment by inverting the truth value of a chosen variable (this is a flip). After a fixed number of flips, another complete assignment is generated, and the process is repeated. Modern SAT solvers explore the search space in a systematic way by developing a search tree, where at each node the current partial assignment is extended by assigning a selected variable and propagating unit literals. In SLS, the search process can lead to a local minimum, in this case several strategies are designed to escape from

such minimum. In modern SAT solvers, the process can lead to a conflict and in this case several learning strategies are designed for resolving such a conflict [38].

Combining stochastic local search and conflict driven clause learning solvers is clearly a challenging issue as stated by Selman *et al.* [35] in 1997 (challenge 7) and in 2003 [24]. Such a combination might exploit the strength of both approaches and might result in a new but more efficient hybrid SAT solver. Several attempts have been made these last years [4] and different hybrid solvers have been designed leading to real progress towards the resolution of this challenging issue (see section 5). However, the challenge remains open as the performance of these proposed hybrid solvers is far from those of the CDCL based solvers particularly on application instances. Our goal in this paper is to design a hybrid SLS/CDCL solver that outperforms the local search techniques, while significantly reducing the gap with CDCL based solvers particularly on application category.

In this paper, we propose a new hybridization of local search and modern SAT solver, named SATHYS (*Sat Hybrid Solver*). In our approach, both components heavily cooperate through relevant information gathered during search. More precisely, our hybrid solver alternatively performs the search process using local search and CDCL based SAT solvers. On the one hand, at each node of the search tree, the local search component is used to extend to the model, the current (consistent) partial assignment built by the CDCL based component. On the other hand, the CDCL part is conditionally invoked by the local search component when a local minimum is encountered. Each solver benefits from the other in several ways. First, each time a local minimum is reached, the local search technique updates the activity of the boundary variables [14]. The idea is to direct the CDCL search towards boundary points proven to be important by Goldberg in [14]. Secondly, the polarities of the literals involved in the best complete assignment found during local search are exploited by the CDCL component. From the other side, the CDCL solver shares with local search the current partial assignment together with the learnt clauses. The originality of our proposed hybrid SAT solver arises in alternating search of both components while exchanging relevant information.

The rest of the paper is organized as follows. Section 2 introduces some definitions and notations. Then, we present CDCL and SLS solvers in a unified way. Section 3 describes our new hybrid solver. In Section 4 we provide some experimental comparison with different SAT solvers including the CDCL solver MINISAT and several well known SLS and hybrid solvers. Before concluding, section 5 discusses the related works.

2 Technical Background

2.1 Preliminary definitions and notations

Let $\mathcal{V} = \{x_1, ..., x_n\}$ be a set of propositional variables, a *literal* ℓ is either a positive x_i or a negative variable $\overline{x_i}$. The two literals x and $\overline{x}$ are said *complementary*. We denote by $\overline{\ell}$ the complementary literal of ℓ. A *clause* $c_i = (\ell_1 \vee \ldots \vee \ell_{n_i})$ is a disjunction of literals. A *unit clause* is a clause with only one literal, called *unit literal*. A formula $\Sigma = (c_1 \wedge \ldots \wedge c_m)$ is in conjunctive normal form (CNF) as it is a conjunction of clauses. The set of variables involved in Σ will be noted $\mathcal{V}_\Sigma$. An *interpretation* $\mathcal{I}$ of a Boolean

formula Σ associates a truth value $\mathcal{I}(x) \in \{false, true\}$ to some variables $x \in \mathcal{V}_\Sigma$. $\mathcal{I}$ is *complete* if it assigns a truth value to every $x \in \mathcal{V}_\Sigma$, and *partial* otherwise. A complete (resp. partial) interpretation will be noted $\mathcal{I}_c$ (resp. $\mathcal{I}_p$). A clause, a CNF formula and an interpretation can be represented using sets.

It should say that a *clause β is falsified (resp. satisfied) by $\mathcal{I}$* if $\forall \ell \in \beta$ (resp. $\exists \ell \in \beta$), β is falsified (resp. satisfied) by ℓ under $\mathcal{I}$. $\Sigma_{|\ell}$ denotes the formula simplified by the assignment of the literal ℓ to $true$, that is $\Sigma_{|\ell} = \{\alpha \setminus \{\bar{\ell}\}$ such that $\alpha \in \Sigma$ and $\ell \notin \alpha\}$. This notation can be extended to an interpretation. Let $\mathcal{I} = \{\ell_1, ..., \ell_n\}$ be an interpretation, $\Sigma_{|_\mathcal{I}} = (...(\Sigma|_{\ell_1})...|_{\ell_n})$.

A *model* of a formula Σ, noted $\mathcal{I} \models \Sigma$, is an interpretation $\mathcal{I}$ such that $\forall c \in \Sigma$, c is satisfied by $\mathcal{I}$. On the contrary, an interpretation $\mathcal{I}$ is called a *nogood* of Σ, noted $\mathcal{I} \not\models \Sigma$, if $\exists c \in \beta$ such that c is falsified by $\mathcal{I}$.

SAT is the problem of checking whether a CNF formula admits a model or not. If the answer is positive, then the formula is called *satisfiable*, else it is called *unsatisfiable*.

2.2 CDCL solvers

CDCL based SAT solvers (Algorithm 1), generally referred as modern SAT solvers [30], are based on classical unit propagation (lines 2, 14 and 15) efficiently combined through incremental data structures, restart policies (line 15) [15], activity-based variable selection heuristics (VSIDS-like) (line 12) [30], and clause learning (line 6) [36].

A CDCL based SAT solver is a sophisticated variant of the well known DPLL [7] procedure. At each node of the search tree, the assigned literals (the decision literal and the propagated ones) are labeled with the same *decision level* starting from 1 and increased at each decision (or branching step). After backtracking, some variables are unassigned and the current decision level is decreased accordingly. At level m, the current partial interpretation $\mathcal{I}_p$ can be represented as an ordered sequence of decision-propagation steps made at the different levels. We note $\mathcal{I}_p^i$ where $i \leq m$, the partial interpretation $\mathcal{I}_p$ restricted to the first i sequences of decision-propagation. An *asserting level* associated to a falsified clause $\alpha = (x \vee \beta)$ under an interpretation $\mathcal{I}_p$ (in short $level(\alpha, \mathcal{I}_p)$) is defined as the level j such that $\alpha|_{\mathcal{I}_p^j} = x$ and x represents the *asserting literal* of α under $\mathcal{I}_p$

Algorithm 1 describes the general skeleton of a CDCL based SAT solver. Let us briefly explain its main components. A more detailed description of modern SAT solvers can be found in [6]. The algorithm initializes the learnt database Γ as the empty set (line 1), then unit propagation is applied on the original formula Σ (line 2). The function $BCP(\Sigma)$ returns the set of propagated literals. The main loop contains several cases. First, if all variables are assigned then the formula is empty and $\mathcal{I}$ is a model of Σ (line 4). If the partial assignment $\mathcal{I}_p$ is consistent, then a new decision variable is chosen and unit propagation is performed (lines 12, 13 and 14). Otherwise, a conflict is reached and a nogood (or a learnt clause) α is computed (line 6). If α is the empty clause, then the instance is proven unsatisfiable (line 8). Otherwise, $\alpha = (x \vee \beta)$ and j the asserting level of α w.r.t. $\mathcal{I}_p$. After that, the clause α is added to the learnt database (line 9) and the algorithm backtracks to level j by computing a new partial interpretation $\mathcal{I}_p^j$ (line 10). At level j the asserting literal $x \in \alpha$ is propagated (line 14). Finally, the algorithm restarts if the cutoff value in terms of the number of conflicts is reached (line 15).

Algorithm 1: CDCL solver

Input: a CNF formula Σ
Output: SAT or UNSAT

1 $\Gamma \leftarrow \emptyset$;
2 $\mathcal{I}_p \leftarrow BCP(\Sigma)$;
3 **while** *(true)* **do**
4 **if** $(\Sigma_{|\mathcal{I}_p} = \emptyset)$ **then return** SAT;
5 **if** $(\emptyset \in \Sigma_{|\mathcal{I}_p})$ **then**
6 $\alpha{=}(x \vee \beta) \leftarrow analyzeConflict(\Sigma \cup \Gamma, \mathcal{I}_p)$;
7 $j \leftarrow level(\alpha, \mathcal{I}_p)$;
8 **if** $(\alpha = \emptyset)$ **then return** UNSAT;
9 $\Gamma \leftarrow \Gamma \cup \{\alpha\}$;
10 $\mathcal{I}_p \leftarrow \mathcal{I}_p^j$;
11 **else**
12 $x \leftarrow chooseDecisionLiteral(\Sigma_{|\mathcal{I}_p})$;
13 $\mathcal{I}_p \leftarrow \mathcal{I}_p \cup \{x\}$;
14 $\mathcal{I}_p \leftarrow \mathcal{I}_p \cup BCP((\Sigma \cup \Gamma)_{|\mathcal{I}_p})$;
15 **if** $(restart())$ **then** $\mathcal{I}_p \leftarrow BCP(\Sigma \cup \Gamma)$;

2.3 Local search solvers

Algorithm 2 gives the general scheme of a local search solver. It uses a stochastic walk over complete interpretations of Σ. At each *step* (or *flip*), it tries to reduce the number of falsified clauses (this step is usually called a *descent*). The next complete interpretation is chosen among the neighbors (interpretations that differ only by the truth value of one variable) of the current one (line 7). A local minimum is reached when no descent is possible. In such a case, a strategy is used in order to escape from this minimum (line 5). Like in CDCL solvers, sometimes a restart (called a try in SLS) is performed and a new complete interpretation is generated. However restarts in CDCL and SLS solvers have not exactly the same role. In the CDCL case, it is now well admitted that restarts are used to reorder variables, directing the solver to the same part of the search space using a different branch. On the other hand, restarts in SLS solvers are used in order to diversify the search by generating a new complete interpretation.

3 SATHYS: A novel Hybrid Approach

In this section, we present SATHYS, our hybrid solver for SAT which is available at `http://www.cril.fr/~lagniez/sathys`. Before giving its formal description, we provide some intuitions and motivations behind the design of SATHYS.

3.1 Motivations

Recently, several works have shown that local search techniques provide relevant information for locating unsatisfiable cores [16, 17]. Motivated by these results, we propose

Algorithm 2: Local Search solver

Input: a CNF formula Σ
Output: SAT

```
1 I_c ← completeInterpretation(Σ) ;
2 while (true) do
3     if (Σ_{|I_c} = ∅) then return SAT;
4     if a local minimum is reached then
5         x ← chooseLitteralToEscapeMinima(Σ, I_c);
6     else
7         x ← chooseLitteralToFlip(Σ, I_c);
8     flip(I_c, x);
9     if (restart()) then I_c ← completeInterpretation(Σ);
```

a new hybridization scheme of SLS and CDCL solvers. In our hybrid approach, the two methods heavily interact and play complementary roles :

1. SLS directs the CDCL search towards the proof of unsatisfiability ;
2. CDCL directs the SLS search towards a model if it exists.

Let us briefly summarize these bidirectional interactions.

SLS $\rightarrow$ CDCL: The SLS affects the CDCL solver at two different levels known to be important for the efficiency of satisfiability solvers. First, the activity of the variables usually maintained by CDCL are dynamically refined by the SLS component. This local search refinement of the activity-based heuristic (VSIDS) is achieved using the notion of boundary points introduced recently by E. Goldberg in [14]. A boundary point is a complete interpretation (point) $\mathcal{I}$ such that there exists at least one literal ℓ belonging to all clauses falsified by $\mathcal{I}$. In [14], it was shown that focusing resolution on such kind of literals ℓ improves the proof quality by reducing the length of the refutation. From the recent work by Grégoire *et al.* [17], we can also deduce that if the boundary point corresponds to a local minimum, then the literal ℓ belongs to at least one unsatisfiable core. Considering these two interesting features of boundary points, we exploit this notion to refine the activity of the clauses. When a local minimum is reached, updating the activity of the variables associated to literals appearing in all the falsified clauses by the boundary point might guide CDCL towards the most important part of the search space. Indeed, favoring the assignment of such literals aims to direct the search to unsatisfiable cores and might reduce the length of the resolution proof.

The second interaction level where SLS influences with CDCL is on the assignment polarity ($false$, $true$) of the chosen variable. More precisely, when the next variable to assign is chosen by CDCL, its polarity is taken from the best complete interpretation (in terms of the number of satisfied clauses) found by the SLS solver. Usually, in CDCL solvers such as Rsat [32], the assignment polarity (or phase) follows a progress saving scheme. Each time a variable is assigned, its truth value is saved, and used as the polarity of the variable when such a variable is selected again.

CDCL → SLS: From CDCL to SLS, the current CDCL branch made of the set of decisions and propagations is provided to SLS. Using such a partial interpretation, the SLS solver tries to extend it into a model by focusing search on the remaining unassigned variables. In this way, local search exploits variable dependencies provided by the decisions and propagations of the CDCL solver. Let us recall that exploiting variable dependencies is identified as an important issue for the efficiency of local search based techniques [35, 24, 31]. On the other hand, as the CDCL solver is called when SLS reaches a local minimum, the new partial interpretation revised by CDCL contains several flipped variables. Consequently, from the local search side CDCL can be seen as a strategy that helps local search to escape from local minima.

3.2 Formal description

Even if the main core of the SATHYS hybrid solver is a local search based algorithm, the integration of the CDCL part leads to a complete hybrid solver able to prove both satisfiability and unsatisfiability. In this section, we give a detailed description of our approach as depicted by the Algorithm 3.

First of all, the algorithm starts with an empty set of learnt clauses (Γ) and a complete interpretation $\mathcal{I}_c$ built by randomly extending the current partial interpretation $\mathcal{I}_p$ obtained by propagating unit literals on the original formula (lines 2–3). Note that function `BCP` (Boolean Constraint Propagation) returns the set of propagated unit literals. After this initialization step local search process is performed. If there exists a neighboring interpretation (an interpretation that differs from $\mathcal{I}_c$ by only the truth value of one variable x), that reduces the number of falsified clauses (descent phase), then the variable x is flipped, i.e. its truth value is reversed (lines 22-23). It is important to note that only variables out of $\mathcal{I}_p$ can be flipped. Indeed, the variables involved in $\mathcal{I}_p$ are considered tabu during the local search step. This partial interpretation evolves during the search and is modified in the CDCL part of the algorithm (lines 11–19). However at each iteration of the loop (line 4), we have $\mathcal{I}_p \subseteq \mathcal{I}_c$.

When a local minimum is reached (line 6), we select one of the two following strategies : (1) call the CDCL component of the solver (lines 11–19) or (2) apply any other repair strategy, like novelty [29] or rsaps [22], to escape from such local minima (lines 8–9). The selection between the two strategies is done using a condition based on the search progress `SLSprogress` that will be explained later.

Each time the CDCL part of the solver is called (lines 11-20), a decision literal ℓ is chosen (line 11). Then unit propagation is performed (line 12). If it leads to a contradiction (line 13), a classical analysis is performed, the learnt clause γ is added to the learnt database Γ before back-jumping to the level j. If the learnt clause is empty, then the the formula is proven unsatisfiable (line 16). As long as the current partial interpretation $\mathcal{I}_p$ is not consistent, the process continues (see the loop line 13). At the end of this process, the consistent partial interpretation $\mathcal{I}_p$ is extended to a complete one $\mathcal{I}_c$ (line 20) and the local search component continues the search process. After each CDCL part, the partial interpretation $\mathcal{I}_p$ is modified (line 19). Consequently, the fixed part of the complete interpretation $\mathcal{I}_c$ is also modified. In this way, we derive a new strategy based on CDCL to escape from local minima.

Algorithm 3: SATHYS

Input: a CNF formula Σ
Output: SAT or UNSAT

1 $\Gamma \leftarrow \emptyset$;
2 $\mathcal{I}_p \leftarrow BCP(\Sigma)$;
3 $\mathcal{I}_c \leftarrow \mathcal{I}_p \cup completeInterpretation(\Sigma_{|\mathcal{I}_p})$;
4 **while** *(true)* **do**
5 **if** $(\Sigma_{|\mathcal{I}_c} = \emptyset)$ **then return** SAT;
6 **if** *a local minimum is reached* **then**
7 **if** $(\texttt{SLSprogress} > 0)$ **then**
8 $\ell \leftarrow chooseLitteralToEscapeLocalMinima(\Sigma_{|\mathcal{I}_p}, \mathcal{I}_c)$;
9 $flip(\mathcal{I}_c, \ell)$;
10 **else**
11 $\mathcal{I}_p \leftarrow \mathcal{I}_p \cup \{\ell\}$ with $\ell \notin \mathcal{I}_p$;
12 $\mathcal{I}_p \leftarrow \mathcal{I}_p \cup BCP((\Sigma \cup \Gamma)_{|\mathcal{I}_p})$;
13 **while** $(\emptyset \in (\Sigma \cup \Gamma)_{|\mathcal{I}_p})$ **do**
14 $\gamma{=}(\ell \vee \beta) \leftarrow analyzeConflict(\Sigma \cup \Gamma, \mathcal{I}_p)$;
15 $j \leftarrow level(\gamma, \mathcal{I}_p)$;
16 **if** $(\gamma = \emptyset)$ **then return** UNSAT;
17 $\Gamma \leftarrow \Gamma \cup \{\gamma\}$;
18 $\mathcal{I}_p \leftarrow \mathcal{I}_p^j$;
19 $\mathcal{I}_p \leftarrow \mathcal{I}_p \cup BCP((\Sigma \cup \Gamma)_{|\mathcal{I}_p})$;
20 $\mathcal{I}_c \leftarrow \mathcal{I}_p \cup (\mathcal{I}_c \setminus \mathcal{I}_p)$;
21 **else**
22 $x \leftarrow chooseLitteralToFlip(\Sigma_{|\mathcal{I}_p}, \mathcal{I}_c)$;
23 $flip(\mathcal{I}_c, \ell)$;
24 **if** *(restart())* **then**
25 $\mathcal{I}_p \leftarrow BCP(\Sigma \cup \Gamma)$;
26 $\mathcal{I}_c \leftarrow \mathcal{I}_p \cup completeInterpretation(\Sigma_{|\mathcal{I}_p})$;

This process is repeated until the next restart (line 24). As long as a model is not found or the unsatisfiability of the formula is not proven the solver restarts the previous process with a new initial complete interpretation (lines 25-26).

In the following, we introduce some improvements embedded in our solver SATHYS.

Calling the CDCL oracle. The behavior of our hybrid method heavily depends on the value of the variable `SLSprogress` (see Algorithm 3, line 7). We use a similar mechanism as it is done in [21]. When the local search engine reduces the MAXSAT value found in the current restart, this variable `SLSprogress` is increased. Our reasoning is that as long as SLS allows improvements i.e. increases the number of satisfied clauses, the execution of SLS is favored. Each time a local minimum is reached, the value of `SLSprogress` is decreased. By getting frequently stuck in local minima, it seems that the local search engine has a real difficulty to improve the current interpretation. In this case, we need to call the CDCL engine in order to escape from these local minima.

Activity based heuristic for CDCL. The heuristic used in the CDCL part of the Algorithm 3 is a slightly modified version of VSIDS [30]. Indeed, as usual, all weights of the variables are increased during the conflict analysis. Furthermore, when a local minimum is reached, we look for a boundary point and we increase the activity of the variable with one of its literals appearing in all the falsified clauses. In this way, and as explained above, our aim is to generate shorter resolution proofs.

Polarity of the decision literals. The polarity of the decision variable is known to be very important for the efficiency of SAT solvers. We propose here to use both engines in order to choose the best polarity for a given variable. At each restart, we store the interpretation associated to the current MAXSAT value and we modify it using progress saving [32]. Then when a given decision variable is chosen by CDCL, its polarity is taken from the last memorized complete interpretation with the best MAXSAT value. Indeed, we focus the search near the MAXSAT value, so near a solution by taking into account dependencies between variables.

4 Experimental validation

In this section, we provide an experimental validation of our hybrid solver SATHYS. Like most other solvers, we use SatElite in a preprocessing step [8]. Instances from the SAT'09 competition are used as a test set. They are divided into three different categories: crafted (281 instances), application (292) and random (570).

Let us note that these instances are carefully selected for the SAT competition because of their relevance. From the results of the last SAT competitions, one can have in mind that most of the state-of-the-art SAT solvers present very close performance in terms of the number of solved instances. For example, PRECOSAT [3] and GLUCOSE [1] were ex-aequo in terms of the number of solved instances in the application category SAT +UNSAT).

The experimentation has been conducted on the same cluster as for last SAT 2009 competition (Intel Xeon 3GHz under Linux with a RAM memory size of 2GB). The time limit (time-out) has been set to 1200 CPU seconds.

4.1 Effectiveness of the collaboration

First of all, our goal is to highlight that our hybrid scheme is really relevant. To measure the relevance of the different improvements introduced to SATHYS (described in the previous sections), we compared SATHYS with the following variants:

- SATHYS$_{nb}$: the VSIDS heuristic is not updated when a boundary point is discovered.
- SATHYS$_{cdclAtEachLM}$: instead of using the variable `SLSprogress` we call the CDCL solver at each local minimum.
- SATHYS$_{noPolarity}$: Literal polarity are not updated with the MAXSAT interpretation.

solver	Crafted	Application	Random	total
SATHYS	104	**148**	189	**441**
SATHYS_{nb}	103	144	**191**	438
$\text{SATHYS}_{cdclAtEachLM}$	101	141	8	250
$\text{SATHYS}_{noPolarity}$	**106**	142	188	436

Table 1. Comparison of different versions of the SATHYS solver. For each category, we provide the number of solved instances. The total number of solved instances on all categories is also given.

The Table 1 summarizes the results obtained by the four versions of SATHYS solver.

Not surprisingly, we can note that SATHYS_{nb} is the best one on random instances. Indeed, random instances are globally unsatisfiable i.e. the unsatisfiable core tends to include all the clauses of the original formula. As the role of the activities (VSIDS) is to focus the search on the most important part of formula, this is clearly not relevant in case of random instances. Also, the boundary points are useless for this kind of instances.

Concerning $\text{SATHYS}_{cdclAtEachLM}$, this version is the worst one. Indeed, too many calls of CDCL consumes clearly too much time. Moreover, as CDCL approaches are not well suited for random instances, calling CDCL penalizes SATHYS. We can note that the number of calls to the CDCL engine has an important impact on the performances of SATHYS. Consequently, fine-tuning the local search progress (`SLSprogress`) is crucial for the efficiency of our hybrid approach.

Concerning $\text{SATHYS}_{noPolarity}$, this version obtains good performance on the crafted category. However, literals polarity seems to be a relevant criteria for application instances considered as the most important category by the SAT community.

In summary, the cooperation scheme designed in SATHYS and described in the previous section improves the robustness and the efficiency of our hybrid solver.

4.2 Comparison

In this section, we compare our solver against some of the well known SAT solvers. Many of them are considered as the state-of-the-art SAT solvers in at least one category of instances. We can note that four solvers considered in our comparative experiments have obtained a gold medal at the SAT competition in 2009 in different categories.

- Two SLS methods: WSAT [34] with rnovelty strategy and ADAPTG2 [27].
- Two hybrid approaches: HYBRIDGM [2] and HINOTOS[26].
- Five complete approaches: MINISAT [9], GLUCOSE [1], PRECOSAT [3], RSAT [32], and CLASP [13].
- A portfolio solver : SATZILLA_I [37].

Table 2 summarizes the obtained results. Let us start this comparison with local search solvers. We can note that they perform better than SATHYS only in the random category of instances. This is not surprising since they are particularly suited to this kind of instances. On the other categories of instances, local search based techniques obtain bad results. As underlined during the SAT'09 competition, the main weak point of local search based solvers resides in their inefficiency on the application category. Improving local search based techniques on application category remains a very challenging issue.

	Crafted			Application			Random			
	total	(sat	unsat)	total	(sat	unsat)	total	(sat	unsat)	total
ADAPTG2	68	(68	0)	8	(8	0)	**294**	(**294**	0)	375
GNOVELTY+	54	(54	0)	7	(7	0)	281	(281	0)	342
SATHYS	104	(71	33)	148	(63	85)	189	(189	0)	**441**
HYBRIDGM	51	(51	5)	0	(5	0)	294	(294	0)	350
HINOTOS	105	(69	36)	107	(39	68)	77	(65	11)	288
MINISAT	99	(72	27)	152	(59	93)	3	(3	0)	254
GLUCOSE	114	(75	39)	152	(54	98)	17	(17	0)	266
PRECOSAT	122	(81	41)	**164**	(**65**	**99**)	2	(2	0)	288
RSAT	105	(71	34)	143	(53	90)	5	(5	0)	253
CLASP	**131**	(78	**53**)	138	(53	85)	84	(66	18)	353
SATZILLA_I	128	(**86**	42)	142	(60	82)	145	(90	**55**)	415

Table 2. SATHYS vs. some other SAT solvers. For each category and each solver, the number of solved instances is provided.

As the skeleton of our solver is an SLS technique, where the CDCL component can be considered as a strategy for escaping from local minima, the good performances of our hybrid solver provides real advances on this important issue.

Comparatively with other hybrid solvers, SATHYS is the best one w.r.t. in the application category. HYBRIDGM seems to be tuned for random instances whereas HINOTOS and SATHYS are comparable on crafted category. The results show clearly that our hybrid solver outperforms all the hybrid approaches.

SATHYS is very competitive with respect to the sate-of-the-art CDCL solvers in both crafted and application categories. To our best knowledge, it is the first time that an hybrid solver is able to obtain such promising results. On random instances, classical CDCL solvers present a very bad behavior. On this last category, clause learning and back-jumping are useless.

Interestingly enough, SATHYS is also very competitive with respect to the portfolio based solver SATZILLA. This portfolio uses CDCL, local search and lookahead solvers. In this approach, the different solvers are called independently.

To summarize, SATHYS is the first competitive solver on all the three categories of SAT instances (crafted, application and random). It is also the most robust one: it obtains the best overall results in terms of the total number of solved instances and it is often close to the first rank on each category.

The previous table provides information about the number of solved instances in the considered time limit. Also, we present the classical scatter plot in order to compare SATHYS with one of the best known hybrid solver HINOTOS, PRECOSAT (the state-of-the-art solver on application category), CLASP (the best solver on crafted category) and SATZILLA (it obtains the second best overall results behind SATHYS). In these curves, the y-axis represents the time (in log scale) of SATHYS and the x-axis the time of the other solver. So, a dot below the diagonal represents an instance where SATHYS is faster than the other solver. This is shown in Figure 1.

Analyzing these figures, we can conclude that SATHYS outperforms HINOTOS. However our approach is slightly slower than PRECOSAT, which is one of the best CDCL solvers. However, SATHYS solves more instances than PRECOSAT. Comparatively with CLASP, we can observe that our method obtains similar performance on application and

crafted instances, but SATHYS is better on random instances. Finally, it is impossible to differentiate SATZILLA and SATHYS in terms of CPU time.

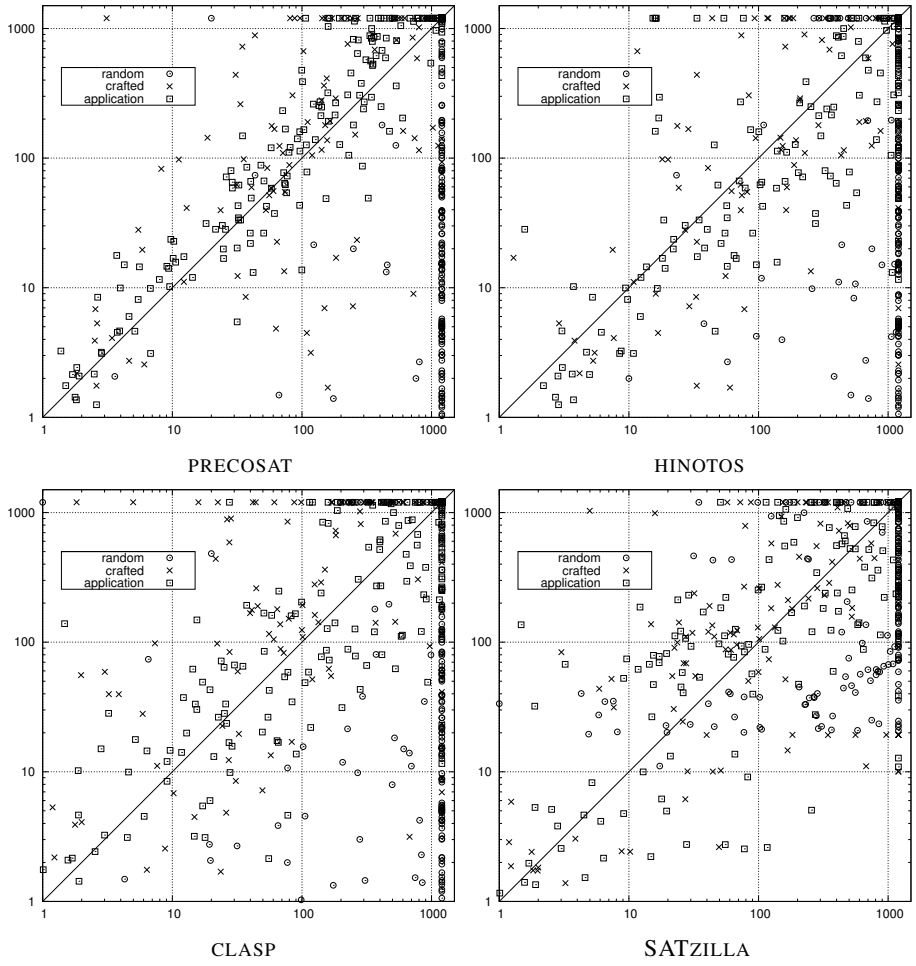

Fig. 1. Scatter plot: SATHYS vs. {PRECOSAT,HINOTOS,CLASP,SATZILLA }

Finally, we also want to hightlight that our solver is competitive on difficult hard SAT and UNSAT instances from crafted and application categories. Table 3 provides results on a selection of instances. Because of lack of space we do not report the results for each solver. However, to be as fair as possible, we also report the best result obtained in the competition over the 50 submitted solvers (BEST column). SATHYS is efficient on SAT as well as UNSAT instances.

In conclusion of this experimental comparison, SATHYS is a very efficient solver in three categories (crafted, application and random) and consequently, it is the most

	SAT	BEST	SATHYS	HINOTOS	GLUCOSE	PRECOSAT	CLASP	SATZILLA
q_query_3_L100_coli	N	183	677	–	577	**414**	780	–
post-c32s-col400-16	N	66	247	380	714	141	**66**	125
countbitsarray02_32	N	926	1100	–	**926**	–	–	–
maxxororand032	N	579	**768**	–	–	–	–	–
minand128	N	16	71	219	26	26	**23**	212
rpoc_xits_08_UNSAT	N	154	1036	1115	398	**159**	–	589
gt-ordering-unsat-gt-060	N	15	171	–	–	**149**	–	–
9dlx_vliw_at_b_iq3	N	430	1137	–	–	–	1095	**430**
gss-19-s100	Y	38	641	–	611	–	–	**38**
UCG-20-5p1	Y	413	835	–	–	**537**	–	–
UTI-15-10p1	Y	392	935	–	**392**	–	–	1140
gt-ordering-sat-gt-040	Y	15	**6**	–	–	149	–	–
em_9_3_5_exp	Y	18	288	209	**51**	181	140	276
ndhf_xits_20	Y	1.6	180	–	–	249	–	75
partial-10-15-s	Y	228	**1092**	–	–	–	–	–
vmpc_30	Y	18	825	–	–	292	**159**	551
velev-pipe-sat-1.0-b7	Y	14	294	–	–	343	–	**87**
new-difficult-21-168-19-90	Y	0.1	**141**	–	–	–	370	416
mod3block_3vars_9gates...	Y	5.7	**4.8**	–	126	63	26	24
rbsat-v945c61409g10	Y	21	362	–	898	**147**	150	933
instance_n8_i9_pp	Y	41	**115**	454	1117	800	–	421

Table 3. Highlight results on a selection of instances. Results are reported in seconds.

robust solver. From the above results, we can consider SATHYS as the first hybrid solver that brings real advances to three of the ten challenges proposed in [35, 24]:

- Challenge 5: *Design a practical stochastic local search procedure for proving unsatisfiability*;
- Challenge 6: *Improve stochastic local search on structured problems by efficiently handling variable dependencies*;
- Challenge 7: *Demonstrate the successful combination of stochastic search and systematic search techniques, by the creation of a new algorithm that outperforms the best previous examples of both approaches.*

5 Related work

We presents here the most noticeable approaches combining local search and DPLL based ones. They can be classified in three different categories depending on which of the two solvers is considered as the main core of the hybrid solver.

The first category includes those using DPLL as the main solver and SLS as an oracle. In [5], SLS is used in a preprocessing step to derive a static variable ordering for DPLL. Weights representing the number of times each clause is falsified during the local search pretreatment is computed. The variable occurring most often in clauses with higher weights is selected first. The approach proposed in [28] extends the previous one, while invoking SLS at each node of the search tree. The SLS oracle is used to both extend the current partial assignment to a model or to select the next variable to assign according to similar clause weighting process. Similarly to [28], in [11], the authors introduce some conditions to reduce the number of calls to SLS. The variables are

dynamically ordered according to local-search statistics. In [19], an hybrid constraint solving schema which retains some systematicity of constructive search while incorporating the heuristic guidance and lack of commitment to variable assignment of local search. The proposed method backtracks through a space of complete but possibly inconsistent solutions while supporting the freedom to move arbitrarily under heuristic guidance. hybridGM is an incomplete SAT solver proposed in [2] focusses the DPLL-search around local minima with only one unsatisfied clause.

If a formula is satisfiable, chances are to find a satisfying assignment around such minima. hybridGM's DPLL component then completely checks these areas of the search space. SATUN [12] an extension of hybridGM performs local search as it is done by hybridGM, but for a limited amount of time. The limit is set in a way that gives local search a reasonable chance to find a satisfying assignment. In case the formula is satisfiable, this will result in a performance competitive with the hybrid's SLS component. As soon as the limit is reached, the formula is expected to be unsatisfiable. SATUN's DPLL component will then perform a search on the complete search space of the formula to confirm that assumption.

The second category considers SLS as the main core of the hybrid solver. Among these approaches, we can cite the approach proposed in [18], where DPLL is used to derive implications between literals. These implications are used to both simplify the formula and to compute dependency relations between literals used during the local search phase.

In [23] the algorithm starts with a partial or complete interpretation. At each step constraint propagation is applied. In case of conflict, a nogood is learnt and local search is applied to repair the current partial interpretation. Otherwise, the current consistent interpretation is extended in a classical way. In [25], the authors propose an hybrid strategy based on shared memory, ideally suited for multi-core processor architectures. They particularly show that DPLL can provide highly effective guidance for a local search style solver for the MAXSAT problem. More precisely, DPLL shares its current partial assignment with SLS and a flip is not allowed if the variable is assigned with the same polarity by DPLL. The two solvers are run simultaneously on two different cores.

Finally, the last category contains hybrid solvers where both engines play an equal role. The hybrid solver HBISAT [10] and its extended and improved version hinotos [26] belong to this category. In both approaches, a local search is used to identify a subset of clauses to be passed to a DPLL SAT solver through an incremental interface. In other words, the guided local search identifies incremental sets of clauses that are hard, and these clauses are subsequently added to the clause database of the DPLL-based solver. In addition, the solution obtained by the DPLL solver on the subset of clauses is fed back to the local search solver to jump over any locally optimal points.

6 Conclusion

In this paper a novel integration of SLS and CDCL based SAT solvers is introduced. This hybrid solver represents an original combination of both engines. The two components heavily cooperate towards proving the satisfiability or unsatisfiability of the SAT for-

mula. Such strong cooperation lies in the exploitation of SLS for directing CDCL search towards unsatisfiability proof; and CDCL for escaping from SLS local minima. SATHYS, the resulting method, obtains very good results on a wide range of instances taken from the last competitions. These results show important performance improvements of the state-of-the-art SLS and hybrid SLS solvers particularly on crafted and application category. More interestingly, SATHYS significantly reduces the performance gap between SLS and CDCL solvers while becoming extremely competitive with most of the state-of-the arts CDCL solvers on application instances. Consequently, SATHYS can be considered as the first successful hybrid SAT solver. A future work, we first plan to investigate how conflict driven clause learning can be adapted to local search based techniques. Secondly, as the cooperation scheme between the two solvers integrated in SATHYS is proven to be efficient, we plan to design a new parallel version of SATHYS in order to benefit from the computational resources of multicore based architectures.

References

1. G. Audemard and L. Simon. Predicting learnt clauses quality in modern SAT solvers. In *proceedings of International Joint Conference on Artificial Intelligence*, pages 399–404, 2009.
2. A. Balint, M. Henn, and O. Gableske. A novel approach to combine a SLS- and DPLL-solver for the satisfiability problem. In *proceedings of SAT*, 2009.
3. A. Biere. PicoSAT essentials. *Journal on Satisfiability*, 4:75–97, 2008.
4. A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, February 2009.
5. J. Crawford. Solving satisfiability problems using a combination of systematic and local search. In *Second Challenge on Satisfiability Testing organized by Center for Discrete Mathematics and Computer Science of Rutgers University*, 1996.
6. Adnan Darwiche and Knot Pipatsrisawat. *Complete Algorithms*, chapter 3, pages 99–130. IOS Press, 2009.
7. M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communication of ACM*, 5(7):394–397, 1962.
8. N. Eén and A. Biere. Effective preprocessing in SAT through variable and clause elimination. In *proceedings of SAT*, pages 61–75, 2005.
9. N. Een and N. Sörensson. An extensible SAT-solver. In *proceedings of SAT*, pages 502–518, 2003.
10. L. Fang and M. Hsiao. A new hybrid solution to boost SAT solver performance. In *proceedings of DATE*, pages 1307–1313, 2007.
11. B. Ferris and J. Fröhlich. Walksat as an informed heuristic to DPLL in SAT solving. Technical report, CSE 573 : Artificial Intelligence, 2004.
12. Oliver Gableske and Julian Rüth. Satun: A complete hybrid sat solver. SAT2010 paper draft, 2010.
13. M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub. Conflict-driven answer set solving. In *Proceedings IJCAI'07*, pages 386–392, 2007.
14. Eugene Goldberg. Boundary points and resolution. In Oliver Kullmann, editor, *SAT*, volume 5584 of *Lecture Notes in Computer Science*, pages 147–160. Springer, 2009.
15. C. Gomes, B. Selman, and H. Kautz. Boosting combinatorial search through randomization. In *AAAI/IAAI*, pages 431–437, 1998.
16. E. Gregoire, B. Mazure, and C. Piette. Extracting MUSes. In *proceedings of ECAI*, pages 387–391, 2006.

17. E. Gregoire, B. Mazure, and C. Piette. Local-search extraction of muses. *Constraints*, 12(3):325–344, September 2007.
18. D. Habet, C.M. Li, L. Devendeville, and M. Vasquez. A hybrid approach for sat. In *Proceedings of Principles and Practice of Constraint Programming - CP*, pages 172–184, 2002.
19. W. Havens and B. Dilkina. A hybrid schema for systematic local search. In *Canadian Conference on AI*, pages 248–260, 2004.
20. E. Hirsch and A. Kojevnikov. Unitwalk: A new SAT solver that uses local search guided by unit clause elimination. *Annals of Mathematical and Artificial Intelligence*, 43(1):91–111, 2005.
21. H. Hoos. An adaptive noise mechanism for walksat. In *proceedings of AAAI*, pages 655–660, 2002.
22. Frank Hutter, Dave A. D. Tompkins, and Holger H. Hoos. Scaling and probabilistic smoothing: Efficient dynamic local search for sat. In *CP*, volume 2470 of *Lecture Notes in Computer Science*, pages 233–248. Springer, 2002.
23. N. Jussien and O. Lhomme. Local search with constraint propagation and conflict-based heuristics. In *AAAI/IAAI*, pages 169–174, 2000.
24. Henri Kautz and Bart Selman. Ten challenges redux: Recent progress in propositional reasoning and search. In *Proceedings of the Ninth International Conference on Principles and Practice of Constraint Programming (CP'03)*, pages 1–18, September 2003.
25. Lukas Kroc, Ashish Sabharwal, Carla P. Gomes, and Bart Selman. Integrating systematic and local search paradigms: A new strategy for maxsat. In *IJCAI*, pages 544–551, 2009.
26. F. Letombe and J. Marques-Silva. Improvements to hybrid incremental sat algorithms. In *proceedings of SAT*, pages 168–181, 2008.
27. C.M. Li, W. Wei, and H. Zhang. Combining adaptive noise and look-ahead in local search for sat. In *proceedings of SAT*, pages 121–133, 2007.
28. B. Mazure, L. Saïs, and E. Grégoire. Boosting complete techniques thanks to local search methods. *Ann. Math. Artif. Intell.*, 22(3-4):319–331, 1998.
29. D. McAllester, B. Selman, and H. Kautz. Evidence for invariants in local search. In *proceedings of AAAI*, pages 321–326, 1997.
30. M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *proceedings of DAC*, pages 530–535, 2001.
31. D.N. Pham, J. Thornton, and A. Sattar. Building structure into local search for sat. In *proceedings of IJCAI*, pages 2359–2364, 2007.
32. Knot Pipatsrisawat and Adnan Darwiche. A lightweight component caching scheme for satisfiability solvers. In *SAT*, pages 294–299, 2007.
33. B. Selman and H. Kautz. An empirical study of greedy local search for satisfiability testing. In *proceedings of AAAI*, pages 46–51, 1993.
34. B. Selman, H. Kautz, and B. Cohen. Noise strategies for improving local search. In *proceedings of AAAI*, pages 337–343, 1994.
35. B. Selman, H. Kautz, and D. McAllester. Ten challenges in propositional reasoning and search. In *proceedings of IJCAI*, pages 50–54, 1997.
36. J. Marques Silva and K. Sakallah. Grasp - a new search algorithm for satisfiability. In *ICCAD*, pages 220–227, 1996.
37. L. Xu, F. Hutter, H. Hoos, and K Leyton-Brown. Satzilla: Portfolio-based algorithm selection for sat. *Journal of Artificial Intelligence Research*, 32:565–606, 2008.
38. L. Zhang, C. Madigan, M. Moskewicz, and S. Malik. Efficient conflict driven learning in boolean satisfiability solver. In *proceedings of ICCAD*, pages 279–285, 2001.