Learning in local search

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

Learning in local search *

Gilles Audemard Jean-Marie Lagniez Bertrand Mazure 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 learning based local search approach for propositional satisfiability is presented. It is based on an original adaptation of the conflict driven clause learning (CDCL) scheme to local search. First an extended implication graph for complete assignments of the set of variables is proposed. Secondly, a unit propagation based technique for building and using such implication graph is designed. Finally, we show how this new learning scheme can be integrated to the state-of-the-art local search solver WSAT. Interestingly enough, the obtained local search approach is able to prove unsatisfiability. Experimental results show very good performances on many classes of SAT instances from the last SAT competitions.

1 Introduction

The SAT problem, namely the issue of checking whether a set of Boolean clauses is satisfiable or not, is a central issue in many computer science and artificial intelligence domains, like e.g. theorem proving, planning, non-monotonic reasoning, VLSI correctness checking. These last two decades, many approaches have been proposed to solve large SAT instances, based on logically complete or incomplete. Both local-search techniques (e.g. [23, 22, 13]) and elaborate variants of the Davis-Putnam-Loveland-Logemann DPLL procedure [5] (e.g. [19, 7]), called modern SAT solvers, can now solve many families of hard SAT instances. These two kinds of approaches present complementary features and performances. Modern SAT solvers are particularly efficient on the industrial SAT category while local search is better on random SAT instances. Consequently, enhancing the performances of local search approaches to the level of modern modern SAT solvers on the industrial category is really an important challenge. This challenge is stated by Bart Selman *et al.* in 1997 [24] (challenge number 6 ”Improve stochastic local search on structured problems by efficiently handling variable dependencies”). Another important issue (challenge number 5) also identified in [24], is to design a practical stochastic local search procedure for proving unsatisfiability. The goal of this work, is to make a step towards the resolution of these two challenges. Our aim is to enhance the performances of SLS techniques on industrial SAT instances and to make such techniques able to prove unsatisfiability.

Let us first recall that some attempts towards these directions have been made recently. In [20] functional dependencies recognized using Ostrowski et al approach [12] have been exploited in local search based techniques leading to interesting improvements particularly on crafted SAT instances (e.g. parity 32 instances). In [21, 3] new local search for proving unsatisfiability have been proposed. In [4], authors propose a stochastic local search solver which add resolvents between two clauses in order to leave local minimum. Such method has been improved by [8, 25]. However, all these different approaches are still premature and progress is needed for solving both challenges.

To go further in this direction, we propose to integrate learning from conflict, one of the most important component behind the efficiency of modern SAT solvers, to local search techniques. However, one of the main difference between DPLL-like and local search techniques that make such adaptation very challenging rises in the way search space is explored by both approaches. In DPLL one searches among partial assignments whereas in SLS search is done on complete assignments. Even if such difference is important, the two search paradigms actually admit many common features. One can connect for example the activity based heuristics (VSIDS) with weighting constraint as done in the break-out local search method [18], restarts in modern SAT solvers with tries in WSAT like algorithms.

In this paper, we propose to improve the methods proposed in [4, 8, 25] by integrating Conflict Driven Clause Learning (CDCL) with implication graph [17, 26] to the

*supported by ANR UNLOC project ANR08-BLAN-0289-01

stochastic local search framework. The goal is twofold. First, similarly to [4, 8, 25], we exploit such learning component as a strategy to escape from local minima. However our approach is more general as the conflict clause is generated using an implication graph, whereas in [4, 8, 25], such a clause is obtained using only one resolution step between two clauses. Secondly, like previous methods, the addition of new learnt clauses makes the local search solver able to prove unsatisfiability.

The rest of this paper is organized as follows. In section 2, after the introduction of some preliminary definitions and notations, local search algorithms and classical SAT conflict analysis are presented. In section 3, we describe our extension of the implication graph to complete assignments and introduce a unit propagation based approach for building such graph. Finally, we integrate such conflict analysis in WSAT-like algorithm [23]. In section 4 experimental results of our proposed approach are presented before concluding.

2 Preliminary definitions and technical background

2.1 Definitions

Let us give some necessary definitions and notations. Let $V = \{x_1 ... x_n\}$ be a set of boolean variables, a literal ℓ is a variable x_i or its negation $\overline{x_i}$. A clause is a disjunction of literals $c_i = (\ell_1 \vee \ell_2 ... \vee \ell_{n_i})$. A unit clause is a clause with only one literal. A formula Σ is in conjunctive normal form (CNF) if it is a conjunction of clauses $\Sigma = (c_1 \wedge c_2 ... \wedge c_m)$. The set of literals appearing in Σ is denoted $\mathcal{V}_\Sigma$. An interpretation $\mathcal{I}$ of a formula Σ associates a value $\mathcal{I}(x)$ to variables in the formula. An interpretation is *complete* if it gives a value to each variable $x \in \mathcal{V}_\Sigma$, otherwise it is said *partial*. A clause, a CNF formula and an interpretation can be conveniently represented as sets. A *model* of a formula Σ, denoted $\mathcal{I} \models \Sigma$, is an interpretation $\mathcal{I}$ which satisfies the formula Σ i.e. satisfies each clause of Σ. Then, we can define the SAT decision problem as follows: is there an assignment of values to the variables so that the CNF formula Σ is satisfied?

Let us introduce some additional notations.

- An empty clause is represented by $\perp$ and is unsatisfiable;
- the negation of a set of literals $\Gamma = \{\ell_1, \ell_2, \ldots, \ell_n\}$ is denoted $\overline{\Gamma}$ ans is equal to $\{\overline{\ell_1}, \overline{\ell_2}, \ldots, \overline{\ell_n}\}$;
- $\Sigma_{|\ell}$ denotes the formula Σ simplified by the assignment of the literal ℓ to true. This notation is extended to interpretations: Let $\mathcal{P} = \{\ell_1, ..., \ell_n\}$ be an interpretation, $\Sigma_{|\mathcal{P}} = (...(\Sigma|_{\ell_1})...|_{\ell_n})$;
- Σ^* denotes the formula Σ simplified by unit propagation;
- $\models_*$ denotes logic deduction by unit propagation: $\Sigma \models_* \ell$ means that the literal ℓ is deducted by unit propagation from Σ i.e. $\perp \in (\Sigma \wedge \overline{\ell})^*$. One notes $\Sigma \models_* \perp$ if the formula is unsatisfiable by unit propagation;
- $\eta[x, c_i, c_j]$ denotes the *resolvent* between a clause c_i containing the literal x and c_j a clause containing the opposite literal $\overline{x}$. In other words $\eta[x, c_i, c_j] = c_i \cup c_j \backslash \{x, \overline{x}\}$. A resolvent is called *tautological* when it contains opposite literals.

2.2 Local Search Algorithms

Local search algorithms for SAT problems use a stochastic walk over complete interpretations of Σ. At each *step* (or *flip*), they try to reduce the number of unsatisfiable clauses (usually called a descent). The next complete interpretation is chosen among the neighbours of the current one (they differ only on one literal value). A local minimum is reached when no descent is possible. One of the key point of stochastic local search algorithms is the method used to escape from local minimum. For lack of space, we cannot provide a general algorithm of local search solver. However, a modified version can be seen in algorithm 1. For more details, the reader will refer to [14].

2.3 Conflict Analysis and Implication Graph

Now, we introduce a fundamental data structure, called implication graph, used by complete CDCL solvers (*Conflict Driven Clause Learning*) for conflict analysis, nogood deduction and backjumping. Some of the following notations have been introduced in [2].

A typical branch of a CDCL solver can be seen as sequences of decision-propagation. At decision level i, the current partial interpretation $\mathcal{I}$ is of the form $\langle (x_k^i), x_{k_1}^i, x_{k_2}^i, \ldots, x_{k_{n_k}}^i \rangle$ where the first literal x_k^i corresponds to the decision literal x_k assigned at level i and each $x_{k_j}^i$ for $1 \leq j \leq n_k$ corresponds to a propagated (unit) literal. Whenever a literal y is propagated, we keep a reference to the clause at the origin of the propagation of y, which we denote $\overrightarrow{cla}(y)$. Of course It can exist more than one such clause, however we are taking into account only one of them, usually the first encountered one. The clause $\overrightarrow{cla}(y)$ has, in this case, the form $(x_1 \vee \cdots \vee x_n \vee y)$ where every literal x_i is false under the current partial assignment ($\mathcal{I}(x_i) = false, \forall i \in 1 \ldots n$), while $\mathcal{I}(y) = true$. When a literal y is not obtained by propagation but comes from a decision, $\overrightarrow{cla}(y)$ is undefined, which we note for convenience $\overrightarrow{cla}(y) = \perp$. When $\overrightarrow{cla}(y) \neq \perp$, we denote by $exp(y)$

the set $\{\overline{x} \mid x \in \overrightarrow{cla}(y) \setminus \{y\}\}$, called set of *explanations* of y. In other words, if $\overrightarrow{cla}(y) = (x_1 \vee \cdots \vee x_n \vee y)$, then the explanations are the literals $\overline{x_i}$ with which $\overrightarrow{cla}(y)$ becomes the unit clause $\{y\}$. When $\overrightarrow{cla}(y)$ is undefined we define $exp(y)$ as the empty set.

The implication graph is a directed acyclic graph, it allows a representation of decision-propagations sequences. In such a graph, each vertex is associated to a literal and incoming edges of a vertex are the set of its explanations. Formally, we have:

Definition 1 (Implication graph) *Let Σ be a CNF formula and $\mathcal{I}_p$ be a partial interpretation. An implication graph associated to Σ, $\mathcal{I}_p$ and exp is $\mathcal{G}_\Sigma^{\mathcal{I}_p} = (\mathcal{N},\mathcal{A})$ where:*

1. *$\mathcal{N} = \{x | x \in \mathcal{I}_p\}$, i.e. there is exactly one node for every literal, decision or implied;*
2. *$\mathcal{A} = \{(x, y) | x \in \mathcal{I}_p, y \in \mathcal{I}_p, x \in exp(y)\}$.*

Example 1 *Let $\Sigma = \{\phi_1, ..., \phi_{11}\}$ be a CNF formula such that.*

$\phi_1 : (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})$	$\phi_2 : (x_1 \vee \overline{x_4} \vee \overline{x_5})$
$\phi_3 : (x_2 \vee \overline{x_1})$	$\phi_4 : (x_4 \vee \overline{x_7} \vee \overline{x_6})$
$\phi_5 : (x_3 \vee \overline{x_5})$	$\phi_6 : (x_5 \vee \overline{x_7})$
$\phi_7 : (x_6 \vee \overline{x_8})$	$\phi_8 : (x_7 \vee \overline{x_8})$
$\phi_9 : (x_8 \vee \overline{x_4})$	$\phi_{10} : (x_1 \vee \overline{x_8})$
$\phi_{11} : (x_7 \vee \overline{x_9})$	

Let $\mathcal{I}_p$ be the following partial interpretation $\mathcal{I}_p = \{\langle(x_2^1)\rangle\langle(x_6^2)\rangle\langle(x_9^3)\ x_7^3\ x_4^3\ x_5^3\ x_3^3\ x_1^3\ \overline{x_1^3}\rangle\}$. The Current decision level is 3 and $\Sigma_{|\mathcal{I}_p} \models_* \perp$. Figure 1 represents the implication graph $\mathcal{G}_\Sigma^{\mathcal{I}_p}$ associated to Σ, $\mathcal{I}_p$ and exp (the set of explanations).

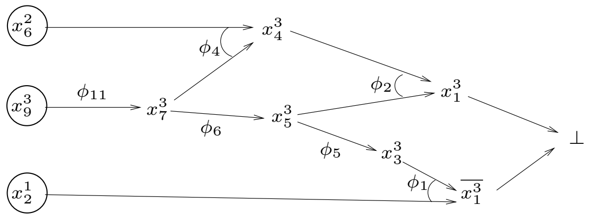


Figure 1. Implication graph (example 1)

As mentioned above, an implication graph allows nogoods extraction. These nogoods are built by a traversal of the implication graph starting from the top (the falsified clause). Different kinds of nogoods can be generated. One of the most used and efficient learning scheme is the first UIP (Unique Implication Point). In the following example, we show a construction of a first UIP. For more details and a formal presentation, the reader can refer to [17, 26].

Example 2 *Consider again example 1 and the same implication graph (Figure 1) associated to interpretation $\mathcal{I}_p$. To generate the first UIP, we perform resolution (starting from the conflict) between clauses encoded in the implication graph (implications):*

- $\beta_1 = \eta(x_1^3, \phi_1, \phi_2) = \overline{x_2^1} \vee \overline{x_3^3} \vee \overline{x_4^3} \vee \overline{x_5^3}$
- $\beta_2 = \eta(x_3^3, \beta_1, \phi_5) = \overline{x_2^1} \vee \overline{x_4^3} \vee \overline{x_5^3}$
- $\beta_3 = \eta(x_5^3, \beta_2, \phi_6) = \overline{x_2^1} \vee \overline{x_4^3} \vee \overline{x_7^3}$
- $\beta = \eta(x_4^3, \beta_3, \phi_4) = \overline{x_2^1} \vee \overline{x_6^2} \vee \overline{x_7^3}$

The clause β contains only one literal from the last decision level (here level 3). Then, the process ends with the clause β, usually called an asserting clause. The literal $\overline{x_7^3}$ is an asserting literal, and the node x_7 in the implication graph (see figure 1) is the first UIP.

3 Local search and conflict analysis

3.1 Conflict graph definition

In section 2.2, we mentioned that one of the key points behind the efficiency of local search algorithms is undoubtedly the strategy used to escape from local minima. In [4, 8, 25], authors proposed to add clauses obtained by resolution in order to leave such minimum. We propose to improve such a strategy by exploiting the implication graph built from a complete interpretation to generate and add nogoods to the clauses data base.

However, in local search framework, one has to deal with complete interpretations. Defining an implication graph in this case is clearly challenging. Indeed, there is no notion of levels or unit propagated literals. Furthermore, a complete interpretation can falsify more than one clause. In this section, we propose a new definition of implication graphs in stochastic local search framework. Before, we give some necessary definitions. Let us consider a CNF formula Σ and a complete interpretation $\mathcal{I}_c$. We say that the literal ℓ satisfies (resp. falsifies) a clause $\beta \in \Sigma$ under $\mathcal{I}_c$ if $\ell \in \mathcal{I}_c \cap \{x \mid x \in \beta\}$ (resp. $\ell \in \mathcal{I}_c \cap \{x \mid \overline{x} \in \beta\}$). We note $\mathcal{L}^+_{\mathcal{I}_c}(\beta)$ (resp. $\mathcal{L}^-_{\mathcal{I}_c}(\beta)$), the set of literals satisfying (resp. falsifying) a clause β under $\mathcal{I}_c$. The following definitions were introduced in [11].

Definition 2 (once-satisfied clause) *A clause β is said* once-satisfied *by an interpretation $\mathcal{I}_c$ on literal z if $\mathcal{L}^+_{\mathcal{I}_c}(\beta) = \{z\}$.*

Definition 3 (critical and linked clauses) *Let $\mathcal{I}_c$ be a complete interpretation. A clause α is critical w.r.t. $\mathcal{I}_c$ if $|\mathcal{L}^+_{\mathcal{I}_c}(\alpha)| = 0$ (α is falsified) and $\forall \ell \in \alpha$, $\exists \alpha' \in \Sigma$ with*

$\bar{\ell} \in \alpha'$ *and* α' *is an once-satisfied clause. Clauses* α' *are linked to* α *for the interpretation* $\mathcal{I}_c$.

Example 3 *Let* $\Sigma = (\overline{a} \vee \overline{b} \vee \overline{c}) \wedge (a \vee \overline{b}) \wedge (b \vee \overline{c}) \wedge (c \vee \overline{a})$ *be a formula and* $\mathcal{I}_c = \{a, b, c\}$ *an interpretation. The clause* $\alpha_1 = (\overline{a} \vee \overline{b} \vee \overline{c})$ *is critical. The other clauses of* Σ *are linked to* α_1 *for* $\mathcal{I}_c$.

Now, we can define conflict graph for complete interpretations.

Definition 4 (Conflict graph on z) *Let* Σ *be a CNF formula,* $\mathcal{I}_c$ *a complete interpretation falsifying* Σ*. Consider two clauses of* Σ*,* $\beta = \{\beta_1, \ldots, \beta_k, z\}$ *falsified by* $\mathcal{I}_c$ *and* $\gamma = \{\gamma_1, \ldots, \gamma_l, \overline{z}\}$ *once-satisfied on* $\overline{z}$*, the conflict graph* $\mathcal{G}^z_{\Sigma,\mathcal{I}_c} = (\mathcal{N}, \mathcal{A})$ *is constructed in the following way:*

1. - $\{z, \overline{z}, \perp\} \subseteq \mathcal{N}$;
   - $\{\overline{\gamma_1}, \ldots, \overline{\gamma_l}\} \subseteq \mathcal{N}$;
   - $\{\overline{\beta_1}, \ldots, \overline{\beta_k}\} \subseteq \mathcal{N}$;
2. - $\{(z, \perp), (\overline{z}, \perp)\} \subseteq \mathcal{A}$;
   - $\{(\overline{\beta_1}, z), \ldots, (\overline{\beta_k}, z)\} \subseteq \mathcal{A}$;
   - $\{(\overline{\gamma_1}, \overline{z}), \ldots, (\overline{\gamma_k}, \overline{z})\} \subseteq \mathcal{A}$;
3. $\forall x \in \mathcal{N}$*, if* $x \neq z$ *and* $\alpha = \bigwedge\{y \in \mathcal{N} \mid (y, x) \in \mathcal{A}\} \not\models \perp$ *then* $\overline{\alpha} \vee x \in \Sigma$ *is once-satisfied on* x.

Since for a given literal z, clauses like β and γ are not unique, then the conflict graph is not unique too.

Example 4 *Consider again example 1. Let* $\mathcal{I}_c$ *be a complete interpretation such that* $\mathcal{I}_c = \{x_1, \ldots, x_9\}$*. Figure 2 represents the conflict graph* $\mathcal{G}^{x_1}_{\Sigma,\mathcal{I}_c}$ *constructed on variable* x_1.

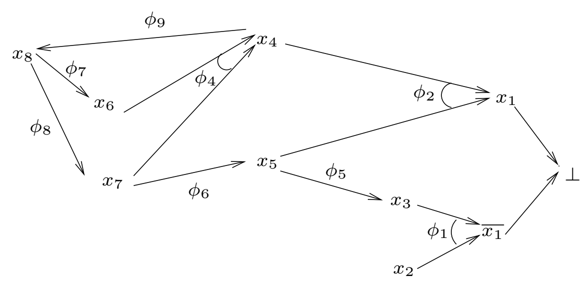


Figure 2. Conflict graph constructed on variable x_1 (example 4)

It is important to note that the existence of such a graph is not ensured. The following proposition gives necessary and sufficient conditions for the construction of the conflict graph.

Proposition 1 *Let* Σ *be a CNF formula,* $\mathcal{I}_c$ *a complete interpretation. The conflict graph* $\mathcal{G}^x_{\Sigma,\mathcal{I}_c}$ *exists if and only if* x *and* $\overline{x}$ *appear respectively in a falsified clause and in a once-satisfied one.*

Proof The proof of this proposition follows directly from the definition of the conflict graph.

Corollary 1 *Let* $\alpha \in \Sigma$ *be a critical clause.* $\forall x \in \alpha$*, it is possible to construct a conflict graph on* x.

Proof By definition of a critical clause $\alpha \in \Sigma$, we have $\forall x \in \alpha$, $\exists \beta \in \Sigma$ once-satisfied on x. From Proposition 1, it is obvious that $\forall x \in \alpha$, it exists a conflict graph on x.

As shown in [11], in a local minima all falsified clauses are necessarily critical. The corollary 1 ensures that, in this case, it is always possible to build a conflict graph. Then, we can use it in order to leave such local minimum. However, generating such nogoods is not obvious. Indeed, there is not notion of levels, the classical notion of conflict analysis and first UIP can not be extended. Furthermore, conflict graphs can contain cycles. In this case, the resolution step can produce tautological clauses which are obviously useless. To overcome these problems, we propose to transform conflict graph of a complete interpretation into a classical implication graph and then use classical learning to generate relevant nogoods.

3.2 Building Conflict Graph

To overcome problems introduced in previous section, we propose a first method, based on unit propagation. Starting from a complete interpretation $\mathcal{I}_c$, we build a partial interpretation $\mathcal{I}'$ by unit propagation where decision variables have the same value as in $\mathcal{I}_c$. This partial interpretation introduced formally in the following definition will help us to build the conflict graph.

Definition 5 (Derived partial interpretation) *Let* Σ *be a CNF formula,* $\mathcal{I}_c$ *a complete interpretation. The derived partial interpretation of* $\mathcal{I}_c$*, denoted* $\mathcal{I}'$*, is incrementally build as follows:*

- $\mathcal{I}'_0 = \emptyset$;
- $\mathcal{I}'_{i+1} = \mathcal{I}'_i \cup \{(x_{i+1}), x^1_{i+1}, ..., x^k_{i+1}\}$ *such that* $x_{i+1} \in \mathcal{V}_\Sigma \setminus \mathcal{I}'_i$ *and* $\forall j, 1 \leqslant j \leqslant k$ *one has* $\Sigma_{|\mathcal{I}'_i \cup \{x_{i+1}\}} \models_* x^j_{i+1}$ *with* $x^j_{i+1} \in \mathcal{I}_c$;
- $\mathcal{I}' = \mathcal{I}'_i \cup \{(x_{i+1}), x^1_{i+1}, ..., x^l_{i+1}\}$ *such that* $x_{i+1} \in \mathcal{V}_\Sigma \backslash \mathcal{I}'_i$ *and* $\forall j, 1 \leqslant j \leqslant l$ *one has* $\Sigma_{|\mathcal{I}'_i \cup \{x_{i+1}\}} \models_* x^j_{i+1}$ *with* $x^j_{i+1} \in \mathcal{I}_c$ *for* $j \neq l$ *and* $x^l_{i+1} \notin \mathcal{I}_c$.

The set of variables associated to the decision literals is called a conflict set. Furthermore, we call conflict variable the one associated to the literal $x \in \mathcal{I}' \setminus \mathcal{I}$. The literal x is also called a conflict literal.

Of course, the choice of the conflict set of variables is a heuristic choice and can lead to different derived partial interpretations.

Example 5 *Let us consider again the CNF Σ of example 1 and the complete interpretation $\mathcal{I}_c = \{x_1, \dots x_9\}$.*
First, considering the decision variables in lexicographic ordering. We have:

- $\mathcal{I}'_0 = \emptyset$
- $\mathcal{I}'_1 = \{(x_1), x_2^1, \overline{x_3^1}\}$

Then, the conflict variable is x_3 and the conflict set is limited to $\{x_1\}$.
Now, considering the inverse lexicographic ordering, the partial interpretation is:

- $\mathcal{I}'_0 = \emptyset$
- $\mathcal{I}'_1 = \{(x_9^1), x_7^1, x_5^1, x_1^3\}$
- $\mathcal{I}'_2 = \{(x_9^1), x_7^1, x_5^1, x_1^3, (x_8^2), x_6^2, x_4^2, x_1^2, x_2^2, \overline{x_2^2}\}$

the conflict variable is x_2 and the conflict set is $\{x_9, x_8\}$.

Complete interpretation is conflicting, so the derived partial interpretation will differ on, at least, one conflict literal. The following proposition asserts that it exists at least one conflict clause containing this literal. The conflict graph is then built on this literal.

Proposition 2 *Let Σ be a CNF formula, $\mathcal{I}_c$ a conflicting complete interpretation and $\mathcal{I}'$ a derived partial interpretation of $\mathcal{I}_c$. Let x be the conflict literal, then $exp(x) \subseteq \mathcal{I}_c$ and the clause $\overrightarrow{cla}(x)$ is falsified by $\mathcal{I}_c$.*

Proof First, by construction of $\mathcal{I}'$, it is obvious that $exp(x) \subseteq \mathcal{I}_c$. Indeed, suppose that $exp(x) \not\subseteq \mathcal{I}_c$ then $\exists y \in exp(x)$ such that $y \notin \mathcal{I}_c$. By definition $exp(x) \subseteq \mathcal{I}'$, so $y \in \mathcal{I}'$, consequently the literal y is also a conflict literal. By construction of $\mathcal{I}'$, it exists only one conflict literal, then $y = x$. This is impossible, because a propagated literal can not be included in its explanation.

Secondly, we have to prove that $\mathcal{I}_c \not\models \overrightarrow{cla}(x)$. Suppose $\overrightarrow{cla}(x)$ is satisfied by $\mathcal{I}_c$. Note that x is a propagated literal, then it exists an explanation $exp(x)$ and a clause $\overrightarrow{cla}(x) \in \Sigma$ such that $\overrightarrow{cla}(x) = \overline{exp(x)} \vee x$. We know that $exp(x) \subseteq \mathcal{I}_c$, then $\overline{exp(x)} \not\subseteq \mathcal{I}_c$. Since $\overrightarrow{cla}(x)$ is satisfied by $\mathcal{I}_c$, it can be only satisfied by x. This is not possible because $x \notin \mathcal{I}_c$.

The following proposition expresses the fact that all clauses (expect the falsified one) which are used to construct the graph are once-satisfied clauses.

Proposition 3 *Let Σ be a CNF formula, $\mathcal{I}_c$ a complete conflicting interpretation, $\mathcal{I}'$ a derived partial interpretation and x the conflict literal associated to $\mathcal{I}'$. Consider the implication graph $\mathcal{G}_\Sigma^{\mathcal{I}'} = (\mathcal{N}, \mathcal{A})$, then $\forall y \in \mathcal{N} \setminus \{x\}$ one has $\overrightarrow{cla}(y) = \perp$ where $\overrightarrow{cla}(y)$ is once-satisfied by $\mathcal{I}'$ on x.*

Proof One need to consider two cases:

1. y is a decision literal, then $\overrightarrow{cla}(y) = \perp$;
2. y is a propagated literal. It exists $\overrightarrow{cla}(y) \in \Sigma$ such that $\overrightarrow{cla}(y) = \overline{exp(y)} \vee y$. By construction of $\mathcal{I}'$, one has $x \notin exp(y)$ and $\mathcal{I}' \setminus \{x\} \subset \mathcal{I}_c$. Since $y \neq x$, one has $\{exp(y), y\} \subseteq \mathcal{I}'$. By transitivity, one obtain $\{exp(y), y\} \subseteq \mathcal{I}_c \setminus \{x\}$. Then, the clause $\overrightarrow{cla}(y)$ is once-satisfied by $\mathcal{I}_c$ on y.

In the proposition 4, we show that the implication graph obtained with the partial derived interpretation can be extended in a conflict graph on the conflict literal. Then, with the help of this implication graph, it is possible to generate nogoods similarly to classical CDCL solvers [7]. Of course, these nogoods will be added to the clauses database.

Proposition 4 *Let Σ be a CNF formula, $\mathcal{I}_c$ a complete interpretation on Σ, $\mathcal{I}'$ a partial derived interpretation and x the conflict literal associated to $\mathcal{I}'$. If $\exists \alpha \in \Sigma$ once-satisfied by $\mathcal{I}_c$ on x, then it is possible to extend the implication graph $\mathcal{G}_\Sigma^{\mathcal{I}'} = (\mathcal{N}, \mathcal{A})$ associated to $\mathcal{I}'$ to a conflict graph $\mathcal{G}_{\Sigma,\mathcal{I}_c}^x = (\mathcal{N}', \mathcal{A}')$ as follows:*

- $\mathcal{N}' = \mathcal{N} \cup \{y \in \overline{\alpha} \setminus x\} \cup \{\overline{x}, \perp\}$ *;*
- $\mathcal{A}' = \mathcal{A} \cup \{(y, \overline{x}) | y \in \overline{\alpha} \setminus x\} \cup \{(x, \perp), (\overline{x}, \perp)\}$.

Proof Before proving that $\mathcal{G}_{\Sigma,\mathcal{I}_c}^x$ is a valid conflict graph, one has to identify clauses $\beta = \{\beta_1, \dots, \beta_k, z\} \in \Sigma$ falsified by $\mathcal{I}_c$ and $\gamma = \{\gamma_1, \dots, \gamma_l, \overline{z}\} \in \Sigma$ once-satisfied on z for $\mathcal{I}_c$. By hypothesis, clause α is once-satisfied by $\mathcal{I}_c$ on x. Obviously, $\gamma = \alpha$. By proposition 2, one can take $\beta = \overrightarrow{cla}(x)$. Indeed, $x \in \overrightarrow{cla}(x)$ and the clause $\overrightarrow{cla}(x)$ is falsified by $\mathcal{I}_c$. The both clauses used for the construction of the conflict graph are now identified. To prove that $\mathcal{G}_{\Sigma,\mathcal{I}_c}^x = (\mathcal{N}', \mathcal{A}')$ is a conflict graph, one has to verify the following properties (see definition 4):

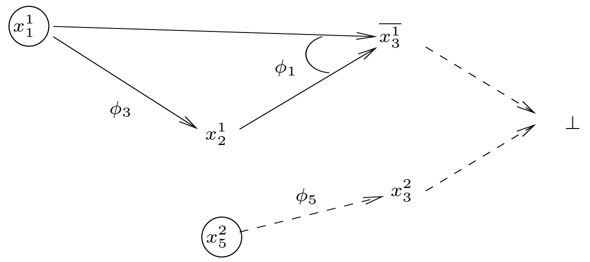
1. - $\{x, \overline{x}, \perp\} \subseteq \mathcal{N}'$. By hypothesis, one has $\{\overline{x}, \perp\} \subseteq \mathcal{N}'$, We only need to prove that $x \in \mathcal{N}'$. We know that $x \in \mathcal{I}'$, by construction of $\mathcal{G}_\Sigma^{\mathcal{I}'}$, then we have $x \subseteq \mathcal{N}$. Since $\mathcal{N} \subseteq \mathcal{N}'$, then $x \in \mathcal{N}'$;

- $\{\overline{\gamma_1}, \dots, \overline{\gamma_l}\} \subseteq \mathcal{N}'$. We have $\{y \in \overline{\alpha} \setminus x\} \subseteq \mathcal{N}'$ and $\gamma = \alpha$. Then $\{y \in \overline{\gamma} \setminus x\} \subseteq \mathcal{N}'$;
- $\{\overline{\beta_1}, \dots, \overline{\beta_k}\} \subseteq \mathcal{N}'$. By hypothesis, $\beta = \overrightarrow{cla}(x) = \beta_1 \vee \dots \vee \beta_k \vee x = \overline{exp}(x) \vee x$. Then, $exp(x) = \{\overline{\beta_1}, \dots, \overline{\beta_k}\}$. By proposition 2, one has $exp(x) \subseteq \mathcal{I}'$, so $exp(x) \subseteq \mathcal{N}$ (see definition 1). Since $\mathcal{N} \subseteq \mathcal{N}'$, by transitivity, one has $exp(x) \subseteq \mathcal{N}'$;

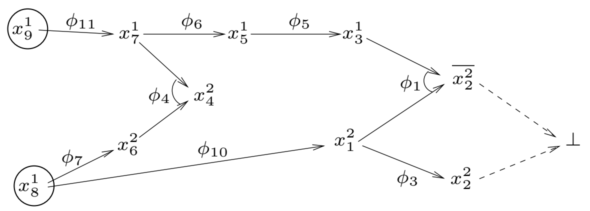
2.
- $\{(x, \perp), (\overline{x}, \perp)\} \subseteq \mathcal{A}'$. By construction ;
- $\{(\overline{\gamma_1}, \overline{x}), \dots, (\overline{\gamma_k}, \overline{x})\} \subseteq \mathcal{A}'$. By construction ;
- $\{(\overline{\beta_1}, x), \dots, (\overline{\beta_k}, x)\} \subseteq \mathcal{A}'$. One knows that $exp(x) = \{\overline{\beta_1}, \dots, \overline{\beta_k}\}$ and $\{exp(x), x\} \subseteq \mathcal{I}'$. By construction of $\mathcal{A}'$ and by definition 1, one has $\{(\overline{\beta_1}, x), \dots, (\overline{\beta_k}, x)\} \subseteq \mathcal{A} \subseteq \mathcal{A}'$;

3. $\forall x \in \mathcal{N}$, if $x \neq z$ and $\alpha = \bigwedge\{y \in \mathcal{N} \mid (y, x) \in \mathcal{A}\} \not\models \perp$ then $\overline{\alpha} \vee x \in \Sigma$ and is once-satisfied on x. By proposition 3, $\forall y \in \mathcal{N}$ such that $y \neq x$ and $\overrightarrow{cla}(y) \neq \perp$, one has $\overrightarrow{cla}(y)$ once-satisfied by $\mathcal{I}_c$ on y. By construction of $\mathcal{G}_{\Sigma}^{\mathcal{I}'}$ and $\mathcal{G}_{\Sigma,\mathcal{I}_c}^{x}$ the previous property is verified.

Example 6 *Let us take again example 1 and partial derived interpretations obtained in example 5. We can extend implication graphs associated to these interpretations in two conflict graphs depicted in Figures 3.a (lexicographic order) and 3.b (inverse lexicographic order).*



a. lexicographic order



b. inverse lexicographic order

Figure 3. Conflict graph constructed with unit propagation (example 6)

3.3 Implementation

We propose to incorporate the conflict graph defined in previous section inside a WSAT like solver [23]. This is done when a local minima is reached. We name this method CDLS. During the construction of the partial derived interpretation, two cases might occur: Either a conflict is reached during unit propagation (see Figure 3.b) or not (see Figure 3.a). In the former, one uses the resulting implication graph, analyzes the conflict and extracts an assertive clause associated to the first UIP. In the latter, one extends the implication graph of the derived partial interpretation to a conflict graph and, in a similar way, an assertive clause is extracted. In both cases, assertive clauses are added to the clauses database of the formula. And, if the assertive clause is the empty one, the unsatisfiability of the formula is proved. Contrary to classical WSAT algorithm, one can flip a set of variables when a local minima is reached. These are variables whose values differ between complete interpretation and derived partial interpretation.

It is important to note that CDLS is not an hybrid algorithm like [9, 10]. It is a simple stochastic local search method. Unit propagation is only used to build the conflict graph, to analyze conflict graph and to extract nogoods.

Algorithm 1: CDLS

Input: Σ a CNF formula
Output: SAT if Σ is satisfiable, $UNSAT$ if Σ is unsatisfiable, else $UNKNOWN$

for $i \leftarrow 1$ *to* $MaxTries$ **do**
  $\mathcal{I}_c \leftarrow$ completePUInterpretation(Σ);
  for $j \leftarrow 1$ *to* $MaxFlips$ **do**
    if $\mathcal{I}_c \models \Sigma$ **then**
      return SAT;
    $\Gamma = \{\alpha \in \Sigma | \mathcal{I}_c \not\models \alpha\}$;
    while $\Gamma \neq \emptyset$ **do**
      $\alpha \in \Gamma$;
      if $\exists x \in \alpha$ *allowing a descent* **then**
        $flip(x)$;
        break;
      else
        $\Gamma \leftarrow \Gamma \setminus \{\alpha\}$;
    if $\Gamma = \emptyset$ **then** `/* local minimum */`
      $\alpha \in \Sigma$ such that $\mathcal{I}_c \not\models \alpha$;
      $\beta \leftarrow conflictAnalysisRL(\Sigma, \mathcal{I}_c, \alpha)$;
      if $\beta = \perp$ **then**
        return UNSAT;
      $\Sigma \leftarrow \Sigma \cup \{\beta\}$;
return $UNKNOWN$;

Algorithm 2: completePUInterpretation

Input: Σ a CNF formula
Output: $\mathcal{I}_c$ a complete interpretation of Σ

1 $\Sigma' \leftarrow \Sigma$; $\mathcal{I}_c \leftarrow \emptyset$;
2 **while** $\Sigma' \neq \emptyset$ **do**
3 $\quad x \in lit(\Sigma')$;
4 $\quad \mathcal{P} \leftarrow \{x\} \cup \{y | \Sigma_{|x} \models_* y\}$;
5 $\quad$ **foreach** $y \in \mathcal{P}$ **do**
6 $\quad\quad$ **if** $\overline{y} \in \mathcal{P}$ **then**
7 $\quad\quad\quad \mathcal{P} \leftarrow \mathcal{P} \setminus \{y\}$;
8 $\quad \Sigma' \leftarrow \Sigma'_{|\mathcal{P}}$;
9 $\quad \mathcal{I}_c \leftarrow \mathcal{I}_c \cup \mathcal{P}$;
10 **return** $\mathcal{I}_c$;

Algorithm 3: conflictSet

Input: Σ a CNF; $\mathcal{I}_c$ a complete interpretation; $\alpha \in \Sigma$ a critical clause for $\mathcal{I}_c$.
Output: $\mathcal{C}$ a conflict literals set

1 $\mathcal{C} \leftarrow \emptyset$;
2 **forall** $x \in \overline{\alpha}$ **do**
3 $\quad \beta \in \Sigma$ once-satisfied on x;
4 $\quad \mathcal{C} \leftarrow \mathcal{C} \cup \{\beta \setminus \{x\}\}$;
5 **return** $\mathcal{C}$;

Algorithm CDLS (see Algorithm 1) takes a CNF formula Σ as input and returns three different values (SAT, $UNSAT$ or $UNKNOWN$). It is based on WSAT algorithm. Note that initial complete interpretations are generated using unit propagation (line 2 of Algorithm 1 and Algorithm 2). In this way, local minimas are quickly reached and variables dependencies are taken into account i.e using unit propagation. Whenever a descent is possible, one flips a variable allowing it. When a local minima is reached, we analyze the conflict as explained previously. A nogood β is generated and added to the clause database.

Algorithm 4 is the core or our proposed framework. It starts by selecting conflict variables set which allows to build the derived partial interpretation. This is done by Algorithm 3. It chooses variables in linked clauses to the falsified clause α in order to make the generated partial interpretation nearest to α. Then, this partial interpretation $\mathcal{I}_p$ is constructed (line 4-7). Two cases might occur. In the former, a conflict is reached, classical conflict analysis is applied (line 9) and an assertive literal is flipped. In the latter, one can extracts the partial derived interpretation of $\mathcal{I}_p$ and generates the associated conflict graph (line 12-14 and proposition 4). At this point, conflict analysis can be achieved. All variables with different values in both interpretations are then flipped.

Algorithm 4: conflictAnalysisRL

Input: Σ a CNF; $\mathcal{I}_c$ a complete interpretation; $\alpha \in \Sigma$ a critical clause for $\mathcal{I}_c$.
Output: β a clause built on $\mathcal{V}_\Sigma$

1 $\mathcal{E} \leftarrow conflictSet(\Sigma, \mathcal{I}_c, \alpha)$; $\gamma \leftarrow \emptyset$; $\mathcal{I}_p \leftarrow \emptyset$;
2 **while** $(\gamma = \emptyset)$ *and* $(\mathcal{I}_p \subset \mathcal{I}_c)$ **do**
3 $\quad \mathcal{E} \leftarrow \mathcal{E} \setminus \mathcal{I}_p$; $\mathcal{I}_p \leftarrow \mathcal{I}_p \cup \{x\}$ such that $x \in \mathcal{E}$; $\gamma \leftarrow BCP()$;
4 **if** $\gamma \neq \emptyset$ **then** `/* CASE 1 */`
5 $\quad \beta \leftarrow firstUIP(\mathcal{G}_\Sigma^{\mathcal{I}_p})$;
6 $\quad flip(x)$ with x assertive literal ;
7 **else** `/* CASE 2 */`
8 $\quad \mathcal{I}' \leftarrow$ partial interpretation associated to $\mathcal{I}_p$;
9 $\quad y \leftarrow$ conflict literal of $\mathcal{I}'$;
10 $\quad \mathcal{G}^y_{(\Sigma,\mathcal{I}')} \leftarrow$ extended conflict graph $\mathcal{G}_\Sigma^{\mathcal{I}'}$;
11 $\quad \beta \leftarrow firstUIP(\mathcal{G}^y_{(\Sigma,\mathcal{I}')})$;
12 $\quad$ **forall** $x \in \mathcal{I}_p \setminus \mathcal{I}_c$ **do** $flip(x)$;
13 **return** β;

	Crafted		Industrial		Random	
	sat	unsat	sat	unsat	sat	unsat
ADAPTG2	326	0	232	0	1111	0
RSAPS	339	0	226	0	1071	0
WSAT	259	0	206	0	1012	0
CDLS	331	146	412	232	943	0
CLS	235	75	227	102	690	0
MINISAT	402	369	588	414	609	315

Table 1. CDLS **versus some other SAT solvers**

4 Experimental results

Experimental results reported in this section were obtained on a Xeon 3.2 GHz with 2 GByte of RAM. CPU time is limited to 1200 seconds. We compare CDLS to three classical incomplete local search methods, WSAT [23], RSAPS [15] and ADAPTG2 [16] ; we also add CLS the complete local search solver proposed by [8] and MINISAT [7] one of state-of-the-art CDCL solver. Instances used are taken from the last SAT competitions. They are divided into different categories: crafted (1439 instances), industrial (1305) and random (2172). All instances are preprocessed wih SatElite [6]. Indeed, it is well known resolution based preprocessors help, in a lot of cases, local search and complete ones [1].

Table 4 summarizes the obtained results on this large number of instances. For more details on this experimental part, the reader can refer to `http://www.cril.fr/~lagniez/cdls`. For each category and for each solver we report the number of solved instances. Of course, MIN-

ISAT a state-of-the-art CDCL based complete solver is considered here, only to mention the gap between local search based techniques and complete modern SAT solvers on industrial and crafted instances. On random satisfiable instances, local search techniques generally outperform complete techniques.

Let us start to analyze the results obtained on the crafted category. Except for MINISAT, CDLS is very competitive and solves approximately the same number of instances than RSAPS and ADAPTG2. Furthermore, CDLS solves much more instances than WSAT, its built-in solver. Comparing to CLS, our solver is better, it solves more SAT and UNSAT instances. For industrial instances, CDLS solves two times more instances than other stochastic local search solvers and 232 unsatisfiable instances. It outperforms CLS, the other complete local search solver. So, conflict analysis allows to solve efficiently structured SAT and UNSAT instances. Finally, for the random category, we can note that CDLS and CLS are unable to solve unsatisfiable problems. As pointed by MINISAT results, learning is not the good approach to solve random instances.

A summary, our solver CDLS is much more efficient than other local search algorithms. It significantly improves WSAT, its built-in solver and CLS another complete local search approach. Even if MINISAT is the best solver on crafted and industrial instances, these first results are very encouraging and reduce the gap between local search based techniques and DPPL-like complete solvers.

5 Conclusion

In this paper, we propose a new approach to keep out local minimum when dealing with a stochastic local search solver. Our approach extends conflict analysis used by modern SAT solvers. With this extension, we significantly improved stochastic local search solvers. More interestingly, our approach is able to prove inconsistency of many SAT instances, and then it can be seen as an important step to the resolution of the challenge number 5 proposed by Selman et al. at IJCAI 1997. These first results are very promising. Our solver is able to solve a lot of unsatisfiable instances and it achieves interesting improvements of local search based techniques on structured SAT instances. In future works, we plan to improve heuristic choice for the conflict set. We also want to analyze and extract nogoods without the help of unit propagation. Finally, we plan to study other schemes for extracting relevant nogoods.

References

[1] Anbulagan, D.N. Pham, J. Slaney, and A. Sattar. Boosting sls performance by incorporating resolution-based preprocessor. In *proceedings of the workshop LSCS (in conjonction to CP)*, 2006.

[2] G. Audemard, L. Bordeaux, Y. Hamadi, S. Jabbour, and L. Saïs. A generalized framework for conflict analysis. In *proceedings of SAT*, pages 21–27, 2008.

[3] G. Audemard and L.Simon. Gunsat: A greedy local search algorithm for unsatisfiability. In *Proceedings of IJCAI*, pages 2256–2261, 2007.

[4] B. Cha and K. Iwama. Adding new clauses for faster local search. In *proceedings of AAAI*, pages 332–337, 1996.

[5] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communication of ACM*, 5(7):394–397, 1962.

[6] N. Eén and A. Biere. Effective preprocessing in SAT through variable and clause elimination. In *proceedings of SAT*, pages 61–75, 2005.

[7] N. Een and N. Sörensson. An extensible SAT-solver. In *proceedings of SAT*, pages 502–518, 2003.

[8] H. Fang and W. Ruml. Complete local search for propositional satisfiability. In *proceedings of AAAI*, pages 161–166, 2004.

[9] L. Fang and M. Hsiao. A new hybrid solution to boost SAT solver performance. In *proceedings of DATE*, pages 1307–1313, 2007.

[10] E. Goldberg. A decision-making procedure for resolution-based SAT-solvers. In *proceedings of SAT*, pages 119–132, 2008.

[11] É. Grégoire, B. Mazure, and C. Piette. Extracting MUSes. In *proceedings of ECAI*, pages 387–391, 2006.

[12] É. Grégoire, R. Ostrowski, B. Mazure, and L. Saïs. Automatic extraction of functional dependencies. In *proceedings of SAT*, pages 122–132, 2004.

[13] E.A. 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.

[14] H.H. Hoos and T. Stützle. *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann / Elsevier, 2004.

[15] F. Hutter, D. Tompkins, and H. Hoos. Scaling and probabilistic smoothing: Efficient dynamic local search for SAT. In *proceedings of CP*, pages 233–248, 2002.

[16] Chu Min Li, Wanxia Wei, and Harry Zhang. Combining adaptive noise and look-ahead in local search for sat. In *proceedings of SAT*, pages 121–133, 2007.

[17] J. Marques-Silva and K. Sakallah. GRASP - A New Search Algorithm for Satisfiability. In *proceedings of ICCAD*, pages 220–227, 1996.

[18] P. Morris. The breakout method for escaping from local minima. In *proceedings of AAAI*, pages 40–45, 1993.

[19] 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.

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

[21] S. Prestwich and I. Lynce. Local search for unsatisfiability. In *proceedings of SAT*, pages 283–296, 2006.

[22] B. Selman and H. Kautz. An empirical study of greedy local search for satisfiability testing. In *proceedings of AAAI*, pages 46–51, 1993.

[23] B. Selman, H. Kautz, and B. Cohen. Noise strategies for improving local search. In *proceedings of AAAI*, pages 337–343, 1994.

[24] B. Selman, H. Kautz, and D. McAllester. Ten challenges in propositional reasoning and search. In *proceedings of IJCAI*, pages 50–54, 1997.

[25] H. Shen and H. Zhang. Another complete local search method for SAT. In *proceedings of LPAR*, pages 595–605, 2005.

[26] L. Zhang, C.F. Madigan, M.W. Moskewicz, and S. Malik. Efficient conflict driven learning in boolean satisfiability solver. In *proceedings of ICCAD*, pages 279–285, 2001.