

The long way from CDC_lL * to CDC_oL †

Daniel Le Berre and Anne Parrain and Olivier Roussel
CRIL-CNRS FRE 2499, Université d'Artois, Lens, FRANCE
{leberre,parrain,roussel}@cril.univ-artois.fr

1 Motivation

Current SAT solvers are powerful enough to be used as engines in real applications. Those applications made the success of a special kind of SAT solvers, namely Conflict Driven Clause Learning SAT solvers (CDC_lL for short), developed initially by Joao Marques Silva with GRASP [8], and popularized by the SAT solver Chaff [9]¹. Despite SAT being a NP-complete problem in theory, it might look tractable in practice when looking at the huge size (several millions of variables and clauses) of some bounded model checking SAT encodings solved in a few minutes by current state of the art solvers. As a consequence, researchers are pushing the limit beyond SAT: Quantified Boolean Formulas (QBF) and Stochastic SATisfiability (SSAT) for instance are two extensions of SAT being studied recently. Another extension of SAT received some attention a decade ago: using pseudo boolean constraints instead of plain clauses [1, 2]. Most of the solvers for those extensions to SAT are developed using techniques that were demonstrated powerful for SAT. Those solvers in the early 90s were based on DPLL[5, 4] while the solvers developed today are often related to CDC_lL solvers.

This is especially true for pseudo boolean solvers: Barth first developed a DPLL version of a pseudo boolean solver [2]. Walser [16] and later Prestwich [11, 12] developed local search or hybrid pseudo boolean solvers. Aloul et al [13] developed a version of Chaff handling pseudo boolean constraints instead of clauses as input, plus symmetry breaking predicates, with clause learning (same thing for the recent MiniSAT [14]). Dixon and Ginsberg[6] developed a pseudo boolean version of Relsat (PRS), which was the first pseudo boolean solver including true pseudo boolean learning. They developed a pseudo boolean version of Chaff (PBChaff[7]) in the same spirit while Chai and Kuehlmann [3] did extended all Chaff techniques (learning scheme and data structures) in the pseudo boolean solver Galena. Very recent work, and partially unpublished,

*Conflict Driven Clause Learning

†Conflict Driven Constraint Learning

¹One can check that all the strong SAT solvers in the industrial category during the two last SAT competitions were CDC_lL solvers.

from Dixon et al [10] describe a generic CDC_oL solver based on group theory while Thiffault et al [15] describe a CDC_iL solver working with arbitrary boolean gates.

The assumption behind those works is that a good solver for SAT can be extended to a solver for an extension of SAT. We do not claim to answer here whether this assumption is valid or not but we would like to emphasize that CDC_iL solvers benefit from many nice properties reducing the computation effort that are no longer valid using more general constraints.

We first review the basic mechanisms characterizing CDC_iL solvers and the underlying theory in propositional logic. Then we provide some definitions and properties concerning pseudo boolean constraints. After defining a generalized Conflict Driven Constraint Learning (CDC_oL) solver, built on the notion of *assertive constraint*, we discuss the issues for implementing such a solver.

2 CDC_iL solvers

DPLL solver are searching the boolean space using a binary tree on the variables truth value. Traditional DPLL solvers are exploring explicitly that search tree. By contrast, CDC_iL solvers are using a specific learning scheme to explore the boolean search space. The idea is the following:

1. Any truth value propagation is either the consequence of one of the constraints, in that case the constraint is called the reason of the propagation and the truth value is said *forced*, or the consequence of the heuristics, in that case the truth value is called a *decision value*. This is true for any search based SAT solver. The basic case in which a clause does propagate a truth value is when that clause contains only one literal (a so called *unit clause*).
2. propagation of the truth values may end with no more variables to assign: in that case the formula is proved SATisfiable
3. most often, the propagation will end with a conflict: a constraint will be falsified. In that case, a classical backtracking algorithm would undo the latest decision value and try the opposite one. In CDC_oL solvers, the conflict analysis engine will analyse the reason of the conflict and produce both a *clause* and a *backtrack level* such that the clause becomes unit at the proposed backtrack level after simplification. Note that a CDC_oL solver may backtrack higher than the latest decision level (back-jumping) and may not flip a decision value when backtracking: the flip implied by the new clause may occurs for any truth value that was fixed below the backtracking level. As a consequence, each time a conflict is found, a decision value becomes forced. The inconsistency is proven when a conflict results from forced only truth values.

3 Linear pseudo boolean constraints

3.1 Definitions

A linear pseudo boolean constraint is defined over a set of boolean variables x_i . The value true will be coded as 1 while the value false will be coded by 0. A general linear pseudo boolean constraint follows the following pattern $\sum_i a_i \cdot x_i \triangleright k$ where a_i and k are constants (integer or real) and \triangleright is one of the classical relational operators ($=, >, \geq, <, \leq$). The multiplication and the sum operators have their usual mathematical meaning.

Such general linear pseudo boolean constraints can be normalized. First, in practice, we never need to solve a constraint with transcendental coefficients. Therefore, we can assume that each coefficient is a fraction and by computing a common denominator for each fraction the constraint can be normalized to use only integer coefficients. Furthermore, these constraints can be normalized to use only the \geq operator (equality constraints can be split in two constraints (\geq and \leq); $<$ and \leq constraints can be multiplied by -1 to obtain $>$ and \geq constraints respectively and at last $\sum_i a_i \cdot x_i > k$ can be normalized into $\sum_i a_i \cdot x_i \geq k + 1$. By introducing literals l_i and \bar{l}_i , we may further normalize so that each coefficient is only a positive integer. Let $l_i = x_i$ and $\bar{l}_i = 1 - x_i$. Every conjunction of linear pseudo boolean constraints can be normalized to become a conjunction of greater or equal constraints over literals with strictly positive coefficients. For example, $3x_1 - 7x_2 < -2$ is normalized into $3\bar{l}_1 + 7l_2 \geq 10$

The right hand side of the constraint (k) is called the degree of the constraint.

Clauses and cardinality constraints can be seen as a special case of linear pseudo boolean constraints. Clause $l_1 \vee l_2 \vee \dots \vee l_n$ translates to $l_1 + l_2 + \dots + l_n \geq 1$, cardinality constraint *atleast*($k, \{l_1, l_2, \dots, l_n\}$) translates to $l_1 + l_2 + \dots + l_n \geq k$ and *atmost*($k, \{l_1, l_2, \dots, l_n\}$) translates to $\bar{l}_1 + \bar{l}_2 + \dots + \bar{l}_n \geq n - k$.

3.2 Inference rules

In propositional calculus, there are two essential inference rules which can be applied to a formula in conjunctive normal form: *resolution* and *merging*. The latter one is often underestimated because usually clauses are assimilated to sets of literals and the merge rule is automatically obtained by the set union. However, this merge rule is necessary for the completeness of a refutation procedure. The merge rule also plays a central role in the predicate calculus where literals may be merged by computing a most general unifier. Here again, both resolution and merging are necessary to obtain a complete refutation procedure, but the merge rule is no more straightforward as in propositional calculus.

With LPB constraint, we need essentially the same two rules to get a complete procedure but we may also use additional inference rules which have no equivalent in the propositional calculus. The rule corresponding to resolution is called *cutting planes* and computes a positive linear combination of two LPB constraints.

$$\text{cutting planes: } \frac{\begin{array}{l} \sum_i a_i \cdot x_i \geq k \\ \sum_i a'_i \cdot x_i \geq k' \end{array}}{\sum_i (\alpha \cdot a_i + \alpha' \cdot a'_i) \cdot x_i \geq \alpha \cdot k + \alpha' \cdot k'}$$

with $\alpha > 0$ and $\alpha' > 0$

In general, the coefficients of the linear combination are chosen so as to eliminate at least one variable, i.e. such that $\exists i, \alpha \cdot a_i + \alpha' \cdot a'_i = 0$. We may notice that, in contrast to resolution in propositional calculus, we may form a combination which doesn't eliminate any variable. Furthermore, one single linear combination may eliminate more than one variable (which is impossible in propositional calculus).

The second essential rule corresponding to the merge rule is called *saturation*. When one coefficient a_j is greater than the degree k , it may be replaced by k (which amounts to merging some occurrences of x_j).

$$\text{saturation: } \frac{a_j \cdot x_j + \sum_{i \neq j} a_i \cdot x_i \geq k \text{ with } a_j > k}{k \cdot x_j + \sum_{i \neq j} a_i \cdot x_i \geq k}$$

One simple way to justify this rule is to consider that, when x_j is true, the constraint will be satisfied even if a_j is reduced to k , and therefore, reducing the coefficient to k doesn't change anything. More rigorously, this rule derives from an iterated application of the rounding rule.

This rounding rule allows us to divide each coefficient as well as the degree by a strictly positive number and round up each number obtained.

$$\text{rounding up: } \frac{\sum_i a_i \cdot x_i \geq k}{\sum_i \lceil a_i / \alpha \rceil \cdot x_i \geq \lceil k / \alpha \rceil \text{ with } \alpha > 0}$$

One last inference rule that will prove useful is reduction. It consists in forgetting one of the variables of the constraint and adjusting the degree in concordance.

$$\text{reduction: } \frac{\sum_i a_i \cdot x_i \geq k}{\sum_{i \neq j} a_i \cdot x_i \geq k - a_j}$$

4 A Conflict Driven Constraint Learning Solver

D. Chai and A. Kuehlmann present in [3] a generalization of the structure of the CDC_lL solvers [8] which takes as input clauses, cardinality or pseudo-boolean constraints and learn any kind of clauses, cardinality or pseudo-boolean constraints. The purpose of this section is to describe the differences between Chai *et al*'s CDC_oL solver and CDC_lL solvers concerning boolean constraint propagation and the conflict analysis algorithms.

4.1 Implicative and Assertive constraints

The definition of unit clauses can be extended to the pseudo-boolean constraints case. However, the term *unit* refers to the syntactic form of such clauses. This term is not meaningful for pseudo-boolean constraints, so we will prefer to name such constraints *implicative constraints*. A unit clause implies exactly one literal, which is no longer true for pseudo-boolean constraints. An *implicative constraint* is a constraint which implies *at least* one literal. More formally, a constraint C is *implicative* iff $\exists a_i x_i \in C$ such that $a_i > \sum a_j - d$. For example, for x_1, x_2, x_3 unassigned literals, the pseudo-boolean constraint $3x_1 + 2x_2 + x_3 \geq 5$ implies x_1 and x_2 are satisfied (assigned to 1).

In CDC₇L solvers, the aim of the conflict analysis algorithm is to learn a clause which will be unit when backtracking. These clauses are called *assertive clauses*. This definition can be extended easily to the case of pseudo-boolean constraints: an *assertive constraint* is a constraint that will become implicative when backtracking.

4.2 Detecting implicative constraints

Dixon *et al* [7] name *poss* the difference between the sum of the coefficients of the literals which are not falsified and the degree of the constraint. The same value is named *slack* by Chai *et al*. It permits to detect easily some states of the constraint:

- a constraint is falsified iff its *poss* value is negative;
- a constraint is implicative iff there exists a literal x_i with coefficient c_i such that $poss - c_i$ is negative.

As a result, maintaining the value of *poss* during the search is sufficient to detect when a constraint become falsified or implicative.

The watched literals scheme proposed by [9] can also be used to detect those states, as proposed in [3]. While it is sufficient to watch two unfalsified literals for a clause, the set of watched literals must contain satisfied or unassigned literals such that the sum of their coefficients is greater than the sum of the degree of the constraint and the largest coefficient of the unassigned literals. When a literal of this set is falsified, it is replaced by the minimal number of literals which allows to verify the condition. As a result, the size of the set may vary during the resolution, which is not the case for a clause. From that property, one can also deduce how many literals must be watched for cardinality constraints: exactly its degree + 1, because the largest coefficient of unassigned literals is 1. It means that the larger the degree, the larger the set of literals to watch. Clauses are the best cases for that approach.

4.3 Detecting assertive constraints

CDC₀L solvers detect assertive clauses using a syntactical test based on the definition of a Unique Implication Point: assertive clauses contain only one

variable of the current decision level. This can be done either by making some cut in the implication graph, or by applying iteratively resolution on the conflict clause and the reasons of the previous assignments, until an assertive clause is found.

For this, the search benefits from a nice property of unit clauses: a clause is unit at most once in each branch in the resolution tree. This property does not stand anymore in the pseudo-boolean case. For example, for x_1, x_2, x_3 unassigned literals, the pseudo-boolean constraint $3x_1 + 2x_2 + x_3 \geq 4$ implies x_1 is satisfied. As x_2 and x_3 are unassigned, it is also easy to see that, as soon as x_2 is falsified, the constraint will imply x_3 to be satisfied, and vice-versa. Then, the same constraint can be the conflict constraint and a reason of a previous assignation.

Furthermore, the resolution of the falsified constraint and the reason of the conflictual assignment does not always result in a falsified constraint, as opposed to the clausal case. This is because an assignment forced by an implicative constraint can over-satisfy this constraint. The *poss* value represents the constraint's margin to the unsatisfiability. To obtain a conflictual constraint by applying resolution, constraints must have *poss* values such that their sum is strictly negative. Reduction of constraints by deleting unassigned or satisfied literals can be needed to reduce the *poss* value of a constraint. This can be seen as concentrating the conflict analysis on (falsified) literals which have been responsible for the conflict.

4.4 Example

Let us consider the following example to illustrate the discussion.

$$\begin{cases} (a) & 5x_1 + 3x_2 + 2x_3 + 2x_4 + x_5 = 8 \\ (b) & x_1 + x_3 + x_4 \geq 2 \end{cases}$$

Using the transformation discussed previously, this formula is translated in

$$\begin{cases} (a_1) & 5x_1 + 3x_2 + 2x_3 + 2x_4 + x_5 \geq 8 \\ (a_2) & 5\bar{x}_1 + 3\bar{x}_2 + 2\bar{x}_3 + 2\bar{x}_4 + \bar{x}_5 \geq 5 \\ (b) & x_1 + x_3 + x_4 \geq 2 \end{cases}$$

At the top decision level (DL = 0), x_5 is assigned 0. (a_1) becomes implicative and x_1 is assigned to 1.

At the following decision level (DL = 1), x_4 is assigned 0. (b) becomes implicative and x_3 is assigned to 1. (a_1) becomes also implicative and x_2 is assigned to 1, which leads to a conflict with (a_2) .

Resolution should be made on (a_1) and (a_2) wrt x_2 . But at decision level 1, the *poss* value of (a_1) is +2, of (a_2) is -2. The sum is equal to 0, so (a_1) has to be reduced. Reduction can be obtained by deleting x_1 , then x_3 . The constraint obtained is now

$$(a_3) \quad x_2 + x_4 + x_5 \geq 1$$

which has a *poss* value of 0. Resolution on (a_3) and (a_2) returns (value and decision level are noted `value@level`)

$$(c) \quad 2\bar{x}_1(0@0) + 2\bar{x}_3(0@1) + x_4(0@1) + 2x_5(0@0) \geq 2$$

which is conflictual, and assertive at decision level 0 (x_3 is implied to 1). This constraint contains two variables on the current decision level which could not be the case for clause learning. Furthermore, this constraint is stronger than

$$x_4 + x_5 \geq 1$$

which is the clause that could have been learned.

5 Implementation issues

There are several issues that prevent the implementation of a fast general Conflict Driven Constraint Learning solver.

Computing conflicting constraints The basis of CDC_L solvers is to derive conflicting clauses from a conflict and to keep an assertive one. While the resolution between two conflicting clauses results in a conflicting clause, it is no longer the case with PB constraints.

Detecting assertive constraints We noticed earlier that while assertive clauses could be detected using a simple syntactical test, this is no longer true in the case of PB constraints. The success of Chaff was to limit the work needed to create an assertive constraint by introducing the first UIP scheme. For PB constraints, the time needed to compute an assertive constraint may be much higher, because of the additional work needed to keep learnt constraints falsified. Even worse, while decision variables were UIP for clauses, they are no longer an upper bound for the conflict analysis process. Many resolution steps may be needed to compute an assertive constraint. The benefit however of that freedom against decision values is that conflict analysis may detect a global inconsistency even if there are some decision truth values [7], which is impossible in the clausal case.

Computing the backtrack level The backtrack level can be easily deduced from a deduced clause. Note it is much more challenging to deduce it from a PB constraint. As pointed out in [3] when a constraint is assertive at different decision levels, it is important to backtrack at the highest level, then eventually undoing numbers of assignments, in order to maintain logical consistency within the solver.

6 Conclusion

We describe in this paper Conflict Driven Clause Learning SAT solvers from a wider viewpoint: as a consequence, we notice that some features we take for

granted using clauses (such as UIP for instance), are no longer true when using more general constraints.

The reasoning scheme used to derive assertive constraints is very important. While SAT solvers are using resolution, a more powerful mechanism is needed to exploit the information provided by PB constraints. Furthermore, a promising way of research for SAT is to find other reasoning schemes to produce assertive constraints.

Chai and Kuehlmann conclude their work saying that building an efficient CDC_oL PB solver using PB reasoning is difficult because of all the additional computations needed for learning PB constraints. They also report that using cardinality constraints reasoning is a good trade-off between efficiency and reasoning power.

At the light of the results on pseudo boolean constraints, a promising research topic for a general CDC_oL solver in the spirit of [15] would be to find additional reasoning mechanisms better than plain resolution on boolean gates.

A preliminary implementation of a CDC_oL solver based on the work of Chai *et al* [3] is available at <http://www.sat4j.org/>.

References

- [1] Peter Barth. Linear 0-1 inequalities and extended clauses. Technical Report MPI-I-94-216, Max-Plank-Institut fr Informatik, Saarbrücken, Germany, 1994.
- [2] Peter Barth. A Davis-Putnam based enumeration algorithm for linear pseudo-Boolean optimization. Technical Report MPI-I-95-2-003, Max-Plank-Institut fr Informatik, Saarbrücken, 1995.
- [3] Donald Chai and Andreas Kuehlmann. A fast pseudo-boolean constraint solver. In *ACM/IEEE Design Automation Conference (DAC'03)*, pages 830–835, Anaheim, CA, 2003.
- [4] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. In *Communications of the Association for Computing Machinery* 5, pages 394–397, 1962.
- [5] M. Davis and H. Putnam. A computing procedure for quantification theory. In *Journal of the ACM*, 7, pages 201–215, 1960.
- [6] Heidi E. Dixon Matthew L. Ginsberg. Combining satisfiability techniques from AI and OR. In *The Knowledge Engineering Review* 15, page 53, 2000.
- [7] Heidi E. Dixon Matthew L. Ginsberg. Inference methods for a pseudo-boolean satisfiability solver. In *Proceedings of The Eighteenth National Conference on Artificial Intelligence (AAAI-2002)*, pages 635–640, 2002.

- [8] Joao P. Marques-Silva and Karem A. Sakallah. GRASP - A New Search Algorithm for Satisfiability. In *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, pages 220–227, November 1996.
- [9] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, pages 530–535, 2001.
- [10] Heidi E. Dixon Matthew L. Ginsberg Andrew J. Parkes. Generalizing boolean satisfiability i: Background and survey of existing work. In *Journal of Artificial Intelligence Research* 21, 2004.
- [11] S. Prestwich. Randomised backtracking for linear pseudo-boolean constraint problems. In *Proceedings of Fourth International Workshop on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimisation Problems (CP-AI-OR'2002)*, pages 7–20, 2002.
- [12] S. Prestwich. Incomplete dynamic backtracking for linear pseudo-boolean problems: Hybrid optimization techniques. *Annals of Operations Research*, 130(1-4):57–73, August 2004.
- [13] Fadi A. Aloul Arathi Ramani Igor L. Markov Karem A. Sakallah. Symmetry-breaking for pseudo-boolean formulas. In *International Workshop on Symmetry on Constraint Satisfaction Problems (SymCon)*, pages 1–12, County Cork, Ireland, 2003.
- [14] Niklas En Niklas Srensson. An extensible sat-solver. In *Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing, LNCS 2919*, pages 502–518, 2003.
- [15] Christian Thiffault, Fahiem Bacchus, and Toby Walsh. Solving Non Clausal Formulas with DPLL Search. In *Proceedings of the Tenth International Conference on Principles and Practice of Constraint Programming (CP'2004)*, Toronto, Canada, September 2004.
- [16] J. P. Walser. Solving Linear Pseudo-Boolean Constraint Problems with Local Search. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, pages 269–274, 1997.