

Exploring hybrid algorithms for SAT

Olivier Fourdrinoy, Éric Grégoire, Bertrand Mazure and Lakdhar Saïs

CRIL CNRS & IRCICA

Rue Jean Souvraz SP 18 F-62307 Lens Cedex France
{fourdrinoy,gregoire,mazure,sais}@cril.univ-artois.fr

Abstract. In this paper, several possible improvements of the combination scheme of systematic DPLL-like and local search techniques for SAT proposed by Mazure et al. are explored. Three important parameters that need to be tuned are investigated. A new weighting heuristic is described. Then, it is investigated how local search strategies that cover more diversified parts of the search space can prove useful in this combination scheme. Finally, it is studied when local search is best called within this hybrid DPLL-like algorithm.

1. Introduction

Various SAT solvers have been proposed these last years, leading to a dramatic breakthrough in the practical handling of large and hard instances. Most of them are based on one of the following two search paradigms: systematic (DPLL-like) and local search (GSAT-like). Each of these search techniques outperforms the other one with respect to some classes of instances. The combination of these search techniques can thus be a promising way to increase the general performance of these solvers. Indeed, hybridising these search methods is currently a hot topic of research within the SAT community. In [8], local search (LS) is considered as a branching heuristic for a DPLL-like algorithm. More precisely, at each node of the DPLL decision tree, a LS is performed on the currently remaining SAT sub-problem. During this LS step, a weight is computed for each variable. If LS fails to prove consistency within a pre-set amount of time, then DPLL selects the variable with the highest weight as the branching one. This basic hybrid approach proved to be efficient with respect to various sets of large and hard SAT instances [8]. However, it could still be improved in various ways. In this paper, three possible improvements are investigated. First, a new weighting heuristic is proposed. Then, it is experimented how LS strategies that cover more diversified parts of the search space can prove useful in this combination scheme. Finally, it is studied when local search is best called within this hybrid DPLL-like algorithm.

2. Weighting heuristics for falsified clauses

Although LS is by itself unable to prove inconsistency, it can provide us with some useful information that can help inconsistency to be detected. Let us elaborate on this. Any inconsistent SAT instance contains one or several inconsistent cores [2,4,9,11]. This means that only one (or several) subset(s) of variables (and/or clauses) is (are) causing the unsatisfiability of the instance. LS can often help us to identify these subsets, at least to some extent [8]. Before introducing heuristics to that end, let us recall some useful definitions about inconsistent cores.

Definition 1

A SAT instance Σ is *globally inconsistent* iff Σ is inconsistent and for every set of clauses Π such that $\Pi \subset \Sigma$, Π is consistent.

When an inconsistent SAT instance is not globally inconsistent, it is called *locally inconsistent*. It is possible to define several forms of *degree of locality*. Indeed, a concept of degree of locality can be defined based on some form of ratio between the size (in terms of the number of involved clauses) of the smallest inconsistent sub-instances and the size of the initial instance. Unsatisfiable sub-instances are called *inconsistent cores*.

Definition 2

Let Σ be an inconsistent SAT instance.

Π is an *inconsistent core* of Σ iff $\Pi \subseteq \Sigma$ and Π is inconsistent.

Π is an *overall inconsistent core* of Σ iff $\Pi \subseteq \Sigma$ and Π is globally inconsistent.

An overall inconsistent core Π of Σ is *minimally inconsistent* iff for all overall inconsistent cores Ω of Σ , $|\Pi| \leq |\Omega|$.

Property 1

Let Σ be an inconsistent SAT instance. For all interpretations I of Σ , at least one clause of each inconsistent core of Σ is falsified.

Based on Property 1, for each clause, the number of times that it is falsified during a failed LS can be counted as an attempt to locate or approximate inconsistent cores [8]. Indeed, intuitively, the most often falsified clauses should have a higher chance of belonging to an overall inconsistent core. As described in the introduction, this heuristic was used in [8]; a call to LS at each branching node of the DPLL search tree delivers the candidate branching variable. We believe that this approach can be refined in several ways.

First, let us note that LS often encounters a sharp decrease with respect to the number of falsified clauses during the first flips. Indeed, the number of falsified clauses after the random selection of an initial truth assignment can be high. In general, the number of falsified clauses decreases quickly during the first flips. It seems natural to think that information collected during this sharp descent is not relevant to locate in-

consistent cores. This can be compared to an initial noise phenomenon. Let us investigate two different strategies to address such an issue.

3. A local minima-based strategy

A first candidate strategy would rely on a threshold that would define a maximal number of falsified clauses. The counting heuristic would be inhibited each time the number of falsified clauses is greater than this threshold. Obviously enough such a strategy can only be useful when inconsistent cores are “small”. Hopefully, many non-random SAT instances do exhibit quite small inconsistent cores [8]. However, the main remaining open problem with the strategy is how the threshold should be determined. Moreover, the optimal value of the threshold should depend on the nature of the instance. Let us note that if the instance contains n mutually independent overall inconsistent cores, leading its Max-sat value to be $c-n$ where c is the number of clauses of the instance, then the threshold must be at least equal to n . Otherwise, no clause is weighted. Unfortunately, we do not have any reliable oracle informing us about the number of overall inconsistent cores.

A local minimum is a non-solution state where no flip of variable can lead the number of falsified clauses to be decreased. Another strategy [5] would consist in counting falsified clauses in local minima, only. To some extent, the numbers of falsified clauses in local minima can appear as dynamic thresholds. Our preliminary experimental validation of this new heuristic is very promising. It outperforms the threshold-based strategy. Although it yields weights that are similar to the ones obtained thanks to the initial heuristic of [8], it is less time-consuming.

As an illustration of the experimental tests that were conducted, Table 1 shows the number of clauses belonging to a minimally inconsistent core among the ten highest weighted clauses. The threshold strategy is investigated using different thresholds ($c/2$, $c/5$, $c/10$, $c/100$ and 1). It is compared with this local minima-based strategy and with the initial one where weights are computed in a systematic way. Such a test has been conducted on many instances. Only one instance (*aim-200-2_0-no-1.cnf*) is presented, because results for other instances are similar and it is known that this specific instance exhibits only one minimally inconsistent core.

Table 1. Threshold comparison

Threshold	Global	$c/2$	$c/5$	$c/10$	$c/100$	1	Local minima
Clauses from core	9	9	9	8	9	5	9
Weighting calls	2000	2000	2000	1980	1200	80	1250

4. Towards a more diversified exploration of the search space

The second possible improvement of the hybrid method of [8] concerns the initial interpretation that is selected by each LS run. The goal is to perform a more diversi-

fied exploration of the search space. Indeed, the more diversified exploration of the search space is, the better the weighting strategy could be. In [8], the initial interpretation of all called LS is chosen randomly. On the contrary, we define the initial interpretation for the current LS, based on the last failure (i.e. the last backtrack in DPLL or last interpretation of the previous LS). As a case study, we investigate a technique requiring us to reverse the value of all variables (not already assigned by DPLL). Such a “global flip” is also called a mirror. In this way, LS focuses on the last encountered problems while considering the opposite configuration.

5. Strategies for calling LS

LS, as it is grafted in DPPL in [8], is time consuming since it is called at *each* node of the DPLL search tree. Such a systematic call to LS could be relaxed as far as the current variable weights remain relevant for successive nodes in the DPLL search trees. In this respect, three strategies for calling LS are investigated.

1. LS is run systematically, i.e. for all nodes in the DPLL search tree [8]
2. LS is run as a pre-processing step, i.e. it is called once, only (like in [3] and [7])
3. LS is run until a pre-set depth has been reached. Then, a traditional DPLL branching rule heuristic is used.

5.1 Methodology

We have implemented and compared these three strategies. As a preliminary experimental validation procedure, a simple LS algorithm, namely WSAT [10], and the basic branching MOMS heuristic [6] were considered. Basic versions of DPLL were implemented, not including the above local-minima and mirror techniques. Two families of instances have been tested:

1. AIM instances resulting from DIMACS [1], as they exhibit interesting properties: satisfiable AIM instances admit only one model whereas the unsatisfiable ones exhibit one minimally inconsistent core.
2. random instances: obtained from the traditional generation model [2]. Each group of instances contains 300 problems (6 groups of 50 problems with 50, 100, 150, 200, 250 and 300 variables, respectively). These random instances are divided into four subsets. The first one (Rand@3.25) contains easy instances that are satisfiable and that are located before the phase transition threshold. Then, Rand@4.25s and Rand@4.25u contain hard satisfiable and unsatisfiable, respectively. These instances were generated at the 4.25 threshold. Finally, Rand@5.25 contains some easy unsatisfiable instances located after the threshold. They are thus intended to represent instances with a lot of models (Rand@3.25), instances where models are grouped inside clusters with a small number of clusters (Rand@4.25s), instances almost globally inconsistent with

large inconsistent cores (Rand@4.25u) and instances with a lot of small inconsistent cores (Rand@5.25).

These tests have been conducted on a Pentium 3 2.4Ghz under Linux Fedora core 2; a time out was set to 1000 seconds.

Table 2. AIM and random instances results

Instances	DP_LS_ALL	DP_LS_PRE	DP_LS_DEPTH	WSAT
Aim_yes	710.07	611.55	502.29	530.18
Aim_no	1750.21	1529.30	557.69	-
Rand@3.25	0.2	0.10	0.34	0.15
Rand@4.25s	329.04	302.32	231.43	262.32
Rand@4.25u	425.43	367.61	331.17	-
Rand@5.25	325.4	291.08	61.33	-

5.2 Results and comments

Table 2 summarises the results, reporting the average times in seconds to solve the instances. DP_LS_ALL, DP_LS_PRE and DP_LS_DEPTH represent the DPLL-based solvers:

- with systematic calls to LS [8];
- with a single call to LS as a pre-processing step;
- calling LS until a pre-set depth (set to 5, as a case study) has been reached, before a standard DPLL branching rule heuristic is applied.

As we hoped it, a limited number of calls to LS appeared to be more efficient to solve unsatisfiable instances. Indeed,

- the single LS run approach suffers from the fact that LS explores limited parts of the search space, only. The branching rule of DPLL is thus based on findings about a specific subset of the search space, only. In this respect, it seems that DP_LS_PRE is probably limited by the lack of reactivity of such a LS-based branching heuristics.
- DP_LS_ALL is handicapped by the time consumed by the numerous calls to LS.

6. Conclusions and perspectives

In this paper, several improvements of the combination scheme proposed in [8] have been proposed. Three specific points have been addressed. First, a new constraint weighting heuristic has been described. Then, a new LS strategy to diversify the part of the explored search sub-space has been proposed. Finally, several strategies for calling LS have been experimentally studied. Our preliminary experimental results appear to validate our expectations and they encourage us to continue in this way. We are currently working on a very extensive validation of the approach and are currently working on an implementation of such hybrid techniques in compet-

itive current solvers. We are also investigating to which extent the DP_LS_DEPTH heuristic limits the search to a small number of clusters of models.

Acknowledgements

We thank the anonymous reviewers for their numerous comments that helped us to improve the paper. This work has been supported by the *Région Nord/Pas-de-Calais*.

References

- [1] Y. Asahiro, K. Iwama, and E. Miyano Random Generation of Test Instances with Controlled Attributes, In *Cliques, Coloring, and Satisfiability: The Second DIMACS Implementation Challenge*. Vol. 26 of DIMACS, 377-394, 1996.
- [2] V. Chvatal and E. Szemerédi, Many hard examples for resolution, *Journal of the ACM*, 35(4):759–768, 1988.
- [3] J. M. Crawford & L.D. Auton, Experimental results on the crossover point in satisfiability problems, *Proceedings of AAAI'93*, 21-27, 1993.
- [4] C. P. Gomes, B. Selman, N. Crato and H. Kautz. Heavy-tail phenomena in satisfiability and constraint satisfaction, *Journal of Automated Reasoning*, 24(1-2):67-100 2000.
- [5] F. Hutter, D. A. D. Tompkins, H. H. Hoos. Scaling and Probabilistic Smoothing: Efficient Dynamic Local Search for SAT. *Proc. of the Eighth Int. Conf. on Principles and Practice of Constraint Programming*, Lecture Notes in Computer Science vol. 2470, 233-248, 2002.
- [6] R. J. Jeroslow and J. Wang, Solving propositional satisfiability problems, *Annals of Mathematics and Artificial Intelligence*, 1:167-188, 1990.
- [7] B. Mazure. De la satisfaisabilité à la compilation de bases de connaissances propositionnelles, Phd Thesis, Université d'Artois, CRIL, Lens, France, 1999.
- [8] B. Mazure, L. Saïs, and É. Grégoire. Boosting complete techniques thanks to local search methods. *Annals of Mathematics and Artificial Intelligence*, 22:319-331, 1998.
- [9] R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman and L. Troyansky. Determining computational complexity from characteristic 'phase transitions', *Nature*, 133-137, 1999.

- [10] B. Selman, H. A. Kautz, and B. Cohen. Noise strategies for improving local search. *Proceedings of AAAI'94*, 337-343, 1994.
- [11] R. Williams, C. P. Gomes and B. Selman, Backdoors to typical case complexity, *Proceedings of IJCAI'03*, 1173-1178, 2003.
- [12] S. D. Prestwich, Local search and backtracking vs. non-systematic backtracking. *AAAI Fall Symposium on Using Uncertainty within Computation*, Technical Report FS-01-04, AAAI Press, 109-115, 2001.