

Using failed local search for SAT as an oracle for tackling harder A.I. problems more efficiently*

Éric Grégoire, Bertrand Mazure, and Lakhdar Saïs

CRIL-CNRS – Université d’Artois
Rue Jean Souvraz SP-18
F-62307 Lens Cedex France
{gregoire,mazure,sais}@cril.univ-artois.fr

Abstract. Local search is often a suitable paradigm for solving hard decision problems and approximating computationally difficult ones in the artificial intelligence domain. In this paper, it is shown that a smart use of the computation of a local search that *failed* to solve a NP-hard decision problem \mathcal{A} can sometimes slash down the computing time for the resolution of computationally *harder* optimization problems containing \mathcal{A} as a sub-problem. As a case study, we take \mathcal{A} as SAT and consider some $P^{NP[\mathcal{O}(\log m)]}$ symbolic reasoning problems. Applying this technique, these latter problems can often be solved thanks to a small constant number of calls to a SAT-solver, only.

Keywords. SAT, local search, minimal models, logic and AI.

1 Introduction

Many hard instances of the NP-complete SAT decision problem can be solved using local search algorithms [20, 19, 15, 12]. Actually, these latter algorithms for SAT were first developed to approximate the MAXSAT optimization problem [9]. Indeed, when the local search fails to solve a decision problem, its result or some of its computations deliver an approximate solution of the corresponding optimization problem.

In this paper, this idea is pushed a step further: failing to prove a decision problem \mathcal{A} can sometimes help us in solving some *harder*¹ symbolic reasoning problems more efficiently. As a case study, we take \mathcal{A} as SAT and consider some $P^{NP[\mathcal{O}(\log m)]}$ optimization problems: namely, computing a preferred maximal consistent subset of clauses and computing a preferred minimal model, respectively. It is shown that a smart use of the computation of a local search that failed to solve a SAT instance can sometimes slash down the computing time for the resolution of these optimization problems. More precisely, it appears that they can often be solved in this way thanks to a small constant number of calls to a SAT-solver, only.

* This work has been funded in part by the CNRS, the IUT de Lens, the *Région Nord/Pas-de-Calais* and by the EC under a FEDER program.

¹ Unless P=NP

But let us start back from the basic consideration that local search algorithms do not cover the whole search space. Accordingly, they cannot conclude that a SAT instance is unsatisfiable, i.e. they cannot solve the dual co-NP-complete UNSAT problem. Recently, we have discovered the following two heuristics about the work performed by local search techniques when they fail to prove the consistency of SAT instances [13].

HEURISTIC 1

Let us count the number of times each clause has been falsified during the failed local search and count the number of times a Boolean variable has occurred in the unsatisfied clauses. The clauses with the highest scores form an approximation of the set-theoretic union of the minimal inconsistent subsets of the instance. Likewise, the Boolean variables with the highest scores take part in these subsets, most probably.

HEURISTIC 2

Focusing a further complete search algorithm on the clauses and on the Boolean variables with the highest scores can often make the instance solved efficiently.

We refer the reader to [13, 11] for the experimental work that gave rise to these heuristics. In order for Heuristic 1 to provide the best possible approximation, the parameters of the local search algorithm should be fine-tuned as to give a good coverage of the search space (mainly, the number of tries should be large and the distance between initial configurations should be maximized). The second heuristic has been introduced independently and in some implicit way by Crawford in the context of hard 3-SAT instances [3]. However, this failed as Heuristic 2 is generally only helpful when Heuristic 1 delivers an approximation of the minimal inconsistent subsets that can be handled by the best complete techniques for SAT. Clearly, this is not the case for large random k-SAT instances at the phase transition, since the size of these subsets does not diverge significantly enough from the size of the initial instances [13, 2].

2 Basic applications to SAT and MAX-SAT

Before we climb the polynomial hierarchy and show how these SAT-related heuristics can help with respect to harder optimization problems, let us briefly review how using failed local search can help with respect to MAX-SAT and SAT themselves.

When a local search algorithm fails to prove that a SAT instance is satisfiable, the largest set of simultaneously satisfied clauses during the failed search is an approximation of the largest consistent subset of the instance, i.e. gives an approximate solution to the MAXSAT problem [9]. Obviously enough, when the search succeeds in showing that the instance is satisfiable, then the MAXSAT problem is solved.

The local search technique can itself take its previous failed steps into account. For instance, [17] attached a dynamic weight on clauses that increases

with the number of times these latter ones have been previously falsified. Focusing on these clauses, a steepest descent towards and a better approximation of the solution can be obtained, as illustrated in Table 1, where GSAT [20] and GSAT+weight [17] are compared on DIMACS instances [5]. In the table, #unsat represents the lowest number of falsified clauses during the search.

Instances	Sat	Size		GSAT		GSAT+Weight	
		Var.	Cla.	time	#unsat	time	#unsat
<i>BF series:</i>							
0432-077	No	1044	3685	5s99	18	6s11	1
1355-240	No	2298	7307	91s42	89	57s94	2
2670-208	No	1379	3423	12s80	25	16s08	1
<i>SSA series:</i>							
0432-001	No	435	1027	0s49	6	0s43	1
2670-128	No	1359	3296	12s78	29	7s92	2
6288-047	No	10410	34238	102s78	425	101s17	56
7552-083	Yes	1448	3298	13s72	32	0s06	0
<i>AIM series:</i>							
200-1.6-yes1-1	Yes	200	320	0s07	1	0s02	0
200-1.6-no-3	No	200	320	0s06	1	0s08	1
<i>II series:</i>							
8b4	Yes	1068	8214	3s42	1	0s08	0
16a1	Yes	1650	19368	33s27	1	0s09	0
32d3	Yes	824	19478	11s94	36	1s70	0

Table 1. GSAT vs GSAT+weight⁴

In [13, 11] several ways to exploit Heuristic 2 in order to solve SAT instances efficiently have been proposed. For instance, DP+TSAT is a combination of a Davis, Logemann and Putnam [4] complete procedure (in short, DP) that proves competitive in most situations. It starts with a call to a tabu-based local search algorithm (TSAT) [12, 14]. If this one fails to prove consistency, then a DP procedure is run. The branching heuristic of DP selects the most often falsified literal by a previous call to a local search algorithm. In Table 2, the very good performance of DP+TSAT is illustrated on various benchmarks from [5].

3 Computing preferred maximal consistent subsets of clauses

Let us now climb the polynomial hierarchy and select some recurrent combinatorial optimization problems in the artificial intelligence community as a case study. A first one consists in computing maximal (set-theoretic) consistent subsets of clauses, while obeying a preference relation between clauses.

² All experimental results in this paper have been obtained on Pentium III 350Mhz under linux kernel 2.2.12. All experimental data are available at: <http://www.cril.univ-artois.fr/~mazure>.

Instances	Sat	Size		Classical DP			DP+TSAT		
		Var.	Cla.	#A	#CH	time	#A	#CH	time
<i>AIM series:</i>									
100-1.6-no-3	No	100	160	3E+07	2E+06	214s71	178	16	0s26
100-1.6-yes1-2	Yes	100	160	495858	30052	4s39	77	6	0s08
100-2.0-no-1	No	100	200	4E+07	2E+06	349s52	46	5	0s10
100-2.0-yes1-1	Yes	100	200	706388	31274	7s41	81	8	0s15
200-1.6-no-1	No	200	320	***	***	>8h	240	16	0s58
200-1.6-yes1-3	Yes	200	320	***	***	>9h	232	11	0s32
200-2.0-no-3	No	200	400	***	***	>15h	120	10	0s42
200-2.0-yes1-1	Yes	200	400	2E+09	7E+07	21859s45	291	27	1s21
50-1.6-no-1	No	50	80	12072	895	0s09	72	8	0s06
50-1.6-yes1-1	Yes	50	80	1540	84	0s01	37	6	0s05
50-2.0-no-1	No	50	100	54014	2759	0s43	52	5	0s05
50-2.0-yes1-1	Yes	50	100	2878	176	0s03	11	3	0s03
<i>BF series:</i>									
0432-007	No	1040	3668	9E+08	6E+06	19553s44	115766	870	85s25
1355-075	No	2180	6778	317628	2047	18s88	4602	28	26s23
1355-638	No	2177	4768	***	***	>17h	6192	32	32s57
2670-001	No	1393	3434	***	***	>25h	490692	4822	519s40
<i>SSA series:</i>									
0432-003	No	435	1027	133794	1570	1s79	1338	16	0s80
2670-130	No	1359	3321	***	***	>33h	2E+07	79426	8040s64
2670-141	No	986	2315	3E+08	2E+06	6350s77	1E+07	92421	6639s44
7552-038	Yes	1501	3575	***	***	>13h	29	1	0s34
7552-158	Yes	1363	3034	1639	78	0s19	12	1	0s29
7552-159	Yes	1363	3032	1557	84	0s21	12	1	0s25
7552-160	Yes	1391	3126	1457	76	0s18	1	1	0s30

(#A = Number of assignments #CH = Number of choices)

Table 2. DIMACS problems

For clarity of presentation, we first need to recall the concept that is dual to the notion of maximal consistent subset of clauses, namely the concept of *minimal inconsistent kernel*. In the following, \mathcal{C} is a finite set of Boolean clauses.

DEFINITION 1

A minimal inconsistent kernel (*in short*, kernel) of \mathcal{C} is a subset of \mathcal{C} that is both inconsistent and minimal with respect to set-theoretic inclusion.

Clearly, \mathcal{C} might contain several different kernels in the general case. Also, dropping one clause belonging to a kernel of \mathcal{C} is enough to *break* the kernel, i.e. to suppress it from the set of kernels of \mathcal{C} .

DEFINITION 2

The inconsistent part of \mathcal{C} , noted $\text{UMIN-UNSAT}(\mathcal{C})$, is the set-theoretic union of all kernels of \mathcal{C} .

The dual concept is thus the notion of maximal consistent subset of \mathcal{C} ; computing one of them is the target of the MAXSAT optimization problem. Let

us redefine this notion when a preference pre-ordering applies on clauses of \mathcal{C} , which is often the case in many artificial intelligence applications. Accordingly, this gives rise to an extended form of MAXSAT that accommodates a preference relation between clauses.

Assume thus that the m clauses in \mathcal{C} are partitioned inside n strata $S_1 \cup \dots \cup S_n$. For simplicity of presentation, we assume that the m clauses are numbered in a way that follows the strata, i.e. when $c_i \in S_k$, $c_j \in S_r$ and $k > r$, then $i > j$. When $c_i \in S_k$ and $c_j \in S_r$ such that $k > r$ that means that c_j is preferred over c_i , i.e. if we must choose to drop one of the two clauses to get a maximal consistent subset then we drop c_i . Accordingly, the strata translate a preference complete pre-ordering on the clauses of \mathcal{C} . Let $A = A_1 \cup \dots \cup A_n$ and $B = B_1 \cup \dots \cup B_n$ be two consistent subsets of \mathcal{C} , where $A_i = A \cap S_i$ and $B_i = B \cap S_i$.

DEFINITION 3 (*Benferhat et al.*)[1]

$A \ll_{\{S_1 \cup \dots \cup S_n\}} B$ iff $\exists i$ s.t. $A_i \subset B_i$ and $\forall j < i$, $A_j = B_j$. A consistent subset of \mathcal{C} that is maximal w.r.t. $\ll_{\{S_1 \cup \dots \cup S_n\}}$ is called a preferred maximal consistent subset of \mathcal{C} (w.r.t. $S_1 \cup \dots \cup S_n$).

In some artificial intelligence problems, like model-based diagnosis [18, 8], a *single kernel* assumption is often assumed. In this respect, the following definition will prove useful.

DEFINITION 4

The set of highest cancelable clauses of \mathcal{C} , noted $\cup\text{MIN-UNSAT}(\mathcal{C})_{\text{highest}}$ is the set of clauses that are maximal according to the complete pre-ordering \ll between clauses in $\cup\text{MIN-UNSAT}(\mathcal{C})$.

Under the single kernel assumption, dropping one element of $\cup\text{MIN-UNSAT}(\mathcal{C})_{\text{highest}}$ delivers a preferred maximal consistent subset of \mathcal{C} . Computing one element of $\cup\text{MIN-UNSAT}(\mathcal{C})_{\text{highest}}$ is however computationally heavy in the worst case; it is easy to show that it is polynomial under a number of calls to an NP-oracle that is logarithmic with respect to the number of strata (which, in the worst case, is the number of clauses of \mathcal{C}).

PROPOSITION 1

When the single kernel assumption applies, computing a preferred maximal consistent subset of \mathcal{C} belongs to $P^{NP[\mathcal{O}(\log n)]}$, where n is the total number of strata.

However, when Heuristics 1 and 2 apply, this task can be achieved more efficiently for most sets of clauses, reducing the logarithmic number of calls to an NP-oracle to only 4 calls to a fast satisfiability check very often.

PROPOSITION 2

When both the single kernel assumption and Heuristics 1 and 2 apply, computing a preferred maximal consistent subset of \mathcal{C} is polynomial under 4 calls to an NP-oracle.

The algorithm is given in the [6]. The main idea is the following. A local search algorithm is run on \mathcal{C} . If it fails to prove the consistency of \mathcal{C} after a preset computing time, a complete DP-based search is performed. When \mathcal{C} is inconsistent, the trace of the search delivers a list \mathcal{L} of clauses c that are sorted in decreasing order w.r.t. the number of times they have been falsified during the search. \mathcal{L} is re-organized in such a way that a clause that belongs to the most often falsified ones and that exhibits the highest stratum among these latter ones appears first. Under the single kernel assumption, the first clause c in the list \mathcal{L} belongs to $\cup\text{MIN-UNSAT}(\mathcal{C})_{\text{highest}}$, most probably. Accordingly, we check the consistency of $\mathcal{C} \setminus \{c\}$ and retract c from \mathcal{L} . If this leads to consistency, we just need one more consistency check to make sure that no preferable clause would lead to consistency when dropped. If this fails, we try the next elements of \mathcal{L} , successively.

In table 3, some typical experimental results are given. Large benchmarks were built, merging several DIMACS benchmarks [5] (sharing a same set of variables). A total order between clauses are assumed. Table 3 gives the initial DIMACS instances that were merged, the size of the final instances, the clauses to be dropped to obtain a maximal consistent preferred subset, and the global computing time.

Instances	Size		#Cla. tested	Time to		
	Var.	Cla.		compute the trace	find c	prove it gives a preferred sub-base
ssa7552-160+dubois10	1391	3206	41	88s62	52s35	4s13
ssa7552-094+pret150-60	1473	3842	400	89s77	869s14	2s68
ssa7552-156+aim-50-2_0-no-2	1444	3382	100	93s76	139s87	1s20

Table 3. Maximal consistent preferred sub-base

When we cannot assume that the single kernel assumption applies, then the problem becomes even harder.

PROPOSITION 3

Computing one preferred maximal consistent subset of \mathcal{C} belongs to $P^{NP[\mathcal{O}(m)]}$, where m is the total number of clauses in \mathcal{C} .

In [6], a basic algorithm that computes an approximate preferred maximal consistent subset of \mathcal{C} is given. Once again, it makes use of the failed local search intensively. It delivers a right set of clauses under the condition that Heuristic 1 allows us to extract a superset of $\cup\text{MIN-UNSAT}(\mathcal{C})$, which is often the case.

PROPOSITION 4

When the trace of the local search allows us to extract a superset of $\cup\text{MIN-UNSAT}(\mathcal{C})$ as the part of L that is interpreted as containing the most often falsified clauses, then Procedure 1 in [6] computes a preferred maximal consistent subset of \mathcal{C} .

In most realistic situations, the number of kernels is bounded by a small constant k . In this case, the computation of a preferred maximal consistent subset of \mathcal{C} can be reduced dramatically when the trace of the local search that runs on \mathcal{C} allows us to extract a superset of $\cup\text{MIN-UNSAT}(\mathcal{C})$.

PROPOSITION 5

When the trace of the local search allows us to extract a superset of $\cup\text{MIN-UNSAT}(\mathcal{C})$ and when the number of kernels of \mathcal{C} is bounded by k , then computing one preferred maximal consistent subset of \mathcal{C} is polynomial under k calls to a NP-oracle.

4 Computing preferred models of sets of clauses

Let us now turn our attention to another hard optimization problem. Let us illustrate it by means of an artificial intelligence application. The knowledge about a physical device is represented using a finite set \mathcal{C} of Boolean clauses. To represent possible faulty behaviors of the device, a pre-ordered set $(AB, <)$ of specific Boolean variables Ab_i [16] is used. For instance, the rule asserting that, under normal circumstances, when the switch is on then the lights should be on is represented by the formula $switch_on \wedge \neg Ab_{37} \Rightarrow lights_on$, and in clausal form by $\neg switch_on \vee Ab_{37} \vee lights_on$ (where \wedge , \vee , \neg , \Rightarrow represent the standard conjunctive, disjunctive, negation and material implication connectives, respectively). If at the same time we also have that both $switch_on$ and $\neg lights_on$ are *true*, then Ab_{37} must also be *true* in order for the whole set of formulas to remain consistent. In the normal circumstances of affairs, all markers can be assumed *false* while the system remains consistent. When the system requires some markers to be *true* in order to be consistent, then these latter markers indicate one possible failure.

Let us recall that a model is a truth assignment that satisfies all clauses from \mathcal{C} . A model is represented by the set of Boolean variables that it sets to *true*. Here, we are interested in finding a model of \mathcal{C} where no marker Ab_i from AB is *true*. If such a model does not exist, then we want to provide the user with the marker with the lowest index that must be *true* in order for the system to be consistent. The motivation is that we want to inform the user of the most important failure, which are supposed to be represented by markers Ab_i with lowest possible index i . This specific form of model-based reasoning is used to solve the Christmas Tree Syndrome (as it is called by airforce pilots when many alarms flash at the same time due to a single source problem, which is often vital to localize quickly). Such a problem is described in e.g. [7].

In the following, \mathcal{C} is assumed consistent. Let M_1 , M_2 and M_3 be models of \mathcal{C} .

DEFINITION 5

M_1 is a preferred model of \mathcal{C} iff

1. M_1 does not contain any Ab_i
2. or M_1 contains at least one Ab_i and, at the same time,
 - there does not exist a model M_2 of \mathcal{C} that does not contain any Ab_j ;
 - there does not exist a model M_3 of \mathcal{C} containing an Ab_j such that $Ab_j < Ab_i$ for all Ab_i in M_1 .

Computing one such preferred model is an heavy task in the general case.

PROPOSITION 6

Computing one preferred model of \mathcal{C} is in $P^{NP}[\mathcal{O}(\log m)]$, where m is the total number of strata in AB .

Fortunately, the technique that is proposed in this paper allows us to find a preferred model (and thus one most important failure) using a small constant number (often 3) calls to a fast propositional satisfiability prover (very often). It works as follows. First, we try to prove the absence of device failure by assigning all markers to *false* and showing that \mathcal{C} remains consistent under these circumstances. To this end, a combination of a local search technique with a complete DP algorithm is used. More precisely, if the local search technique delivers a model, then this proves the absence of failure. Else, a complete DP technique is run (possibly focusing on the trace delivered by the previous call to the local search algorithm). If a model is found, no failure is exhibited. Else, the algorithm takes the trace delivered by the local search into account. In order to be consistent, at least one marker should be *true*. In this respect, the following result proves essential.

PROPOSITION 7

Let D be any superset of $\cup \text{MIN-UNSAT}(\mathcal{C} \cup \{\neg Ab_i\}_{(\forall Ab_i \in AB)})$. Any model of \mathcal{C} contains at least one marker occurring in D .

Accordingly, if the heuristic about the trace is correct, then it indicates by means of the score of the clauses to which they belong, which markers should be *true* to get a model of \mathcal{C} . Fortunately, this heuristic is experimentally correct extremely often for realistic sets of clauses representing physical devices.

HEURISTIC 3

The trace of the failed (local) search of a model of $\mathcal{C} \cup \{\neg Ab_i\}_{(\forall Ab_i \in AB)}$ is a good oracle of the markers that are required to be true in order for \mathcal{C} to be consistent.

The idea is thus to select the marker with the lowest index in the clauses with the highest scores and set it to *true*. Assume it is Ab_i : if this leads to a model, it remains to check that no model containing a marker with a strictly lower index exists. This can be checked by testing the consistency of $\mathcal{C} \cup \{\neg Ab_i\} \cup \{Ab_1 \vee Ab_2 \vee \dots \vee Ab_r\}$, where the last clause is formed from all markers Ab_k such that $Ab_k < Ab_i$. If this leads to a model (low probability), then the process is iterated

with the marker *true* with the lowest index in this last clause. If no model can be found, then Ab_i is the *most important* failure to be reported to the user.

Obviously enough, this process might lead all markers to be considered successively and yield a $P^{NP[O(\log m)]}$ complexity where n is the number of markers (although a dichotomy-based approach could reduce it to the logarithmic result of Proposition 6). Fortunately, applying the heuristic often leads us to the result immediately.

In Table 4, typical experimental results are given for ISCAS [10] benchmarks, translated into the DIMACS format [5]. The time to compute a preferred model in case of device failure, and the total number of candidates Ab_i that had to be considered are given.

Instances	Size		Ab_i	# Ab_i tested	Time to obtain	
	Var.	Cla.			trace	pref. model
c17	29	73	6	2	0s00	0s02
c432	692	1903	160	5	0s02	1s61
c1355	2165	5293	514	0	0s00	0s12
c1908	3129	7880	718	3	1s00	16s75
c2670	4499	11200	997	3	1s05	27s48
c3540	6104	15720	1446	3	3s02	66s51
c5315	8865	23201	1994	2	6s06	76s91

Table 4. ISCAS problems

5 Conclusion

In this paper, it has been shown how the computation of a local search that failed to solve a SAT instance can prove useful in allowing *harder* symbolic reasoning problems to be solved more efficiently. We believe that similar results could be obtained with respect to other decision and optimization problems.

Obviously enough, the price to pay is the restricted applicability of the approach. In particular Heuristic 2 does not work well for large random instances at the transition phase, where really hard problems are often thought to be found. On the other hand, the heuristics that are used in this paper appear to work very often for realistic real-world problem instances.

References

1. S. Benferhat, C. Cayrol, D. Dubois, J. Lang, and H. Prade. Inconsistency management and prioritized syntax-based entailment. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI'93)*, pages 640–645, 1993.
2. Yacine Boufkhad and Olivier Roussel. Redundancy in random sat formulas. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI'00)*, pages 273–278, 2000.
3. James M. Crawford. Solving satisfiability problems using a combination of systematic and local search. In *Working notes of the DIMACS Workshop on Maximum Clique, Graph Coloring, and Satisfiability*, 1993.

4. Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem proving. *Journal of the Association for Computing Machinery*, 5:394–397, 1962.
5. Second Challenge on Satisfiability Testing organized by the Center for Discrete Mathematics and Computer Science of Rutgers University, 1993. <http://dimacs.rutgers.edu/Challenges/>.
6. Éric Grégoire. Handling inconsistency efficiently in the incremental construction of stratified belief bases. In A. Hunter and S. Parsons, editors, *Proceedings of the Fifth European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty (ECSQARU'99)*, volume 1638 of *Lecture Notes in Computer Science*, pages 168–178, London (UK), July 1999. Springer.
7. Éric Grégoire and David Ansart. Overcoming the christmas tree syndrome. *International Journal on Artificial Intelligence Tools (IJAIT)*, 9(2):97–111, 2000.
8. W. Hamscher, L. Console, and J. De Kleer. *Readings in Model-Based Diagnosis*. Morgan Kaufmann, 1992.
9. Pierre Hansen and Brigitte Jaumard. Algorithms for the maximum satisfiability problem. *Journal of Computing*, 22:279–303, 1990.
10. International Symposium on Circuits And Systems, 1985. http://www.eecs.umich.edu/~mhansen/imodels/ISCAS_HLM.html.
11. Bertrand Mazure. *De la Satisfaisabilité à la Compilation de Bases de Connaissances Propositionnelles*. Thèse de doctorat, Université d'Artois, Centre de Recherche en Informatique de Lens (Faculté Jean Perrin), January 1999.
12. Bertrand Mazure, Lakhdar Saïs, and Éric Grégoire. Tabu search for SAT. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI'97)*, pages 281–285, Providence (Rhode Island, USA), July 1997.
13. Bertrand Mazure, Lakhdar Saïs, and Éric Grégoire. Boosting complete techniques thanks to local search methods. *Annals of Mathematics and Artificial Intelligence*, 22:319–331, 1998.
14. Bertrand Mazure, Lakhdar Saïs, and Éric Grégoire. System Description: CRIL Platform for SAT. In *Proceedings of the Fifteenth International Conference on Automated Deduction (CADE'15)*, volume 1421 of *Lecture Notes in Artificial Intelligence*, pages 116–119, Lindau (Germany), July 1998.
15. David A. McAllester, Bart Selman, and Henry A. Kautz. Evidence for invariants in local search. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI'97)*, pages 321–326, August 1997.
16. J. McCarthy. Applications of circumscription to formalizing common-sense knowledge. *Artificial Intelligence*, 28:89–116, 1986.
17. P. Morris. The break out method for escaping from local minima. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI'93)*, pages 40–45, 1993.
18. R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32:57–95, 1987.
19. Bart Selman, Henry A. Kautz, and B. Cohen. Local search strategies for satisfiability testing. In *Working notes of the DIMACS Workshop on Maximum Clique, Graph Coloring, and Satisfiability*, 1993.
20. Bart Selman, Hector J. Levesque, and David Mitchell. Gsat: A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI'92)*, pages 440–446, 1992.