GUNSAT: a Greedy Local Search Algorithm for Unsatisfiability

Gilles Audemard[*]
Univ d'Artois, CRIL - CNRS, FRE2499,
Lens, F-62307
audemard@cril.univ-artois.fr

Laurent Simon
Univ Paris-Sud, LRI - CNRS, UMR8623,
INRIA-Futurs, Orsay, F-91405
simon@lri.fr

Abstract

Local search algorithms for satisfiability testing are still the best methods for a large number of problems, despite tremendous progresses observed on complete search algorithms over the last few years. However, their intrinsic limit does not allow them to address UNSAT problems. Ten years ago, this question challenged the community without any answer: was it possible to use local search algorithm for UNSAT formulae? We propose here a first approach addressing this issue, that can beat the best resolution-based complete methods. We define the landscape of the search by approximating the number of filtered clauses by resolution proof. Furthermore, we add high-level reasoning mechanism, based on Extended Resolution and Unit Propagation Look-Ahead to make this new and challenging approach possible. Our new algorithm also tends to be the first step on two other challenging problems: obtaining short proofs for UNSAT problems and build a real local-search algorithm for QBF.

1 Introduction

Over the last ten years, a lot of impressive progresses have been made in the practical solving of the satisfiability testing (SAT). All the methods that address this typical NP-Complete problem may be divided in two categories: Complete ones, usually based on the Davis, Logemann and Loveland procedure [Davis *et al.*, 1962; Moskewicz *et al.*, 2001], can prove that a formula is satisfiable or not. Incomplete solvers (also called *one-sided* solvers) only give an answer if the instance has a given property (generally if it is satisfiable). Most of them, which are based on local search, perform a stochastic walk over the search space [Selman *et al.*, 1994; Hoos and Stutzle, 2004]. Those methods give very good results on certain classes of problems, like random 3SAT problems [LeBerre and Simon, 2006].

In 1997, [Selman *et al.*, 1997] challenged the local search community with a quest for an efficient local search algorithm for unsatisfiable formulae. We propose here the first successful approach, called GUNSAT, that pushes the local search formalism into resolution proofs search space. Because state-of-the-art complete SAT solvers are terribly efficient in practice, we only call *efficient* a solver, based on resolution, that can defeat state-of-the-art resolution-based solvers. For doing this, we propose to make a greedy walk among the resolution search space in which, at each step of the local search algorithm, we try to compute a "*better*" neighbour proof, *i.e.* a proof which differs from the previous ones by at most two clauses, one added by resolution, and one that may have been removed. To find such a neighbour, we approximate the number of filtered models by the proof. This is achieved by a score given to all pairs of literals depending on their frequencies in the formula. As we will show it in our experimental investigation, the use of higher reasoning mechanism, based on Extended Resolution [Robinson, 1965] and Unit Propagation Look-Ahead [LeBerre, 2001] is a key to make this new and challenging approach possible. Because resolution-based methods seems to be an efficient way for solving quantified boolean formulae (QBF), we believe that our algorithm may be the first step on two other challenging problems: build a real local-search algorithm for QBF and obtain short proofs for UNSAT problems and QBF ones.

The paper is organised as follows. We start by introducing some definitions and notations, then, in section 3, we discuss previous works done on local search for unsat problems, and we present our new approach. Before conclude, we provide some experiments.

2 Preliminaries and definitions

Let $V = \{x_1, \ldots, x_n\}$ be a set of *boolean variables*, a *literal* l_i is a variable x_i or its *negation* $\neg x_i$. A *clause* is a disjunction of literals $c_i = l_1 \vee l_2 ... \vee l_{n_i}$. A *unit* clause (or *mono-literal*) is a clause restricted to a single literal. A formula Σ is in *conjunctive normal form* (CNF) if Σ is a conjunction of clauses $\Sigma = c_1 \wedge c_2 ... \wedge c_m$. The SAT decision problem is defined as follows: given a formula Σ in CNF, is there an assignment of the variables V Σ such that Σ is satisfied, *i.e.* all the clauses of Σ are satisfied?

2.1 Local Search for SAT problems

Local search algorithms for SAT problems use a stochastic walk over total interpretations of Σ (*i.e.* all variables are assigned). At each *step* (or *flip*), they try to reduce the number of unsatisfiable clauses (under certain conditions). The next

[*]supported by ANR "Planevo" project n^{o}JC05_41940.

Algorithm 1: General scheme of a local search SAT solver

```
Data    : $\Sigma$ a CNF formula
Result  : SAT if a model is found, UNKNOWN otherwise
begin
    for i=1 to MaxTries do
        Choose a random interpretation $I$;
        for j=1 to MaxFlips do
            if $I$ is a model of $\Sigma$ then return SAT ;
            $I$ = neighbour($I$);
        end
    end
    return UNKNOWN ;
end
```

total interpretation is chosen among the neighbours of the current one (they differ only on one literal value). After a given number of steps, a restart is done to jump into an other part of the landscape (*escape phase*). We recall, algorithm 1, the general scheme of local search algorithms for SAT.

A lot of improvements have been proposed in the literature, like *tabu search* or the use of different heuristics to find the next interpretations. The interested reader may refer to [Hoos and Stutzle, 2004] for a detailed coverage of all methods.

2.2 Resolution Proofs

The resolution rule allows to deduce a new clause (called resolvent) with the help of two clauses. It plays a central role in many works based on clausal representations of boolean formulae, like preprocessing steps [Eén and Biere, 2005].

Definition 1 ([Robinson, 1965]) *Let* $c_1 = (x \vee a_1 \vee a_2 \vee \dots a_n)$ *and* $c_2 = (\neg x \vee b_1 \vee b_2 \vee \dots b_m)$ *be two clauses. The clause* $c = (a_1 \vee a_2 \vee \dots a_n \vee b_1 \vee \dots b_m)$ *is called the* resolvent *of the clauses* c_1 *and* c_2 *by performing a* resolution *on the variable* x*. We note* $c = c_1 \otimes c_2$ *this rule.*

One may apply the resolution rule many times including on resolvent clauses to derive new clause. The obtained clause is *proved* by resolution:

Definition 2 (Resolution proof [Robinson, 1965]) *Let* Σ *be a CNF formula. A resolution proof of a clause* c *is a succession of clause* $P = c_1, c_2, \dots, c_k$ *such that* $c = c_k$ *and for all* $i \leq k$ *one of the following conditions holds :*

- $c_i \in \Sigma$
- $\exists c_m, c_n \in P$ *(*$m < i$ *and* $n < i$*) such that* $c_i = c_m \otimes c_n$

If the empty clause ($\perp$) may be proved by resolution then the formula Σ is unsatisfiable: The resolution proof system is complete for refutation. Restrictions on general resolution exist and may still answer both SAT and UNSAT questions (like *Ordered Resolution*). However, in practice, resolution-based solvers are more suitable to knowledge compilation problematics (preprocessing, prime implicates computation, ...) than to SAT checking. Directional Resolution [Dechter and Rish, 1994] is one of the most famous restriction, based on the well known work of [Davis and Putnam, 1960]. [Simon and del Val, 2001] has shown how a simple generalisation of the resolution rule, called multi-resolution, allowed to scale-up most of the resolution based solvers, essentially for compilation purpose. This two resolution-based solvers will help us measure the performances of GUNSAT. We call *efficient* a solver, based on the resolution rule, that can beat both of them.

Extended Resolution

It is well known that the number of necessary clauses could be exponential to prove refutation by general resolution proof (even if no restrictions are imposed in the order of resolutions steps). This is the case for the well known pigeon hole problem [Haken, 1985], the Urquhart problems [Urquhart, 1987] and even the random problems [Chvátal and Szemerédi, 1988]. We have here to notice that all those problems need *large* clauses (*i.e.* unbounded size) in the resolution proof before producing $\perp$. On a lot of instances, allowing large clauses to be generated is essential.

However, it is striking to notice that as soon as one allows the introduction of new variables during the proof (like lemmas in theorem proving), then short proofs for the above problems exist. [Tseitin, 1970] introduced this principle as the *extended resolution* rule. If Σ is a formula, one application of this rule allows us to consider $\Sigma \wedge (e \Leftrightarrow l_1 \vee l_2)$ instead, where e is a fresh variable (not appearing in Σ) and l_1 and l_2 are literals from Σ. Even if this rule looks very simple, there is no proof system known to be stronger than extended resolution. Despite its simplicity and its theoretical interests, no practical applications of extended resolution are used in modern SAT solvers. Some works implicitly use a restriction of it, like symmetry breaking algorithm based on the introduction of new variables. The work of [Simon and del Val, 2001] can also be viewed as a restriction of this rule, where nodes of the graph representation of the formula may be viewed as new propositional variables. However, at the end, the whole problem remains: even if this rule looks simple, how to choose the pair of literals to extend?

3 Local search for UNSAT problems

After a discussion of previous works, we present GUNSAT, our new local search algorithm for unsat checking.

3.1 Previous Work

Among the ten challenges proposed ten years ago [Selman *et al.*, 1997], not so much work has been done to address the fifth challenge: *"design a practical stochastic local search procedure for proving unsatisfiability"*.

Hybrid solvers

Almost all previous works on incomplete algorithms for unsatisfiability has been done on hybrid solvers only. Some methods use a classical local search solver but record *nogoods*, based on resolvent. However, in the general case, those methods improve performance on satisfiable problems only [Fang and Ruml, 2004; Shen and Zhang, 2005].

With the new interest in QBF solving and the relative failure of direct DPLL extensions on them, some effort have been done on finding new methods, based on resolution or local search. The solver WalkQSAT [Gent *et al.*, 2003] has two

Algorithm 2: GUNSAT

```
Data   : $\Sigma$ a CNF formula
Result : UNSAT if a derivation of $\bot$ is found, UNKNOWN otherwise
begin
    for i=1 to MaxTries do
        for j=1 to MaxFlips do
            if 2-Saturation($\Sigma$) returns UNSAT then
                return UNSAT ;
            if $|\Sigma| > MaxSize$ then
                Remove-One-Clause($\Sigma$)
            Add-One-Clause($\Sigma$);
            Add-Extended-Variables($\Sigma$);
            Simplify-Look-Ahead($\Sigma$);
        end
        Replace $\Sigma$ by all its vital clauses;
    end
    return UNKNOWN;
end
```

distinct phases: the first one is a complete algorithm based on DPLL for QBF; the second one use local search to quickly find models. This incomplete solver can prove either the validity and the invalidity of a QBF formula, but may never prove it.

RANGER

Ranger [Prestwich and Lynce, 2006] is also a local search algorithm for UNSAT problems that may looks very similar to GUNSAT. In this recent work, arbitrary resolution is performed in order to generate a large number of the shortest possible clauses, as fast as possible. To avoid combinatorial explosion, they may remove clauses. As we will see after, GUNSAT has a more powerful reasoning mechanism and a finest heuristic to guide moves whereas Ranger is simpler but performs many more moves per second.

3.2 Skeleton of our local search algorithm

The solver GUNSAT has, for a large part, the same skeleton than classical local search algorithms. Intuitively, our algorithm will remove and add new clauses to the set of clauses Σ, trying to derive $\bot$ by the resolution rule. The whole problem is to ensure that our algorithm may be able to perform *greedy moves* in the search space. It is also fundamental to add a higher level reasoning mechanism to ensure good practical results. The GUNSAT algorithm, detailed in the later, is given in algorithm 2.

3.3 Inside GUNSAT

Estimation of the number of filtered clauses

Given a set of clauses Σ built on n variables, we want to approximate the total number of its potential models that are directly filtered out (over the 2^n potential ones). The measure must be based on explicit information only, and the computational effort to obtain it must be polynomially bounded. Otherwise, one may need to exactly count the number of its models, which is clearly harder than the coNP-hard problem we are initially facing.

Let $c_i = l_1 \vee l_2 ... \vee l_{n_i}$ be a clause in the formula Σ. For any c_i s.t. $n_i \geq 1$, it is clear that the clause filter out 2^{n-n_i} of potential models.

Our first measure (*depth 0* of the approximation) estimates the number of filtered models considering clauses independently. Each clause c_i of length n_i weights $w_0(n_i) = 2^{n-n_i}$. The whole weight of the proof may then be measured as $\Sigma_{i=1}^{n} w_0(n_i)$. It is however clear that filtered models are not independent and this measure gives a very inaccurate indication of the *quality* of the current proof. It remains at trying to produce as many short clauses as possible.

At depth 1 of the approximation, we fix the granularity to literals, by maintaining the estimation of the number of filtered models for each literal. For each literal l, we may sum its weight in all clauses where it occurs by considering that a clause c_i of length $n_i \geq 1$ filter out 2^{n-n_i} of the models that contain $\neg l$ (over its 2^{n-1} potential models). If we suppose that all these filtered models are separately and equally distributed over all literals in the clause, which is a strong assumption on which our estimation is based, then the clause c_i allows literal l to filter out $w_1(n_i) = \frac{2^{n-n_i}}{n_i}$ of the models containing $\neg l$. Even through this is a refinement of depth 0, it is clear that if l occurs in the two clauses $l \wedge q$ and $l \wedge \neg q$ then this scoring scheme is particularly inappropriate. In some way, it is important to take the locality of common variables into account.

This is what approximation of depth 2 does. The granularity is here pairs of literals. For a clause c_i of length n_i, the 2^{n-n_i} filtered models are supposed as equally distributed over the $n_i.(n_i-1)/2$ pairs of literals occurring in c_i. Each pair (l_1, l_2) appearing in c_i is credited a weight of $w_2(n_i) = \frac{2^{n-1-n_i}}{n_i.(n_i-1)}$. The score of a pair of literal (l_1, l_2) is defined as the sum of its weights in all clauses and noted $S(l_1, l_2)$. The score $S(c)$ of a clause c is the sum of the scores of all the pairs of literals it contains. In the remaining sections, w and S may be used without indices for depth 2.

The deeper the approximation is, higher the cost to maintain it. We may refine our measurement by considering triplet of literals, but any explicit representation of all possible triplets of literals may be quickly unpracticable on realistic UNSAT benchmarks.

Remark 1 (Clauses scoring) *If a given clause c_i has a score $S(c_i) \simeq n_i.(n_i-1).w_2(n_i)/2$ then c_i is nearly the only one that filter the models composed by the negation of its literals. Even if such a clause is long, it should be kept in the proof. At the opposite, if $S(c_i) \gg n_i.(n_i-1).w_2(n_i)/2$ then there is a little hope that this clause is from great importance in the current proof.*

What are Greedy Moves?

We now have a scoring scheme for pairs and clauses. If we try to improve the total sum of clauses scores over Σ, then our algorithm will look for large clauses in which a lot of frequent pairs occur. This is the completely opposite goal.

Instead of trying to improve a measurement over the whole proof, we'll focus on *quadruplets* of pairs only. In order to derive $\bot$, we need a step in our proof where l and $\neg l$ are in

Σ. Because we perform 2-Saturation with Unit-Clause propagation at each step of our moves (see in the later), such a case never occurs. So, we have to find two literals l_1 and l_2 such that clauses $l_1 \vee l_2$, $\neg l_1 \vee l_2$, $l_1 \vee \neg l_2$ and $\neg l_1 \vee \neg l_2$ can be derived from Σ. In other words, we'll try to improve the scores of the four pairs built on the same variables, called *quadruplets* and noted $[x_1, x_2]$. Because we want to localise quadruplets where all pairs have a high score, we define the score $S_q([x_1, x_2])$ over a quadruplet as the sum of the squares of the scores of its pairs $S_q([x_1, x_2]) = S(l_1, l_2)^2 + S(\neg l_1, l_2)^2 + S(l_1, \neg l_2)^2 + S(\neg l_1, \neg l_2)^2$. Any move that enhance the score of one of the best scored quadruplets is thus a greedy move.

Neighbourhood

Adding any clause $c' \notin \Sigma$ to Σ such that $\Sigma \vdash c'$ may define the neighbourhood. However, the deduction mechanism must be restricted to a correct but polynomially bounded and uncomplete method. Otherwise, one may reduce GUNSAT to a single call to any complete SAT solver to check if $\Sigma \vdash \bot$ and then add it to Σ.

We chose to use the 1-Resolution mechanism, noted $\vdash^1_{\text{Res}}$, which is defined as $\Sigma \vdash^1_{\text{Res}} c$ iff $c = c_1 \otimes c_2$ where c_1 and c_2 occur in Σ. Even if this rule is very simple, we are already facing a huge neighbourhood. To illustrate this, let's take a random k-SAT formula at a given ratio r, with n variables and $r \times n$ clauses. Before the first move, each literal occurs on average $k.r.n/2$ times, and $k^2.r^2.n^2/4$ clauses may be derived from 1-Resolution on it, which give for the whole formula $n^3.k^2.r^2/4$ potential clauses. For a 700 variables 3SAT formula at threshold ($r = 4.25$), we already have over 10^{10} potential moves to rank. Random moves in this huge neighbourhood may only lead to blind search for unsatisfiability, without any hope to beat ordered resolution-based algorithms like [Dechter and Rish, 1994; Simon and del Val, 2001].

Finding the bests Greedy Moves

Any explicit exploration of the neighbourhood is impossible in practice. However, if we know in advance that we have to increase the score of one of the bests quadruplets, we may try to increase the score of any of one of its pairs.

Let $[x_1, x_2]$ be the best scored quadruplet in Σ. Increasing the score of any of its pairs (l_1, l_2) amounts to adding any new clause containing both l_1 and l_2. Short clauses will grant (l_1, l_2) a higher score, and should be preferred. However, the score of the new clause itself has to be taken into account. If we add a short clause with a high score, then this clause will probably be considered as useless in the next iteration of the algorithm. Thus, we are looking for a short clause with the lowest possible score to filter out new potential models.

When looking for this new clause with l_1 and l_2, we first try to localise the pivot variable on which we'll perform the resolution rule. If such a pivot p exists, then $S(l1, p) > 0$ and $S(\neg p, l_2) > 0$. We thus have to choose a clause c_p containing l_1 and p, and a clause $c_{\neg p}$ containing $\neg p$ and l_2. To prevent the new clause from having a high score, we only try to generate the new clause c from the two clauses having the lowest scores. Because of this restrictions, it is not always possible to produce a new clause c containing l_1 and l_2 (for instance, c may be subsumed by some clause of Σ or may simply be a tautology). If one pair score cannot be improved, then the other pairs of the same quadruplet are iteratively tried. If GUNSAT fails on all pairs of the best quadruplet, then the second quadruplet is tried and so on. This is what the call to `Add-One-Clause` does.

Removing *useless* clauses

As we have seen, clause scoring will be preferred to a simple measurement of clause length to localise useless ones. This mechanism allows us to keep large clauses in Σ, especially if it is the only one to filter out a large number of pairs. However, it is essential to keep *vital* clauses in Σ. They ensure that we are preserving its unsatisfiability. Vital clauses are initial clauses, or any clause that previously subsumed another vital clauses. We also forced the algorithm to keep binary clauses. This step is performed by the call to `Remove-One-Clause`.

Finding a random start

As described in GUNSAT skeleton, random start and general iteration of the algorithm only differ in the removing or not of clauses at each step. During the random start initialisation, no clauses are removed (except any subsumed clause) in order to fill the proof up to `MaxSize` clauses.

3.4 Other refinements

It has been essential to add three powerful mechanisms to GUNSAT in order to make it competitive to other resolution-based reasoning systems. The first but essential refinement concerns subsumptions. Before adding a new clause, we perform forward/backward subsumption detection [Zhang, 2005].

Binary clause saturation

The power of binary clause saturation has been exploited with success in [Bacchus, 2002] for preprocessing purposes. At each step of the walk, binary clauses, related to our pairs of literals, have a special treatment in the call to `2-Saturation`. Each time a new binary clause is found in Σ, all resolutions between the set of binary clauses are performed to saturate Σ with it. In order to exploit their full power, an equivalency literal search is performed: if clauses $l_1 \vee \neg l_2$ and $\neg l_1 \vee l_2$ are found in Σ, then all occurrences of l_1 are replaced by l_2 and l_1 is tagged as a new potential extended variable to be used in the latter search.

While performing the binary clause saturation, the algorithm may find new unary clauses l_1. The literal l_1 is then propagated in the whole formula by unit propagation (all clauses containing l_1 are deleted, all clauses containing $\neg l_1$ are shortened). An inconsistency may be found at this step and then returned to the main algorithm.

Extended Resolution

When the algorithm has tried to increase the score of a given pair of literals too many-times without any success, we chose to adopt a tricky solution that really pays: use extended resolution (ER) to artificially increase the score of this pair of literals.

The extended rule $e \Leftrightarrow l_1 \vee l_2$ is encoded by the three clauses $(\neg e \vee l_1 \vee l_2)$, $(e \vee \neg l_1)$ and $(e \vee \neg l_2)$. If necessary, the

Add-Extended-Variable step may add a fresh variable by adding all those three clauses to Σ. Of course, we ensured that a pair of literals can only appear in at most one extended rule. As we'll see section 4, this very simple rule gives very good results in practice.

Look Ahead techniques

The patented Stålmarck method [Stålmarck, 1994], which gives good results in practice on some structured benchmarks, uses a powerful Look Ahead technique to detect equivalencies between literals until an inconsistency is found. It uses a reformulation of the initial formula, based on a set of *triplets* ($p \Leftrightarrow q \Leftrightarrow r$ and $p \Leftrightarrow q \Rightarrow r$ only). When formulae are initially written in CNF, the power of this method may be partially captured by unit-propogation lookahead (LH) [LeBerre, 2001].

To enhance the power of GUNSAT, we added LH techniques on pairs of literals. The four possible values of pairs are iteratively propagated in Σ, searching for more, implied, unit propagations. If any literal l of Σ is set to $\perp$ in all the four tries, then LH proved that $\Sigma \vdash \neg l$, and the unary clause $\neg l$ is added to Σ.

Restarting

When GUNSAT fails to derive $\perp$ from Σ after MaxFlips steps, a restart is performed. All clauses, except binary ones and the set of *vital* clauses, are removed. To ensure that GUNSAT will explore another part of the search space, we added random number to cut ties of quadruplets having the same scores up to a given ϵ. Each restart is followed by a new random generation of quadruplets random numbers. All clauses containing at least one extended variables are deleted after each restart, including binary ones.

A particularity of our local search algorithm may be emphasised here: because of unit propagation, binary clause saturation and subsumption deletion, Σ may evolve from starts to restarts. Hopefully, Σ will only evolve to a simpler and simpler formula.

4 Experimental evaluation

Our results were obtained with a Xeon 3.4Ghz with 2GB memory. GUNSAT is a prototype written in java and a lot of optimisations are still planned (Java API LinkedList data structure are for instance used and may be greatly improved). In addition, our current implementation use an explicit representation of all pairs, which is a very costly operation. We are planning to manipulate only the Top-M best pairs of literals in our next versions.

Because of these two weak points, we selected instances with only a very restricted number of clauses and variables (*aim*, *xor* and *jnh*). Furthermore, like other local search algorithms, GUNSAT has a lot of parameters to tune and some of them may be improved in the near future. We chose to perform 50 restarts and to fix the number of flips to 6 times the initial number of clauses.

As we have previously said, our purpose is not here to compare GUNSAT to state-of-the-art SAT solvers. It is clear that conflict learning solvers like ZCHAFF [Moskewicz *et al.*, 2001] for structured instances and solvers like KCNFS [Dubois and Dequen, 2001] for random instances outperform GUNSAT. We rather want to compare it with resolution-based solvers.

We compared 4 versions of our solver: the basic one, the one with lookahead (LH) only, with extended resolution only (ER) and the one with both of them (LH+ER).

	basic		LH		ER		LH + ER	
	% S	T (F)	% S	T (F)	% S	T (F)	% S	T (F)
aim-50 (8)	12	2.14 (26)	100	1.41 (146)	60	15.14 (1749)	100	1.58 (142)
aim-100 (8)	10	36.83 (3954)	55	11.93 (923)	27	139.74 (4998)	97	49.37 (1726)
aim-200 (8)	0	-	60	63.62 (1098)	5	739.00 (9099)	85	201.29 (2009)
jnh (33)	6	0.95 (0)	57	8.48 (276)	18	68.15 (986)	62	4.21 (687)
xor (39)	0	-		-	1.5	308.83 (6932)	11	31.91 (5197)

Table 1: Results on structured instances. %S gives the percentage of solved instances (5 launches per instance). T means Time (in seconds, averaged of all successful runs), and F gives the averaged total number of flips.

		LH + ER	
V	R	% S	T (F)
50	4.25	58	60 (3880)
50	5.0	86	18 (1520)
50	6.0	97	5 (545)
60	4.25	35	126 (5785)
60	5.0	68	67 (3346)
60	6.0	92	16 (1094)
70	4.25	23	189 (6626)
70	5.0	51	187 (6193)
70	6.0	87	59 (2389)

Table 2: 3-SAT Random instances - Each category contains 100 instances, each instance is solved 50 times

4.1 Structured instances

Table 1 reports the results obtained by GUNSAT on structured problems. We do not report the original DR performances [Dechter and Rish, 1994]: over the 187 runs, only 47 finished without a memory allocation error and only 9 instances were solved (all aim-50 and 1 xor). ZRes [Simon and del Val, 2001] was able to solve all aim-50, half aim-100, but failed to solve aim-200 and jnh. It was able to solve all xor instances, because of its ability to handle large set of highly structured clauses.

It is clear that proposed refinements (LookAhead and Extended Resolution) and especially their combination provide a realistic local search solver for UNSAT problems.

4.2 Random instances

Table 2 reports results on random instances with different number of variables (from 50 to 70) and different ratio (4.25, 5 and 6). We only report results for ER+LH because all other versions of GUNSAT were only able to solve at most 1% of the instances. The solver DR cannot solve either these instances. ZRes solves only instances with 50 variables at the threshold in 250 seconds on average. [Prestwich and Lynce, 2006] also reports bad results on random instances with RANGER.

Even on hard examples for resolution based algorithms, GUNSAT LH+ER showed promising results, in comparison with resolution based solvers and RANGER, especially for large clause/variable ratios.

One may also notice the very low number of flips in all the cases, which mean that the obtained proof length is short, in comparison with ZRes experimentations on random instances.

5 Conclusion and future work

We report the first good results on local search for unsatisfiability since [Selman *et al.*, 1997] challenged the community. GUNSAT differs from classical local search algorithm in its high-cost flip mechanism. Finding and adding good clauses, while maintaining the score of all pairs, is a costly – but necessary – task to prevent a blind local search. GUNSAT learns from start to restarts and may be able to prove unsatisfiability start after restarts, by suddenly finding a derived unit clause before starting again with this new high-valuable clause. We showed that LH techniques with ER are two key points to allow local search on resolution proofs. With both mechanisms, made possible by our scoring based on pairs of literals, GUNSAT was able to defeat state-of-the-art resolution based solvers.

Those good results should be relativized by the fact that GUNSAT cannot yet challenge state of the art DPLL solvers. However, those solvers have two main drawbacks: they hardly exhibit a proof of unsatisfiability of the initial formula that can be checked by a third-party trusted algorithm and they seem to reach their limit when extended to QBF solving. Resolution-based algorithms may be the next generation of efficient QBF solving, with a short proof checking made possible. In this new and still promising area, we believe that GUNSAT may take a good place.

References

[Bacchus, 2002] F. Bacchus. Enhancing davis putnam with extended binary clause reasoning. In *proceedings of AAAI*, pages 613–619, 2002.

[Chvátal and Szemerédi, 1988] V. Chvátal and E. Szemerédi. Many hard examples for resolution. *Journal of the ACM*, 35:759–768, 1988.

[Davis and Putnam, 1960] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, pages 201–215, 1960.

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

[Dechter and Rish, 1994] R. Dechter and I. Rish. Directional resolution: the davis-putnam procedure, revisited. In *proceedings of KR*, pages 134–145, 1994.

[Dubois and Dequen, 2001] O. Dubois and G. Dequen. A backbone-search heuristic for efficient solving of hard 3–sat formulae. In *Proceedings of IJCAI*, 2001.

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

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

[Gent *et al.*, 2003] I. Gent, H. Hoos, A. Rowley, and K. Smyth. Using stochastic local search to solve quantified boolean formulae. In *Proc.of CP*, pages 348–362, 2003.

[Haken, 1985] A. Haken. The intractability of resolution. *Theorical Computer Science*, 39:297–308, 1985.

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

[LeBerre and Simon, 2006] D. LeBerre and L. Simon, editors. *Special Volume on the SAT 2005 competitions and evaluations*, volume 2. Journal on Satisfiability, Boolean Modeling and Computation, 2006.

[LeBerre, 2001] D. LeBerre. Exploiting the real power of unit propagation lookahead. In *Proceedings of SAT*, 2001.

[Moskewicz *et al.*, 2001] 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.

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

[Robinson, 1965] J. Robinson. A machine-oriented logic based on the resolution principle. *JACM*, 12:23–41, 1965.

[Selman *et al.*, 1994] B. Selman, H. Levesque, and D. Mitchell. A New Method for Solving Hard Satisfiability Problems. In *Proceedings of AAAI*, pages 440–446, 1994.

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

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

[Simon and del Val, 2001] L. Simon and A. del Val. Efficient consequence finding. In *proceedings of IJCAI*, pages 359–365, 2001.

[Stålmarck, 1994] G. Stålmarck. A system for determining propositional logic theorem by applying values and rules to triplets that are generated from a formula. US Patent 5,276,897; Canadian Patent 2,018,828; European Patent 0 403 545; Swedish Patent 467 076, 1994.

[Tseitin, 1970] G. Tseitin. On the complexity of derivation in propositional calculus. *Studies in Constructive Mathematics and Mathematical Logic*, 2:466–483, 1970.

[Urquhart, 1987] A. Urquhart. Hard examples for resolution. *JACM*, 34(1):209–219, 1987.

[Zhang, 2005] L. Zhang. On subsumption removal and on the fly cnf simplification. In *proceedings of SAT*, volume 3569 of *LNCS*, pages 482–489, 2005.