Reducing Chaos in SAT-like Search: Finding Solutions Close to a Given One

Ignasi Abío[1], Morgan Deters[2], Robert Nieuwenhuis[1], and Peter J. Stuckey[3]

[1] Technical University of Catalonia (UPC), Barcelona
[2] New York University
[3] National ICT Australia, University of Melbourne

Abstract. Motivated by our own industrial users, we attack the following challenge that is crucial in many practical planning, scheduling or timetabling applications. Assume that a solver has found a solution for a given hard problem and, due to unforeseen circumstances (e.g., rescheduling), or after an analysis by a committee, a few more constraints have to be added and the solver has to be re-run. Then it is almost always important that the new solution is "close" to the original one.
The *activity-based* variable selection heuristics used by SAT solvers make search *chaotic*, *i.e.*, extremely sensitive to the initial conditions. Therefore, re-running with just one additional clause added at the end of the input usually gives a completely different solution. We show that naive approaches for finding close solutions do not work at all, and that solving the Boolean optimization problem is far too inefficient: to find a reasonably close solution, state-of-the-art tools typically require much more time than was needed to solve the original problem.
Here we propose the first (to our knowledge) approach that obtains close solutions quickly. In fact, it typically finds the optimal (i.e., closest) solution in only 25% of the time the solver took in solving the original problem. Our approach requires no deep theoretical or conceptual innovations. Still, it is non-trivial to come up with and will certainly be valuable for researchers and practitioners facing the same problem.

1 Introduction

For many practical problems, good encodings into propositional logic exist that make them amenable to be solved with SAT. Due to techniques such as conflict-driven backjumping, lemma learning and restarts, state-of-the-art SAT solvers can in many cases efficiently solve large and hard real-world instances. For problems that have no good or compact direct encodings into propositional logic, several extensions of SAT are emerging. One of these extensions is *SAT Modulo Theories* (SMT), where atoms need not be propositional symbols, but may belong to *theories*, like, for example, linear arithmetic, as in the formula $x \leq 2 \ \wedge \ (x{+}y \geq 10 \ \vee \ 2x{+}3y \geq 30) \ \wedge \ y \leq 4$. In SMT, a SAT solver cooperates with *theory solvers* that can handle *conjunctions* of theory atoms (see, *e.g.*, [NOT06] for details). Another extension of SAT is the Lazy Clause Generation approach of [OSC09], where new propositional clauses are generated on demand each time a given constraint propagates, thus frequently reducing the number of clauses needed in comparison with a direct *a priori* SAT encoding.

SAT and SAT-like solving approaches almost universally make use of *activity-based* search heuristics, which roughly speaking, select the variables that have been involved in many recent conflicts. A drawback of activity-based heuristics is that they make the search behave *chaotically* (explaining why is out of the scope of this paper), i.e., extremely sensitive to the initial conditions, the so-called *butterfly effect*.

But in practice it is almost always important that the new solution is "close" to the original one. For example, analyzing a solution may take time and effort and include discussions with other people. If someone, inspired by the solution, suggests adding a few new constraints, it is undesirable that a new solution for the extended problem has nothing in common with what was analyzed previously. Something similar happens in the context of *rescheduling*, where a solution that was intended to be used for a period of time has to be adapted due to unforeseen circumstances: changes should be minimal since many resources (people, vehicles, machines) are already allocated according to the original solution.

In this paper, Section 2 gives a short introduction to state-of-the-art SAT solving. In Section 3 we accurately define the problem and we discuss the distance metrics, e.g., what it means for a solution to be *close*. Section 4 presents the experimental setting and the large set of real-world benchmarks used along the paper. In particular, in Section 5 we use them to experimentally demonstrate the extremely chaotic behavior of SAT Solvers, and in Section 6 to evaluate a naive attempt for finding close solutions inspired by local search methods.

Since this method does not solve the problem, in Section 7 we introduce a new approach. It combines a polarity heuristic, incremental SAT and branch-and-bound. In Section 8 we compare our method with (i) SAT-based optimization and Max-SAT solvers; (ii) modeling the problem as a 0-1 integer optimization problem and using CPLEX on it. As we shall see, approaches (i) and (ii) behave very poorly,[4] but our new approach obtains close solutions very quickly. In fact, it typically finds the optimal (i.e., closest) solution in only 25% of the time the solver takes in solving the original problem.

Finally, Section 9 gives a factor analysis of our approach: experiments reveal that all ingredients contribute. Related work is discussed and conclusions are given in Section 10.

2 State-of-the-art SAT Solvers

Let P be a fixed finite set of propositional symbols. If $p \in P$, then p and $\neg p$ are *literals* of P. The *negation* of a literal l, written $\neg l$, denotes $\neg p$ if l is p, and p if l is $\neg p$. A *clause* is a disjunction of literals $l_1 \vee \ldots \vee l_n$. A *unit clause* is a clause consisting of a single literal. A (CNF) *formula* is a conjunction of one or more clauses $C_1 \wedge \ldots \wedge C_n$. A (partial truth) *assignment* M is a set of literals such that $\{p, \neg p\} \subseteq M$ for no p. A literal l is *true* in M if $l \in M$, is *false* in

[4] An earlier (rejected) submission about this work failed to explain this adequately and to show this experimentally for approach (i). In addition, here we also consider approach (ii) and compare with more related work.

M if $\neg l \in M$, and is *undefined* in M otherwise. A literal is *defined* in M if it is either true or false in M. A clause C is true in M if at least one of its literals is true in M. It is false in M if all its literals are false in M, and it is undefined in M otherwise. A formula F is true in M, denoted $M \models F$, if all its clauses are, and then M is a *model* of F. If F has no models then it is *unsatisfiable*. If F and F' are formulas, we write $F \models F'$ if F' is true in all models of F. If C is a clause $l_1 \vee \ldots \vee l_n$, we write $\neg C$ to denote the formula $\neg l_1 \wedge \ldots \wedge \neg l_n$.

Following [NOT06], here we say that a *state* in a SAT solver is a pair of the form $M \parallel F$, where F is a finite set of clauses, and M is a (partial) assignment, where a literal l may be annotated as a *decision literal* (see below), writing it as l^{d}. A clause C is a *conflict* in a state $M \parallel F, C$ if $M \models \neg C$. A SAT solving procedure can be modeled by a set of rules over such states:

UnitPropagate :

$$M \parallel F, C \vee l \implies M\ l \parallel F, C \vee l \quad \textbf{if} \quad \begin{cases} M \models \neg C \\ l \text{ is undefined in } M \end{cases}$$

Decide :

$$M \parallel F \implies M\ l^{\mathsf{d}} \parallel F \quad \textbf{if} \quad \begin{cases} l \text{ or } \neg l \text{ occurs in a clause of } F \\ l \text{ is undefined in } M \end{cases}$$

Fail :

$$M \parallel F, C \implies \mathit{Fail} \quad \textbf{if} \quad \begin{cases} M \models \neg C \\ M \text{ contains no decision literals} \end{cases}$$

Backjump :

$$M\ l^{\mathsf{d}}\ N \parallel F, C \implies M\ l' \parallel F, C \quad \textbf{if} \quad \begin{cases} M\ l^{\mathsf{d}}\ N \models \neg C, \text{ and there is} \\ \text{some clause } C' \vee l' \text{ such that:} \\ \quad F, C \models C' \vee l' \text{ and } M \models \neg C', \\ \quad l' \text{ is undefined in } M, \text{ and} \\ \quad l' \text{ or } \neg l' \text{ occurs in } F \text{ or in } M\ l^{\mathsf{d}}\ N \end{cases}$$

Learn :

$$M \parallel F \implies M \parallel F, C \quad \textbf{if} \quad \begin{cases} \text{each atom of } C \text{ occurs in } F \text{ or in } M \\ F \models C \end{cases}$$

Forget :

$$M \parallel F, C \implies M \parallel F \quad \textbf{if} \quad \{ F \models C$$

For deciding the satisfiability of an input formula F, one can generate an arbitrary derivation $\emptyset \parallel F \implies \ldots \implies S_n$, where S_n is a final state (no rule applies). Under simple conditions, this always terminates. Moreover, for every derivation ending in a final state S_n, *(i)* F is unsatisfiable iff S_n is *Fail*, and *(ii)* if S_n is of the form $M \parallel F$ then M is a model of F (cf.[NOT06] for details).

In the current state-of-the-art SAT solvers such as MiniSAT [ES04], the *variable selection heuristics* are *activity-based*: roughly, Decide is done on variables with *many* occurrences in *recent* conflicts. In this paper we also consider the choice of *polarity* for Decide, *i.e.*, whether the variable is set to true or to false.

We say that a state M is at *decision level* n if in M there are n decision literals. In Backjump, $C' \vee l'$ is called the *backjump clause*. This clause is a logical consequence to which UnitPropagate would have applied at a lower decision

level, and Backjump does precisely this, after reverting to that decision level. In practice, the backjump clause is computed in a *conflict analysis* process, which is beyond the scope of this paper.

The Learn rule corresponds to adding *lemmas* (clauses that are logical consequences) such as the backjump clause. Since a lemma is aimed at preventing future similar conflicts, when these conflicts are not very likely to be found again the lemma can be removed by the Forget rule. In practice, a lemma is removed when its *activity* drops below a certain threshold; the activity can be, *e.g.*, the number of times it becomes a unit or a conflicting clause [GN02].

3 Problem definition

Assume we have found a solution Sol to a problem defined by a formula (a set of clauses) F and we are given a small set of additional clauses δ. We wish to find a solution Sol' that is *close* to Sol for the clause set $F \cup \delta$.

One way for defining solutions' proximity is by considering their Hamming distance (the number of variables which take a different value). As many problems have some *hidden* auxiliary variables in their SAT encoding F, it is frequently useful to consider only the *visible* (i. e. non hidden) variables for the distance definition.

Certain applications can require slightly more involved cost functions instead of just Hamming distance. For example, a single property of the solution, seen by the user, may depend on *combinations* of visible variables. For example, in the sports scheduling problems we will use later, a property like a match may depend on a variable m_{ijr} saying that these two teams i and j meet on round r, and another two h_{ir} and h_{jr} saying whether team i and j plays at home on round r. A more accurate cost function to capture "nearness to the existing solution" in this case would count a distance of 1 if either of m_{ijr} or h_{ir} differ from their previous values, but not count 2 if both differ.

However, in this paper we have only considered Hamming distance cost functions for simplicity in the computations. In the majority of the practical cases, a close solution for some distance is also a close solution for the Hamming distance (see the previous example).

4 Benchmarks

We have considered 40 instances of real-world benchmarks coming from five different families. Each instance consists of a different SAT formula F, the first solution Sol, and a number of required additional constraints δ. The first four families are for scheduling a double round-robin tournament among N (16, 20 or 24) teams:

`r16`: 10 instances with about 3000 variables and around 55000 clauses each;
`r20`: 10 instances with around 5000 variables and 180000 clauses each;
`R20`: 10 instances with around 5000 variables, 140000 clauses each;

`r24`: 4 instances with around 9000 variables, 270000 clauses each.
All teams meet each other once in the first $N-1$ weeks and again in the second $N-1$ weeks, with exactly one match per team each week. A given pair of teams must play at the home of one team in one half, and at the home of the other in the other half, and such matches must be spaced at least a certain minimal number of weeks apart. Additional constraints include, e.g., that no team ever plays at home (or away) three times in a row, other (public order, sportive, TV revenues) constraints, blocking given matches on given days, etc. Instances are rather different among each other, but most of them have around 10% hidden variables. The `R20` instances are also different in that their δs contain more constraints and hence the closest solution is usually not as close (see below).

The fifth family of benchmarks has six problems `tt0 - tt5` coming from real-world hard curriculum-based course timetabling problems, from the International Timetabling Competition, see the Barcelogic results on formulation 2 at `http://tabu.diegm.uniud.it/ctt`. These problems are very different from the `r` ones. Their numbers of (visible) variables and clauses are:

instances	variables	visible variables	clauses
`tt0`	12537	1500	71919
`tt1`	137688	6314	667470
`tt2`	60968	3150	305601
`tt3`	556569	9810	3372803
`tt4`	125029	4494	1001737
`tt5`	124330	3381	612475

For each instance, we consider Hamming distance on the visible variables as the cost function. All experiments were performed on a 2.66MHz Xeon.

5 Chaotic behavior of SAT

In this section we analyze what happens when simply re-executing the Solver with the new input $F \cup \delta$. Table 1 contains results on all 40 instances.

Here *Time original* denotes the time (in seconds) spent to compute the original solution *Sol*, *Time re-execution* denotes the time spent in the computation of Sol'. d_{opt} denotes the minimal Hamming distance from the original solution *Sol* to any solution of $F \cup \delta$. *Time ratio* is defined by the ratio between the re-execution time and the original time.

The *quality* of a solution Sol' at distance d of *Sol* is a real number between 0 and 1 defined by d_{opt}/d. For example, if d_{opt} is 10, then a solution at distance 50 has quality 0.2.

These experiments show the chaotic behavior of SAT Solvers: re-running the same solver with the same set of clauses except one or two added at the end of the input file causes the solver to perform a completely different search, giving a very different execution in terms of distance of the solutions and also

Instance	Time original	d_{opt}	Quality	Time re-execution	Time ratio
r16-0	0.88	12	0.03	0.93	1.06
r16-1	1.58	14	0.04	1.27	0.80
r16-2	1.66	8	0.02	0.74	0.45
r16-3	0.97	8	0.02	1.63	1.68
r16-4	3.56	64	0.14	7.09	1.99
r16-5	0.03	12	0.22	0.04	1.33
r16-6	0.02	14	0.03	0.05	2.50
r16-7	0.4	18	0.04	0.69	1.72
r16-8	3.55	8	0.02	1.27	0.36
r16-9	1.39	12	0.03	0.61	0.44
r20-0	12.23	24	0.04	12.37	1.01
r20-1	59.6	8	0.01	20.00	0.34
r20-2	9.47	12	0.02	9.65	1.02
r20-3	12.82	14	0.03	2.83	0.22
r20-4	20.15	18	0.19	20.03	0.99
r20-5	20.48	16	0.02	8.82	0.43
r20-6	8.81	18	0.04	2.09	0.24
r20-7	10.88	20	0.03	13.46	1.24
r20-8	13.52	16	0.04	8.95	0.66
r20-9	7.04	12	0.03	12.39	1.76
R20-0	1.77	8	0.02	3.56	2.01
R20-1	2.37	88	0.17	6.30	2.66
R20-2	6.69	96	0.19	9.53	1.42
R20-3	9.46	8	0.01	5.30	0.56
R20-4	5.4	136	0.25	1.14	0.21
R20-5	1.14	1	0.00	7.04	6.18
R20-6	7.71	104	0.19	4.95	0.64
R20-7	5.45	26	0.05	0.62	0.11
R20-8	0.61	82	0.16	7.03	11.52
R20-9	7.49	94	0.16	1.78	0.24
r24-0	227.97	42	0.04	143.51	0.63
r24-1	124.28	58	0.05	315.14	2.54
r24-2	277.49	14	0.01	226.80	0.82
r24-3	200.53	8	0.01	416.14	2.08
tt-0	1.62	10	0.03	0.36	0.22
tt-1	0.96	10	0.07	0.93	0.97
tt-2	0.38	6	0.04	0.28	0.74
tt-3	16.3	8	0.01	14.20	0.87
tt-4	27.42	26	0.04	16.17	0.59
tt-5	1.75	8	0.02	1.73	0.99

Table 1. Results of re-execution.

in computation time. In particular, qualities are typically below 0.1, that is, ten times more distant than the optimal solution.

6 Trying a local search-like solution

In local search techniques, to find close solutions one usually resumes the search at the point where the original solution was found with the hope that another solution is found in the nearby neighborhood. Therefore, at first sight, mimicking local search might seem a good option for overcoming the chaotic behavior of SAT.

More specifically, we want to re-execute the solver in the region of the search tree where the original solution was found. A simple way of implementing this idea is by changing the *variable selection heuristics* as follows. We remember the ordered sequence of decision literals of the original solution, and when the solver is re-launched with the new constraints, it always decides on the first undefined literal of the sequence, with the same polarity, until the first conflict occurs. After that, we fall back to the standard decision heuristic. Note that this will always find the same solution Sol if Sol is also a solution of $F \cup \delta$.

Unfortunately, the results do not improve significantly upon re-running from scratch as described in the previous section. Table 2 contains the results of this method. We have obtained similar results with some variations of this method (keeping the lemmas of the original execution as in the next section, keeping this heuristic, or a combination of both).

7 Our Barcelogic approach

As we have seen in the previous sections, the naive approaches are not effective for solving this problem in practice. The good news is that an adequate combination of three quite well-known ingredients does obtain close solutions very quickly.

The first ingredient is a polarity selection heuristic: the SAT solver uses its standard heuristic for picking the next variable to decide upon, but for visible variables it sets this variable's polarity as in the original solution Sol (other optimization tools do this too: first try those values that minimize the cost function; it is also related to, but different from, *phase saving* [PD10]).

Second, a branch-and-bound wrapper is placed around the standard SAT loop. Each time the cost of the best solution discovered so far is exceeded by the current partial assignment, due to literals $l_1 \dots l_n$ (on visible variables) that disagree with Sol, a backjump is forced from a conflict analysis on an "explanation" $\neg l_1 \vee \dots \vee \neg l_n$ of why the cost is currently too high. In particular, this is done each time a better model is found, in order to find, from then on, only lower-cost models. Here this explanation clause need not be learned. The backjump clause itself is learned as usual. Eventually this process terminates by discovering unsatisfiability—that there is no "better" solution to the best already found. As

Instance	Time original	d_{opt}	Quality	Time re-execution	Time ratio
r16-0	0.88	12	0.03	0.80	0.91
r16-1	1.58	14	0.03	1.87	1.18
r16-2	1.66	8	0.02	0.66	0.40
r16-3	0.97	8	0.02	2.81	2.90
r16-4	3.56	64	0.13	3.82	1.07
r16-5	0.03	12	0.03	0.08	2.67
r16-6	0.02	14	0.03	0.00	0.00
r16-7	0.4	18	0.04	0.18	0.45
r16-8	3.55	8	0.02	0.85	0.24
r16-9	1.39	12	0.03	1.27	0.91
r20-0	12.23	24	0.04	9.50	0.78
r20-1	59.6	8	0.31	0.03	0.00
r20-2	9.47	12	0.55	0.03	0.00
r20-3	12.82	14	0.03	6.64	0.52
r20-4	20.15	18	0.33	0.03	0.00
r20-5	20.48	16	0.03	21.12	1.03
r20-6	8.81	18	0.03	17.50	1.99
r20-7	10.88	20	0.03	6.69	0.61
r20-8	13.52	16	0.03	2.15	0.16
r20-9	7.04	12	0.02	6.96	0.99
R20-0	1.77	8	0.02	2.43	1.37
R20-1	2.37	88	0.16	5.20	2.19
R20-2	6.69	96	0.15	2.26	0.34
R20-3	9.46	8	0.02	4.77	0.50
R20-4	5.4	136	0.26	6.34	1.17
R20-5	1.14	1	0.00	5.84	5.12
R20-6	7.71	104	0.20	6.18	0.80
R20-7	5.45	26	0.05	11.27	2.07
R20-8	0.61	82	0.16	4.94	8.10
R20-9	7.49	94	0.17	2.82	0.38
r24-0	227.97	42	0.04	134.14	0.59
r24-1	124.28	58	0.05	3574.00	28.76
r24-2	277.49	14	0.01	157.08	0.57
r24-3	200.53	8	0.01	296.43	1.48
tt-0	1.62	10	0.03	0.21	0.13
tt-1	0.96	10	0.29	0.29	0.30
tt-2	0.38	6	0.60	0.13	0.34
tt-3	16.3	8	0.01	18.13	1.11
tt-4	27.42	26	0.04	11.88	0.43
tt-5	1.75	8	0.01	2.36	1.35

Table 2. Results of a local-search-like approach.

is well-known, it may require far more time to prove optimality than it does to find an optimal solution.[5] However, good solutions can often be found in a short time. See, e.g., [MMS04,LNORC09,LNORC11] and references of these for many more details and an abstract framework for Boolean optimization.

Third, the lemmas the SAT Solver generated when finding the original solution are added; this is sound since there are only *additional* constraints, no removed ones; this latter idea is also used in the context of incremental SAT solving for, e.g., verification applications.

8 Experimental comparison with Cplex and other tools

In this section we compare experimentally our approach with other tools. We first encoded $F \cup \delta$ together with the cost function as a pseudo-Boolean (0-1 Integer Programming) optimization problem and tried the state-of-the-art pseudo-Boolean solver *Bsolo* [MMS04] and the well-known commercial CPLEX solver.

We also tried several state-of-the-art Max-SAT solvers. MiniMaxSAT [HLO08] found close solutions only in a few cases. The unsatisfiable-core-based MaxSAT solvers *msuncore* [MSP09] and *PM2*[ABL09] were not competitive either, among other reasons because unsat-core-based solvers find no solution before the optimal one. We do not report here on these MaxSAT solvers' results: they were always much worse than the listed ones.

We also tried Barcelogic omitting its ingredients one by one, i.e., without keeping the lemmas from the first run or without the modified polarity heuristic. The results are described in the next section. Bsolo and CPLEX results are without the lemmas: the number of lemmas was much bigger than the number of original constraints and these solvers perform much worse if we add them.

The results are given in Table 3.

Solution quality: As before, the table lists *solution qualities* as real numbers between 0 and 1: d_{opt} denotes the minimal Hamming distance from the original solution *Sol* to any solution of $F \cup \delta$ and again we say that a solution Sol' at distance d of *Sol* has *quality* d_{opt}/d.

Entries in the table: The table gives results on all 40 instances for Barcelogic, Bsolo and CPLEX. For each instance, column 2 lists the time T the (Barcelogic) SAT solver took to compute the initial solution *Sol*. The third column indicates the cost of the optimal solution, d_{opt}. For each approach, the table lists the quality of the solution found after 25% of T, after 50% of T, etc., up to 800% of T. Moreover, the two average rows show the average of, respectively, the first 20 problems and the 20 other (harder) ones. The two plots of figure 1 represent graphically these averages. They also give some intuition about how the approaches scale.

[5] In fact, for some of the benchmarks in this paper *proving* optimality took days of CPU time.

	Time	d_{opt}	Barcelogic						Bsolo						Cplex					
			25	50	100	200	400	800	25	50	100	200	400	800	25	50	100	200	400	800
r16-0	0.88	12	1	1	1	1	1	1	0	0	.67	.67	1	1	0	0	0	1	1	1
r16-1	1.58	14	1	1	1	1	1	1	0	0	1	1	1	1	0	0	0	0	0	1
r16-2	1.66	8	1	1	1	1	1	1	0	1	1	1	1	1	0	1	1	1	1	1
r16-3	0.97	8	1	1	1	1	1	1	0	0	.67	1	1	1	0	0	1	1	1	1
r16-4	3.56	64	.86	.86	.94	1	1	1	.67	.67	.67	.67	.67	.67	0	0	0	0	0	0
r16-5	0.03	12	0	0	.50	.60	1	1	0	0	0	0	0	0	0	0	0	0	0	0
r16-6	0.02	14	0	0	0	0	.12	.64	0	0	0	0	0	0	0	0	0	0	0	0
r16-7	0.4	18	.82	.82	.82	.82	1	1	0	0	0	.36	.36	.36	0	0	0	0	0	0
r16-8	3.55	8	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
r16-9	1.39	12	1	1	1	1	1	1	0	.60	.60	.60	.60	1	0	0	1	1	1	1
r20-0	12.23	24	1	1	1	1	1	1	.34	.34	.34	.50	1	1	0	0	0	0	0	1
r20-1	59.6	8	1	1	1	1	1	1	.67	1	1	1	1	1	1	1	1	1	1	1
r20-2	9.47	12	1	1	1	1	1	1	.27	.38	.38	.60	.60	.75	0	1	1	1	1	1
r20-3	12.82	14	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1
r20-4	20.15	18	1	1	1	1	1	1	.41	.41	.41	.43	.43	.64	0	0	.38	.38	1	1
r20-5	20.48	16	1	1	1	1	1	1	.57	.57	.57	.89	.89	1	0	0	0	0	.24	1
r20-6	8.81	18	1	1	1	1	1	1	.30	.82	.82	.82	.82	1	0	0	0	.90	.90	1
r20-7	10.88	20	1	1	1	1	1	1	.83	.83	.83	.91	1	1	0	0	.32	.32	.32	1
r20-8	13.52	16	1	1	1	1	1	1	.35	.35	.35	.35	.35	1	0	0	0	0	1	1
r20-9	7.04	12	1	1	1	1	1	1	0	.22	.25	.55	.60	.86	0	1	1	1	1	1
Av.	-	-	**.88**	**.88**	**.91**	**.92**	**.96**	**.98**	**.32**	**.46**	**.58**	**.67**	**.72**	**.81**	**.10**	**.30**	**.43**	**.53**	**.62**	**.80**
R20-0	1.77	8	1	1	1	1	1	1	0	0	1	1	1	1	0	0	1	1	1	1
R20-1	2.37	88	.57	.57	.57	.57	.72	.75	0	0	0	0	.66	.66	0	0	0	0	0	0
R20-2	6.69	96	.74	.74	.74	.80	.84	.89	0	.53	.53	.55	.55	.70	0	0	0	0	0	0
R20-3	9.46	8	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
R20-4	5.4	136	.65	.65	.86	.86	.91	.97	0	0	0	0	0	0	0	0	0	0	0	0
R20-5	1.14	1	1	1	1	1	1	1	0	0	0	1	1	1	0	0	0	1	1	1
R20-6	7.71	104	.80	.80	.88	.88	.88	.88	0	0	0	.42	.57	.57	0	0	0	0	0	0
R20-7	5.45	26	.93	.93	1	1	1	1	.68	.68	.68	.68	.68	.68	0	1	1	1	1	1
R20-8	0.61	82	.84	.84	.84	.85	.98	.98	0	0	0	0	.60	.60	0	0	0	0	0	0
R20-9	7.49	94	.64	.77	.77	.84	.90	.90	0	.43	.59	.59	.59	.59	0	0	0	0	0	0
r24-0	227.97	42	1	1	1	1	1	1	0	0	0	0	.57	.57	0	0	0	0	0	0
r24-1	124.28	58	.58	.58	.58	.74	.74	.74	0	.42	.42	.42	.42	.42	0	0	0	0	0	0
r24-2	277.49	14	1	1	1	1	1	1	.37	.37	.37	.37	.37	.37	0	1	1	1	1	1
r24-3	200.53	8	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
tt-0	1.62	10	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	1
tt-1	0.96	10	0	.36	.36	.36	.36	.36	0	0	0	0	0	0	0	0	0	0	0	0
tt-2	0.38	6	0	.60	.60	.60	.75	.75	0	0	0	0	0	0	0	0	0	0	0	0
tt-3	16.3	8	.57	.57	.57	.57	.67	.67	0	0	0	0	0	0	0	0	0	0	0	0
tt-4	27.42	26	.10	.10	.50	.50	.50	.50	0	0	0	0	0	0	0	0	0	0	0	0
tt-5	1.75	8	0	0	0	0	.13	.14	0	0	0	0	0	0	0	0	0	0	0	0
Av.	-	-	**.67**	**.73**	**.76**	**.78**	**.82**	**.83**	**.15**	**.22**	**.28**	**.35**	**.45**	**.46**	**.10**	**.20**	**.25**	**.30**	**.30**	**.35**

Table 3. Comparative results of the three most competitive approaches: Barcelogic, Bsolo and CPLEX.

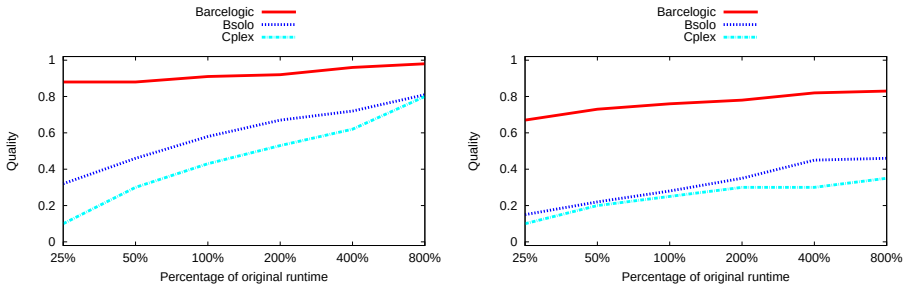

Fig. 1. Average quality of the different approaches on the first 20 problems (left) and the second 20 harder ones (right).

9 Factor analysis of the Barcelogic approach

In this section we evaluate separately the different ingredients used in our approach. More specifically, we show the experimental results of our solver with just a Branch and Bound ("B&B" in the table; first column), adding the lemmas ("B&B + lemmas"; second column), with the modified polarity heuristic ("B&B + polarity"; third column) and finally "B&B + All" (fourth column). The results are given in Table 4. As in the previous section, the table shows the quality of the solution found after 25%, 50%, etc. of the time spent in solving the original problem.

Clearly, the polarity decision heuristic hugely improves the method. On the other hand, keeping the lemmas helps significantly for the hard problems, while on the easier ones the overhead of reading the additional clauses frequently does not pay off.

Again, the two plots of figure 2 represent graphically the results of the table for average solution qualities of, respectively, the first 20 instances, and the other much harder 20 ones.

10 Related work and conclusions

We have studied, from a practical point of view, the problem of, given a SAT formula F with a model Sol, and a small set of additional clauses δ, finding a model of $F \cup \delta$ that is *close* to Sol.

Similar problems were studied before in a more theoretical (complexity) setting. [HHOW05] examine the problem of finding a set of diverse or similar solutions for a single problem using constraint programming. Their MostClose question is very similar to the problem we examine looking for the closest solution to an existing solution, but both solutions are for the same problem. They outline two approaches: a reformulation approach that at least doubles the size of the problem, and a more efficient heuristic approach which is simply a branch

	Basic B&B						B&B + lemmas						B&B + polarity						B&B + All					
	25	50	100	200	400	800	25	50	100	200	400	800	25	50	100	200	400	800	25	50	100	200	400	800
r16-0	0	0	0	.04	.04	.04	.03	.03	.03	.04	.04	.04	1	1	1	1	1	1	1	1	1	1	1	1
r16-1	0	0	.04	.04	.04	.04	0	0	.04	.04	.04	.04	1	1	1	1	1	1	1	1	1	1	1	1
r16-2	0	.02	.02	.02	.02	.02	.02	.02	.02	.02	.02	.02	1	1	1	1	1	1	1	1	1	1	1	1
r16-3	0	0	0	.03	.03	.03	0	0	.02	.02	.02	.02	1	1	1	1	1	1	1	1	1	1	1	1
r16-4	0	0	0	.14	.16	.16	0	0	.14	.14	.15	.16	.80	.80	.82	.82	.86	.91	.86	.86	.94	1	1	1
r16-5	0	0	.38	.38	.38	.38	0	0	0	.03	.03	.03	0	0	.24	1	1	1	0	0	.50	.60	1	1
r16-6	0	0	0	.03	.03	.04	0	0	0	0	0	.03	0	.12	1	1	1	1	0	0	0	0	.12	.64
r16-7	0	0	0	.04	.05	.05	0	0	0	.05	.05	.05	.69	.82	.90	.90	1	1	.82	.82	.82	.82	1	1
r16-8	0	.02	.02	.02	.02	.02	0	.02	.02	.02	.02	.02	1	1	1	1	1	1	1	1	1	1	1	1
r16-9	0	.04	.04	.04	.04	.04	0	0	0	.03	.03	.03	1	1	1	1	1	1	1	1	1	1	1	1
r20-0	0	0	0	.05	.05	.06	.04	.04	.05	.05	.05	.05	1	1	1	1	1	1	1	1	1	1	1	1
r20-1	0	.01	.01	.01	.01	.02	.01	.01	.01	.01	.01	.01	1	1	1	1	1	1	1	1	1	1	1	1
r20-2	0	0	0	.02	.02	.02	0	0	.02	.02	.02	.02	1	1	1	1	1	1	1	1	1	1	1	1
r20-3	.03	.03	.03	.03	.03	.03	0	0	.02	.02	.03	.03	1	1	1	1	1	1	1	1	1	1	1	1
r20-4	0	0	0	.20	.20	.20	0	0	.04	.04	.04	.04	1	1	1	1	1	1	1	1	1	1	1	1
r20-5	0	.02	.02	.03	.03	.03	0	.03	.03	.03	.03	.03	1	1	1	1	1	1	1	1	1	1	1	1
r20-6	.04	.04	.04	.04	.04	.04	.04	.04	.04	.04	.04	.04	1	1	1	1	1	1	1	1	1	1	1	1
r20-7	0	0	0	.03	.03	.05	.06	.06	.06	.06	.06	.06	.91	.91	1	1	1	1	1	1	1	1	1	1
r20-8	0	0	.04	.04	.04	.04	0	.03	.03	.03	.03	.03	1	1	1	1	1	1	1	1	1	1	1	1
r20-9	0	0	0	.03	.03	.03	0	0	.02	.02	.02	.02	1	1	1	1	1	1	1	1	1	1	1	1
Av.	**0**	**.01**	**.03**	**.06**	**.06**	**.07**	**.01**	**.01**	**.03**	**.04**	**.04**	**.04**	**.87**	**.88**	**.95**	**.99**	**.99**	**1**	**.88**	**.88**	**.91**	**.92**	**.96**	**.98**
R20-0	0	0	0	0	.02	.02	0	0	0	.02	.02	.02	1	1	1	1	1	1	1	1	1	1	1	1
R20-1	0	0	0	0	.18	.18	0	0	0	.15	.15	.15	0	0	.64	.64	.64	.71	.57	.57	.57	.57	.72	.75
R20-2	0	0	0	.19	.19	.20	0	0	.22	.25	.25	.25	.52	.69	.73	.75	.86	.91	.74	.74	.74	.80	.84	.89
R20-3	0	0	.01	.02	.02	.02	0	.02	.02	.02	.02	.02	1	1	1	1	1	1	1	1	1	1	1	1
R20-4	.25	.25	.25	.25	.26	.26	.23	.25	.25	.25	.25	.27	0	.44	.77	.77	.79	.85	.65	.65	.86	.86	.91	.97
R20-5	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
R20-6	0	0	.20	.20	.26	.26	0	.16	.16	.23	.23	.23	.81	.81	.87	.87	.90	.91	.80	.80	.88	.88	.88	.88
R20-7	.05	.05	.06	.06	.06	.06	0	.05	.05	.05	.05	.06	.93	.93	1	1	1	1	.93	.93	1	1	1	1
R20-8	0	0	0	0	0	0	0	0	0	0	.13	.13	.50	.50	.50	.59	.72	.93	.84	.84	.84	.85	.98	.98
R20-9	0	.16	.20	.20	.20	.20	.18	.21	.21	.21	.24	.24	.71	.81	.85	.85	.89	.89	.64	.77	.77	.84	.90	.90
r24-0	0	0	.04	.04	.04	.04	0	0	0	.04	.04	.04	1	1	1	1	1	1	1	1	1	1	1	1
r24-1	0	0	0	0	.05	.05	0	.05	.05	.06	.06	.06	.67	.67	.76	.76	.76	.76	.58	.58	.58	.74	.74	.74
r24-2	0	0	.01	.01	.01	.01	0	.01	.01	.01	.01	.01	1	1	1	1	1	1	1	1	1	1	1	1
r24-3	0	0	0	0	.01	.01	.01	.01	.01	.01	.01	.01	1	1	1	1	1	1	1	1	1	1	1	1
tt-0	.03	.03	.03	.03	.04	.04	0	0	0	0	.03	.03	.05	.14	.14	.19	.25	.50	1	1	1	1	1	1
tt-1	0	0	.07	.07	.07	.07	0	0	.04	.04	.04	.04	0	.36	.36	.36	.36	.36	0	.36	.36	.36	.36	.36
tt-2	0	0	.04	.04	.04	.04	0	0	0	.02	.02	.02	0	.60	.60	.60	.60	1	0	.60	.60	.60	.75	.75
tt-3	0	0	.01	.01	.01	.01	0	0	0	.01	.01	.01	0	0	.01	.04	.11	.29	.57	.57	.57	.57	.67	.67
tt-4	0	0	.04	.04	.04	.04	0	.04	.04	.04	.04	.04	0	0	.06	.41	.41	.41	.10	.10	.50	.50	.50	.50
tt-5	0	0	0	.02	.02	.02	0	0	.02	.02	.02	.02	0	0	0	0	.14	.14	0	0	0	0	.13	.14
Av.	**.02**	**.02**	**.05**	**.06**	**.08**	**.08**	**.02**	**.04**	**.05**	**.07**	**.08**	**.08**	**.51**	**.60**	**.66**	**.69**	**.72**	**.78**	**.67**	**.73**	**.76**	**.78**	**.82**	**.83**

Table 4. Results of the factor analysis.

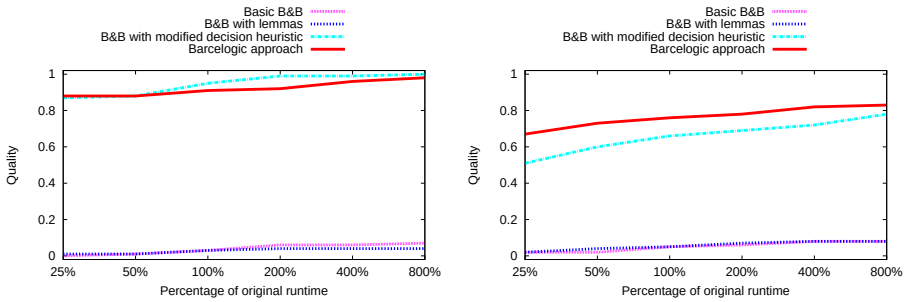

Fig. 2. Average quality of the factor analysis on the first 20 problems (left) and the second 20 harder ones (right).

and bound search. Our results show that this by itself is not enough in the SAT context. Distance-SAT [BM06] explores the decision problem, given a formula G and an *arbitrary* partial interpretation I, is there a model of G that disagrees with I on at most k variables? [BM06] tries on random and handcrafted problems two algorithms based on the classical Davis/Logemann/Loveland (DLL) procedure [DLL62], but a translation into CNF is reported to work better. For our case, where deciding SAT for G is already hard, such a translation is rather hopeless. One clearly needs to exploit that in our problem I is a model of a *known* subformula of G that is almost the same as G.

Indeed, our experiments reveal that, while state-of-the-art Boolean optimization solvers behave poorly, our Barcelogic approach behaves very well, frequently finding the optimal (i.e., closest) solution in only 25% of the time the SAT solver took in solving the original problem.

Acknowledgements NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council.

Abío and Nieuwenhuis are partially supported by Spanish Min. of Educ. and Science through the LogicTools-2 project (TIN2007-68093-C02-01). Abío is also partially supported by FPU grant.

References

[ABL09] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. Solving (weighted) partial maxsat through satisfiability testing. In *Int. Conf. Theory and Appl. of Satisfiability Testing (SAT), LNCS 4501*, pp 427–440, 2009.

[BM06] Olivier Bailleux and Pierre Marquis. Some computational aspects of distance-sat. *J. Autom. Reasoning*, 37(4):231–260, 2006.

[DLL62] M. Davis, G. Logemann, and D. Loveland. A Machine Program for Theorem-Proving. *Comm. of the ACM, CACM*, 5(7):394–397, 1962.

[ES04] N. Eén and N. Sörensson. An Extensible SAT-solver. In *6th Int. Conf. on Theory and Applications of Satisfiability Testing, SAT'03*, LNCS 2919, pages 502–518. Springer, 2004.

[GN02] E. Goldberg and Y. Novikov. BerkMin: A Fast and Robust SAT-Solver. In *2002 Conference on Design, Automation, and Test in Europe, DATE'02*, pages 142–149. IEEE Computer Society, 2002.

[HHOW05] Emmanuel Hebrard, Brahim Hnich, Barry O'Sullivan, and Toby Walsh. Finding diverse and similar solutions in constraint programming. In *20th National Conf. on Artificial Intelligence (AAAI)*, pages 372–377, 2005.

[HLO08] Federico Heras, Javier Larrosa, Albert Oliveras. MiniMaxSAT: An efficient Weighted Max-SAT Solver. *J. Artificial Intell. Research*, 31:1–32, 2008.

[LNORC09] Javier Larrosa, Robert Nieuwenhuis, Albert Oliveras, Enric Rodríguez-Carbonell. Branch and bound for boolean optimization and the generation of optimality certificates. In *12th Int. Conf. Theory and Applications of Satisfiability Testing, SAT'09*, LNCS 5584, pp 453–466, 2009.

[LNORC11] Javier Larrosa, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. A framework for certified boolean branch-and-bound optimization. *J. Autom. Reasoning*, 46(1):81–102, 2011.

[MMS04] Vasco Manquinho and João Marques-Silva. Satisfiability-based algorithms for boolean optimization. *Ann. Math. Artif. Intell.*, 40(3-4):353–372, 2004.

[MSP09] Vasco Manquinho, João Marques Silva, and Jordi Planes. Algorithms for weighted boolean optimization. In *Int. Conf. Theory and Applications of Satisfiability Testing (SAT), LNCS 4501*, pages 495–508, 2009.

[NOT06] R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Solving SAT and SAT Modulo Theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). *Journal of the ACM, JACM*, 53(6):937–977, 2006.

[OSC09] O. Ohrimenko, P.J. Stuckey, and M. Codish. Propagation via lazy clause generation. *Constraints*, 14(3):357–391, 2009.

[PD10] Knot Pipatsrisawat and Adnan Darwiche. On modern clause-learning satisfiability solvers. *J. Autom. Reason.*, 44(3):277–301, 2010.