Relaxed DPLL Search for MaxSAT

Lukas Kroc Ashish Sabharwal Bart Selman

Department of Computer Science
Cornell University, Ithaca NY 14853-7501, U.S.A.
{kroc,sabhar,selman}@cs.cornell.edu*

Abstract. We propose a new incomplete algorithm for the Maximum Satisfiability (MaxSAT) problem on unweighted Boolean formulas, focused specifically on instances for which proving unsatisfiability is already computationally difficult. For such instances, our approach is often able to identify a small number of what we call "bottleneck" constraints, in time comparable to the time it takes to prove unsatisfiability. These bottleneck constraints can have useful semantic content. Our algorithm uses a relaxation of the standard backtrack search for satisfiability testing (SAT) as a guiding heuristic, followed by a low-noise local search when needed. This allows us to heuristically exploit the power of unit propagation and clause learning. On a test suite consisting of all unsatisfiable industrial instances from SAT Race 2008, our solver, `RelaxedMinisat`, is the only (MaxSAT) solver capable of identifying a *single* bottleneck constraint in all but one instance.

1 Introduction

In recent years, we have seen tremendous progress in the area of Boolean Satisfiability (SAT) solvers. Current solvers can handle instances with over a million variables and millions of clauses. These advances have led to an ever growing range of applications, particularly in hardware and software verification, and planning. In fact, the technology has matured from being a largely academic endeavor to an area of research with strong academic and industrial participation. The current best SAT solvers for handling "structured" instances are based on Davis-Putnam-Logemann-Loveland (DPLL) style complete search.

Determining whether a Boolean formula is satisfiable or not is a special case of the maximum satisfiability (MaxSAT) problem, where the goal is to find an assignment that satisfies as many clauses as possible. Even though MaxSAT is a natural generalization of SAT, and thus closely related, progress has been much slower on the MaxSAT problem. There is a good explanation as to why this is the case. Two of the key components behind the rapid progress for DPLL-based SAT solvers are highly effective unit propagation and clause learning.

* This research was supported by IISI, Cornell University (AFOSR grant FA9550-04-1-0151), NSF Expeditions in Computing award for Computational Sustainability (0832782), NSF IIS award (0514429), and NSF EMT award (0829861). Part of this work was done while the second author was visiting McGill University.

Both techniques in a sense focus on avoiding local inconsistencies: when a unit clause occurs in a formula, one should immediately assign the appropriate truth value to the variable so that it satisfies the clause, and when a branch reaches a contradiction, a no-good clause can be derived which captures the cause of the local inconsistency. In a MaxSAT setting, these strategies can be quite counter-productive and inaccurate. For example, for an unsatisfiable instance, the best assignment, i.e., one satisfying the most clauses, may be the one that violates several unit clauses. Also, when a contradiction is reached, the best solution may be to violate one of the clauses that led to the contradiction rather than adding a no-good clause which effectively steers the search away from the contradiction. So, neither unit propagation nor clause learning appear directly suitable for a MaxSAT solver.[1] Unfortunately, taking such mechanisms out of DPLL search dramatically reduces its effectiveness. This is confirmed when one considers the performance of exact solvers for MaxSAT that, in effect, employ a branch-and-bound search but do not have unit propagation or clause learning incorporated. The MaxSAT instances that can be solved by exact solvers in practice are generally much smaller than instances that can be handled by SAT solvers [cf. 6].

The question we ask in this work is the following: *can traditional SAT solver techniques like unit propagation and clauses learning be directly used as effective heuristics for an incomplete MaxSAT algorithm?* Our results demonstrate that the answer is clearly affirmative, as long as the instances are not too easy to prove unsatisfiable. Specifically, we consider a relaxation of the standard DPLL-based solver where it is allowed to essentially *ignore* the first ℓ conflicts. We show that this strategy is able to solve challenging industrial MaxSAT instances that are hard to prove unsatisfiable, specifically *all* unsatisfiable instances from SAT Race 2008 [10], better than currently available exact as well as approximate alternative techniques. In a few of these instances where the candidate MaxSAT solution found by this approach is sub-optimal, we show that performing a low-noise (and hence very greedy and fast) local search initiated at this candidate solution is often results in an optimal MaxSAT solution within seconds.[2]

Bottleneck Constraints: Interestingly, our approach revealed that the optimal solutions for all but one of the unsatisfiable industrial instances from SAT Race 2008 have only *one* violated clause. Note that, in contrast, all other MaxSAT solvers incorrectly suggest that these very instances have hundreds, if not thousands, of violated clauses in the best MaxSAT solutions. This one clause can be thought of as a *bottleneck constraint* whose semantics can sometimes be used to guide the problem designer towards appropriate additional resources that may be acquired to turn the instance into a feasible one. Of course, it is unclear that a bottleneck constraint always provides meaningful semantic information about additional resources that are realistic to acquire; the violated constraint

[1] Unit propagation is used *indirectly* in many exact MaxSAT solvers to generate good lower bounds [cf. 6, 8], and so are resolution-based inference mechanisms [6]. We compare our results against these approaches as well as local search methods.

[2] Performing local search also works for instances that are too easy to prove unsatisfiable and thus not ideal for us; here our method falls back to pure local search.

could, in principle, be a "frame axiom" or "consistency constraint" enforcing that the encoding is meaningful.

To investigate this further, we conducted experiments on AI planning instances for which we have full knowledge of the variable and clause semantics. Specifically, we considered the TPP (Traveling Purchase Problem) domain from the IPC-5 Competition [3], giving the planner one fewer time step than what it needs to solve the instance. We translated instances of this infeasible problem into unsatisfiable SAT formulas following the SatPlan framework [4]. Our solver, `RelaxedMinisat`, was able to identify a single violated clause in instances thus generated. With different random seeds, we obtained a number of different "explanations" of the unsatisfiability of each instance, in the form of a bottleneck constraint. As an example, in an instance involving 10 goods to be purchased in various quantities and transported, 1 storage depot, 3 markets, and 3 trucks, a violated bottleneck constraint had the following precondition-style semantics: "in order to drive truck-3 from depot-1 to market-2 at time-step-7, truck-3 must be at depot-1 at the end of time-step-6." One thing this bottleneck constraint suggests is that there is a plan that "almost" works if the post condition achieved by the action under consideration is somehow made to hold, i.e., if we could somehow make a truck available at market-2 at the end of time-step-7. Indeed, if we add a fourth truck as a new resource at market-2 as part of the initial conditions, the instance becomes satisfiable in the given time steps.

This suggests that the small number of bottleneck constraints identified by our approach can provide useful semantic information about potential additional resources that can make the problem feasible. This information is complementary to that provided by, for example, "minimal unsatisfiable cores" and related concepts [cf. 8]. For the specific example discussed above, the minimal unsat core returned by the `zChaff` solver involves 522 clauses. For a more detailed discussion of this motivation, we refer the reader to our recent related work [5].

2 Using Relaxed DPLL as a Heuristic for MaxSAT

Due to lack of space we assume familiarity with Boolean formulas in conjunctive normal form (CNF), the satisfiability testing problem (SAT), the maximum satisfiability problem (MaxSAT), the standard DPLL style systematic backtrack search for SAT with conflict clause learning, and basic local search. The reader may want to refer to the Handbook of Satisfiability [1] for a review.

The idea behind our solver, `RelaxedMinisat`, is relatively simple: use a state-of-the-art DPLL solver such as `Minisat` [2] but relax it to "ignore" a fixed number ℓ of conflicts on each search branch, and quit once a truth assignment violating at most ℓ clauses is found. If necessary, run a low-noise local search initiated at the partial assignment found by the DPLL solver.

We chose `Minisat` [2] as the DPLL solver to build on. To implement the one extra feature we need—allowing up to ℓ conflicts on a search branch—we slightly modify the routine performing unit propagation. When a conflict is detected on a search branch b and it is amongst the first ℓ conflicts along b, the clause

causing the conflict is silently ignored until the solver later backtracks the closest branching decision made before getting to this point. All other functionality of the solver is left intact, including clause learning. If ℓ conflicts are reached on the branch b, conflict directed backtracking is performed as usual. It is not hard to see that the conflict clause C learned at this point has the following property: *if any truth assignment σ satisfies all clauses except the ℓ conflict generating clauses seen on branch b, then σ also satisfies C.* Adding C as a learned clause therefore preserves the soundness of the technique. (But C could, in principle, rule out other potential solutions that violate a different set of ℓ or fewer clauses; adding C therefore does not preserve completeness.) If a solution is found before reaching ℓ conflicts, this solution is reported as a candidate MaxSAT solution; this provides an upper bound for the optimal MaxSAT solution. Alternatively, `RelaxedMinisat` can return the "UNSAT" status, in which case we increase the parameter ℓ to attempt to find some other truth assignment. (The "UNSAT" status of `RelaxedMinisat` does *not* mean that there is no assignment violating at most ℓ clauses.) Using binary search, one can find the smallest value of ℓ for which `RelaxedMinisat` does report an assignment; for nearly satisfiable instances such as the ones with very few bottleneck constraints that we focus on, this is very quick as ℓ is small. Experiments suggest that *rapid restarting* improves the performance of `RelaxedMinisat`. We restart after every 100 conflicts.

For some of the harder formulas, the candidate solution found by `RelaxedMinisat` was not clearly optimal (i.e., had more than one violated clause). In this case, we ran the local search solver `Walksat` [9] (without any modification) with the search initiated at this candidate truth assignment, looking for a better solution. Empirically, we observed that rapid restarts are again beneficial, along with very low noise to make the local search extremely greedy and focused.

3 Experimental Results

We conducted experiments on all 52 unsatisfiable formulas from the SAT-Race 2008 suite [10], which are all non-trivial to prove unsatisfiable and, as we will see, often beyond the reach of existing MaxSAT solvers. The solvers used in the comparison were from four families: exact MaxSAT solvers `maxsatz` [6] and `msuf` [8]; local search SAT solvers `saps` [11] and `adaptg2wsat+p` [7]; our previous hybrid approach `MiniWalk` [5] which tries to combine the power of DPLL and local search methods; and `RelaxedMinisat`. We used a cluster of 3.8 GHz Intel Xeon computers running Linux 2.6.9-22.ELsmp with a time limit of 1 hour (see two exceptions discussed below) and a memory limit to 2 GB.

The exact MaxSAT solvers selected were those that performed exceptionally well in MaxSAT Evaluation 2007, especially on industrial instances. The local search algorithms were selected as the best performing ones on our suite from a wide pool of choices offered by the `UBCSAT` solver [12]. Three runs for each problem and local search algorithm were performed with default parameters (or those used in the accompanying papers for the solvers, e.g., $\alpha = 1.05$ for `saps`), and the best run is reported.

For `RelaxedMinisat`, we first run the modified DPLL part allowing at most one conflict ($\ell = 1$), and increase this relaxation parameter when necessary. We report in Table 1 the final number of unsatisfied clauses reached, followed by, in parentheses, the relaxation parameter, the time taken by the DPLL part, and the additional time taken to run the local search part afterwards (if applicable). In fact, in only 8 instances the DPLL part alone did not find the sure optimum solution; for these instances we increase ℓ (not necessarily by one) to the number reported in the table so that a candidate truth assignment is found. The table reports the time for the last run of `RelaxedMinisat`, with the appropriate value of ℓ (often 1). On only two instances did `RelaxedMinisat` require more than one hour (ibm-2002-22r-k60 and post-c32s-gcdm16-23). For local search, we use the default quick restarts of `Walksat` every 100,000 flips. (However, for post-c32s-col400-16 and post-c32s-gcdm16-23 instances, this was increased to 1 million.) In all cases the noise parameter for `Walksat` was set to a low value of 5%.

Table 1 shows that `RelaxedMinisat` is the only solver able to solve 51 of out 52 instances to optimality. (For aloul-chnl11-13, we know that the optimum is not 1.) Moreover, 39 of these instances, i.e. 76%, were solved in under one minute. In contrast, the best local search approaches often could not find a solution violating less than a few hundred or thousand clauses in one hour of computation time. The hybrid method, `MiniWalk`, showed better performance but was still unable to solve to optimality 16 instances out of 52 (i.e., 30% of the instances). These results demonstrate that our relaxed DPLL solver is able to efficiently identify bottleneck constraints in challenging unsatisfiable problem instances.

References

[1] A. Biere, M. J. H. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability*, vol. 185 of *Frontiers in AI and Applications*. IOS Press, Feb. 2009.

[2] N. Eén and N. Sörensson. MiniSat: A SAT solver with conflict-clause minimization. In *8th SAT*, St. Andrews, U.K., June 2005.

[3] A. Gerevini, Y. Dimopoulos, P. Haslum, and A. Saetti (Organizers). IPC-5 international planning competition, June 2006. URL http://zeus.ing.unibs.it/ipc-5.

[4] H. A. Kautz and B. Selman. BLACKBOX: A new approach to the application of theorem proving to problem solving. In *Workshop on Planning as Combinatorial Search, held in conjunction with AIPS-98*, Pittsburgh, PA, 1998.

[5] L. Kroc, A. Sabharwal, C. P. Gomes, and B. Selman. Integrating systematic and local search paradigms: A new strategy for MaxSAT. In *21st IJCAI*, July 2009.

[6] C. M. Li, F. Manyà, J. Planes. New inference rules for Max-SAT. *JAIR*, 30, 2007.

[7] C. M. Li, W. Wei, and H. Zhang. Combining adaptive noise and look-ahead in local search for SAT. In *10th SAT*, pp. 121–133, Lisbon, Portugal, May 2007.

[8] J. P. Marques-Silva and V. M. Manquinho. Towards more effective unsatisfiability-based maximum satisfiability algorithms. In *11th SAT*, pp. 225–230, May 2008.

[9] B. Selman, H. Kautz, and B. Cohen. Local search strategies for satisfiability testing. *The Second DIMACS Implementation Challenge*, pp. 521–532, 1996.

[10] C. Sinz (Organizer). SAT-race 2008, May 2008.

[11] D. A. D. Tompkins and H. H. Hoos. Scaling and probabilistic smoothing: Dynamic local search for unweighted MAX-SAT. In *16th Canadian AI*, pp. 145–159, 2003.

[12] D. A. D. Tompkins and H. H. Hoos. UBCSAT: An impl. and experim. env. for SLS algorithms for SAT and MAX-SAT. In *7th SAT*, Vancouver, BC, May 2004.

Table 1. Comparison of MaxSAT results for exact, local search, hybrid, and currently proposed methods. Timelimit: 1 hour (except for two instance which took two hours). If a sure optimum was achieved (i.e., one unsatisfied clause), the time is reported in parenthesis. The `RelaxedMinisat` column shows the final number of unsatisfied clauses reached, and in parenthesis the number of allowed conflicts ℓ, time for the DPLL part, and, if applicable, time for local search.

Instance	#vars	#cls	Exact	Local Search		Hybrid	**Relaxed DPLL**
			#unsat	best #unsat		best #uns	best #unsat
			`maxsatz` or `msuf`	`Adapt-g2wsat+p`	`SAPS`	`MiniWalk`	`RelaxedMinisat`
babic-dspam-vc1080	118K	375K	—	728	306	20	**1** **(1,35s,–)**
babic-dspam-vc973	274K	908K	—	2112	1412	267	**1** **(4,100s,44s)**
ibm-2002-22r-k60	209K	851K	—	198	409	10	**1** **(3,115m,1s)**
ibm-2002-24r3-k100	148K	550K	—	205	221	2	**1** **(1,7m,–)**
manol-pipe-f7nidw	310K	923K	—	810	797	7	**1** **(1,3m,–)**
manol-pipe-f9b	183K	547K	—	756	600	177	**1** **(1,8s,–)**
manol-pipe-g10nid	218K	646K	—	585	727	27	**1** **(1,12s,–)**
manol-pipe-g8nidw	121K	358K	—	356	336	7	**1** **(1,6s,–)**
post-c32s-col400-16	286K	840K	—	88	111	698	**1** **(50,1m,8s)**
post-c32s-gcdm16-23	136K	404K	—	25	225	127	**1** **(3,100m,1m)**
post-cbmc-aes-ele	277K	1601K	—	864	781	2008	**1** **(1,14s,–)**
simon-s03-fifo8-400	260K	708K	—	89	289	13	**1** **(1,11m,–)**
aloul-chnl11-13	286	1742	—	**4**	**4**	**4**	**4** **(4,3s,×)**
anbul-dated-5-15-u	152K	687K	—	12	22	1 (15m)	**1** **(1,8s,–)**
een-pico-prop05-75	77K	248K	—	2	47	**1** **(4s)**	1 (1,2m,–)
fuhs-aprove-15	21K	74K	—	35	31	**1** **(0s)**	**1** **(1,0s,–)**
fuhs-aprove-16	52K	182K	—	437	246	1 (1s)	**1** **(1,0s,–)**
goldb-heqc-dalumul	9426	60K	—	11	10	**1** **(0s)**	1 (1,1s,–)
goldb-heqc-frg1mul	3230	21K	—	**1 (0s)**	**1 (0s)**	**1** **(0s)**	**1** **(1,0s,–)**
goldb-heqc-x1mul	8760	56K	—	**1 (0s)**	**1 (0s)**	**1** **(0s)**	**1** **(1,0s,–)**
hoons-vbmc-lucky7	8503	25K	—	**1 (0s)**	3	9	1 (7,27s,36s)
ibm-2002-25r-k10	61K	302K	—	111	95	1 (9s)	**1** **(1,2s,–)**
ibm-2002-31_1r3-k30	44K	194K	—	78	101	**1** **(2s)**	1 (1,6s,–)
ibm-2004-29-k25	17K	78K	—	14	12	1 (6m)	**1** **(1,5s,–)**
manol-pipe-c10nid_i	253K	751K	—	678	695	1 (20m)	**1** **(1,13s,–)**
manol-pipe-c10nidw	434K	1292K	—	1013	1363	**1** **(16s)**	1 (1,37m,–)
manol-pipe-c6bidw_i	96K	284K	—	239	274	1 (24s)	**1** **(1,3s,–)**
manol-pipe-c8nidw	269K	800K	—	697	742	**1** **(7s)**	1 (1,6m,–)
manol-pipe-c9n_i	35K	104K	—	214	66	1 (3s)	**1** **(1,0s,–)**
manol-pipe-g10bid_i	266K	792K	—	723	822	1 (103s)	**1** **(1,18s,–)**
post-c32s-ss-8	54K	148K	—	1 (2s)	1 (8s)	**1** **(0s)**	**1** **(1,0s,–)**
post-cbmc-aes-d-r2	278K	1608K	—	834	734	1 (69s)	**1** **(1,8s,–)**
post-cbmc-aes-ee-r2	268K	1576K	—	839	760	1 (37s)	**1** **(1,12s,–)**
post-cbmc-aes-ee-r3	501K	2928K	—	1817	1822	1 (37m)	**1** **(1,47s,–)**
schup-l2s-abp4-1-k31	15K	48K	—	7	16	**1** **(0s)**	1 (1,2s,–)
schup-l2s-bc56s-1-k391	561K	1779K	—	5153	26312	**1** **(168s)**	1 (1,11m,–)
simon-s02-f2clk-50	35K	101K	—	1 (110s)	32	**1** **(12s)**	1 (1,17s,–)
velev-vliw-uns-2.0-iq1	25K	261K	—	1 (40m)	4	**1** **(0s)**	**1** **(1,0s,–)**
velev-vliw-uns-2.0-iq2	44K	542K	—	2	2	**1** **(1s)**	**1** **(1,1s,–)**
velev-vliw-uns-2.0-uq5	152K	2466K	—	40	11	1 (18s)	**1** **(1,4s,–)**
velev-vliw-uns-4.0-9-i1	96K	1814K	—	12	10	1 (23s)	**1** **(1,4s,–)**
velev-vliw-uns-4.0-9	154K	3231K	—	2	3	1 (10s)	**1** **(1,3s,–)**
babic-dspam-vc949	113K	360K	1 (315s)	797	216	250	**1** **(2,10s,0s)**
cmu-bmc-barrel6	2306	8931	1 (19m)	**1 (0s)**	**1 (0s)**	**1** **(0s)**	**1** **(1,0s,–)**
cmu-bmc-longmult13	6565	20K	1 (171s)	5	12	1 (1s)	**1** **(1,0s,–)**
cmu-bmc-longmult15	7807	24K	1 (137s)	6	4	1 (5s)	**1** **(1,0s,–)**
een-pico-prop00-75	94K	324K	1 (4m)	23	108	276	**1** **(2,2m,1s)**
goldb-heqc-alu4mul	4736	30K	1 (14m)	1 (105s)	1 (47m)	1 (1s)	**1** **(1,0s,–)**
jarvi-eq-atree-9	892	3006	1 (158s)	**1 (0s)**	**1 (0s)**	**1** **(0s)**	**1** **(1,0s,–)**
marijn-philips	3641	4456	1 (336)	**1 (0s)**	**1 (0s)**	**1** **(0s)**	**1** **(1,0s,–)**
post-cbmc-aes-d-r1	41K	252K	1 (177s)	7	10	**1** **(1s)**	**1** **(1,1s,–)**
velev-engi-uns-1.0-4nd	7000	68K	1 (76s)	1 (3s)	2	**1** **(0s)**	**1** **(1,0s,–)**