Multi-Core strategy combining local and systematic search for boolean optimization

IJCAI-09 conference

Integrating Systematic and Local Search Paradigms: A New Strategy for MaxSAT

Lukas Kroc and **Ashish Sabharwal** and **Carla P. Gomes** and **Bart Selman**
Department of Computer Science, Cornell University
Ithaca, NY 14853-7501, U.S.A.
{kroc,sabhar,gomes,selman}@cs.cornell.edu

Abstract

Systematic search and local search paradigms for combinatorial problems are generally believed to have complementary strengths. Nevertheless, attempts to combine the power of the two paradigms have had limited success, due in part to the expensive information communication overhead involved. We propose a hybrid strategy based on shared memory, ideally suited for multi-core processor architectures. This method enables continuous information exchange between two solvers without slowing down either of the two. Such a hybrid search strategy is surprisingly effective, leading to substantially better quality solutions to many challenging Maximum Satisfiability (MaxSAT) instances than what the current best exact or heuristic methods yield, and it often achieves this within seconds. This hybrid approach is naturally best suited to MaxSAT instances for which proving unsatisfiability is already hard; otherwise the method falls back to pure local search.

1 Introduction

Boolean Satisfiability (SAT) solvers have seen tremendous progress in recent years. Several of the current best open source SAT solvers scale up to instances with over a million variables and several million clauses. These advances have led to an ever growing range of applications, such as in hardware and software verification, and planning (cf. Handbook of SAT [Biere *et al.*, 2009]). 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 or DPLL [Davis and Putnam, 1960; Davis *et al.*, 1962] style complete, systematic search. The competing search paradigm for SAT solving is based on local search (cf. [Hoos and Stützle, 2004]), which performs well in certain problem domains but, in general, is not as effective on highly structured problem domains.

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, or constraints, as possible. Even though MaxSAT is a natural generalization of SAT, and thus closely related, progress has been much slower on efficient solution strategies for the MaxSAT problem. There is a good explanation as to why this is the case. Two key components behind the rapid progress for DPLL based SAT solvers are: highly effective unit propagation, and clause learning. (Other factors include randomization and restart strategies, and effective data structures.) Both techniques in a sense focus on avoiding local inconsistencies: when a unit clause occurs in a formula, one should immediately assign a 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, at least in the context of branch-and-bound techniques, can be quite counterproductive and in fact lead to incorrect results. For example, for an unsatisfiable instance, the optimal 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 which effectively steers the search away from the contradiction. Hence, neither unit propagation nor clause learning appear directly suitable for a MaxSAT solver. Unfortunately, taking such mechanisms out of the DPLL search strategy dramatically reduces the effectiveness of the search. 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 clauses learning incorporated. Although progress has been made in the area of exact MaxSAT solvers, the instances that can be solved in practice are generally much smaller than instances that can be handled by SAT solvers. Just as an example, a bounded model checking instance considered in our experiments, cmu-bmc-barrel6.cnf, is solved by the state-of-the-art exact MaxSAT solvers such as `maxsatz` [Li *et al.*, 2007a] and `msuf` [Marques-Silva and Manquinho, 2008] in 20-30 minutes while within only a couple of seconds by an exact SAT solver like `MiniSat` [Eén and Sörensson, 2005].

A more natural fit for the MaxSAT problem is to use local search. Such methods are incomplete but they can find approximate solutions for large problem instances. The ad-

vantage of a local search strategy is that it in a sense operates in a more global manner, generally using the current number of unsatisfied clauses as a gradient to guide a further descent. In this process local search doesn't hesitate to violate unit clauses if that appears beneficial. Some of the earliest local search results for MaxSAT were based on the `Walksat` procedure [Selman *et al.*, 1996]. From the perspective of local search, the basic SAT and MaxSAT strategies are quite similar. Researchers have recently showed that more sophisticated local search strategies can significantly improve upon the `Walksat` performance. For example, two state-of-the-art local search solvers, which also work very well for MaxSAT, are `saps` [Hutter *et al.*, 2002; Tompkins and Hoos, 2003] and `adaptg2wsat+p` [Li *et al.*, 2007b].

The performance of the recent local search methods on large hard problem instances (unsatisfiable instances at the edge of feasibility for current SAT solvers) appears impressive. For example, on the industrial instance babic-dspam-vc973.cnf from the SAT Race-2008 [Sinz (Organizer), 2008], a typical unsatisfiable benchmark instance with 900,000+ clauses, `Walksat` can find an assignment that leaves around 35,000 clauses unsatisfied but `saps` and `adaptg2wsat` can find solutions with around 1,500 clauses unsatisfied (see Table 1). A systematic SAT solver, `MiniSat` [Eén and Sörensson, 2005], can prove the instance to be unsatisfiable in around 3 hours. The question remains, how close the obtained MaxSAT solution is to the optimal solution? The fact that different local search MaxSAT methods converge to roughly the same number of unsatisfied clauses and running them for many more hours does not further improve the solution may lead one to conjecture that 1,000 might be close to optimal. However, one reason to be less confident of the quality of the solution is that local search methods have been shown to have trouble dealing with long chains of dependencies in structured problem instances, even though special encodings and the addition of inferred clauses can help alleviate some of these problems (e.g., [Prestwich, 2007; Hirsch and Kojevnikov, 2005]).

The issue of problem structure, on which DPLL methods excel but which challenges local search style methods, leads us naturally to the two main questions addressed in this paper: (1) *How good are the current best solutions on structured problems?* (2) *Can state-of-the-art SAT solvers help on the MaxSAT problem?* As we will demonstrate, the current best solutions on structured problems are often surprisingly far from optimal. We show this by finding solutions with a single unsatisfied clause (and therefore optimal) for unsatisfiable problem instances, where the best previously known solutions had hundreds or thousands of unsatisfied clauses. For example, the optimal solution for the instance mentioned above does not have around 1,000 unsatisfied clauses but actually just one (out of 900,000+ clauses total). We demonstrate this using a new solver that came forth out of our study of the second question—can we use DPLL to boost a local search solver? We introduce a **hybrid solution strategy**, where information from a DPLL SAT solver provides continued guidance for a local search style MaxSAT solver. We use `MiniSat` as our DPLL solver and `Walksat` as our MaxSAT solver. Our hybrid solver is called **MiniWalk**. The integration of the solvers is surprisingly clean, requiring only a few lines of code.

The incremental, systematic search approach behind backtrack search (as in `MiniSat`) and the stochastic local search approach (as in `Walksat`) represent the two main combinatorial search paradigms. It appears natural to integrate these approaches to leverage each other's strengths. In fact, there have been various attempts at such integration, for example, using local search to find good branching variables for DPLL or to identify minimal unsatisfiable subsets (e.g., [Mazure *et al.*, 1998; Grégoire *et al.*, 2007a]). However, these attempts, especially those targeted at traditional SAT solving, have not been as effective as one would have hoped. One particular issue that hampers integration of solvers in general is that time spent in the less effective solver is often more costly than the time saved by the faster solver when using the information obtained with the slower one. More concretely, although `Walksat` may be able to provide better branching information leading to a smaller DPLL tree, the time saved through the reduction in tree size is often less than the time spent on running `Walksat`. Hence, the issue of how much time to spend in each solver becomes a careful balancing act, and is often problem instance dependent.

Fortunately, the compute paradigm based on multi-core processors eliminates much of these difficulties. In particular, in our hybrid solver, we run both `MiniSat` and `Walksat` at full speed in parallel on two different cores of a standard dual-core processor. During the run, `MiniSat` writes its current branching information into a shared memory. More specifically, this memory contains the values of all variables that are set in the current branch of the DPLL search. Running on the other processor, before flipping the value of a variable, `Walksat` "peeks" at the shared memory and only makes the flip if the variable is not set on the DPLL branch or if the flip will set the variable on the branch to its current value. (Stated differently, `Walksat` does not flip any variable to a setting conflicting with that of the current DPLL branch.) In this setting, information from DPLL continuously steers the `Walksat` search strategy. And, as we will see, the DPLL search frequently steers `Walksat` to extremely promising regions of the search space, where near-satisfying assignments exist. Moreover, without the guidance, `Walksat` or other local search methods do not appear to reach such promising areas of the search space. We will discuss the search behavior of our hybrid strategy in more detail in the text. We will see that our hybrid search progresses in a manner not observed in any non-hybrid search strategy. We again stress that the dual-core mechanism is a key factor behind the success of our approach, because it eliminates the need for intricate time allocations for the two types of search.

In summary, our results show that DPLL can provide highly effective guidance for a local search style solver for the MaxSAT problem, leading to the optimal solution on many structured instances. From the SAT Race-2008 benchmark set, we find a provably optimal solution (one unsatisfied clause) on 37 of the 52 unsatisfiable instances,[1] while

[1] We have recently been able to further improve upon this using

the current best alternative approaches suggest hundreds, if not thousands, of unsatisfied clauses in the best solutions to these very instances. This work therefore provides a step towards closing the performance gap between SAT solvers and MaxSAT solvers. The results also demonstrate that there is a real potential in using multi-core processors in combinatorial search, where shared memory is used to provide a low-cost communication channel between the processes.

We note that the focus of our hybrid approach is naturally on instances that are non-trivial for both DPLL and local search solvers. Most of the MaxSAT benchmarks currently available, such as the ones used in MaxSAT Evaluation 2007 [Argelich *et al.*, 2007], are targeted towards the scalability region of exact MaxSAT solvers, and are thus too easy for DPLL-based SAT solvers such as `Minisat`. On such instances, the DPLL part of our hybrid solver, `MiniWalk`, often terminates within a second, providing little guidance to the local search part. The hybrid method therefore essentially falls back to pure local search, performing no better (but also no worse) than alternative approaches on these instances. To illustrate the strength and promise of our approach, we perform an evaluation on *all* unsatisfiable instances used in SAT Race-2008, which are challenging not only as MaxSAT instances but also as satisfiability instances. We hope our positive results will encourage the development of MaxSAT benchmarks that are non-trivial to prove unsatisfiable.

We also note that given the performance of other MaxSAT solvers on the SAT Race-2008 instances, it is quite surprising that all but one of these instances have only a *single* unsatisfied clause in the optimal MaxSAT solution. This clause can be thought of as a "bottleneck constraint" for the instance. In fact, by running our solver multiple times with different random seeds, we can identify several different bottleneck constraints, relaxing any one of which will turn the instance into a satisfiable one. In this sense, bottleneck constraints provide a form of explanation of unsatisfiability, complementing the information provided by other concepts being explored in the literature such as minimal unsatisfiable cores, minimal sets of unsatisfiable tuples, etc. (cf. [Marques-Silva and Manquinho, 2008; Grégoire *et al.*, 2007a; 2007b]). The kind of information provided by the single violated constraint obviously depends on the problem encoding. In the standard AI planning encodings based on the Satplan/Blackbox framework [Kautz and Selman, 1998], we found that, contrary to what one might expect, the bottleneck constraint is often not simply the "goal predicate" that is being violated. It is, in fact, more common to find clauses encoding constraints in the intermediate steps of the plan. The semantic meaning of the violation of the constraint is tightly tied to the problem domain and its encoding. Often how one may physically "fix" the issue highlighted by the violated bottleneck constraint is not obvious. Designing special encodings where violated constraints do indicate ways to fix the underlying issue is an interesting direction for future research.

The rest of the paper is organized as follows. After discussing some basic concepts, we present our hybrid solver, `MiniWalk`, in more detail in Section 3. In Section 4, we evaluate the performance of `MiniWalk` and compare it with other state-of-the-art MaxSAT solvers. In Section 5, we show how the search performed by `MiniWalk` differs qualitatively from that of other search methods. Finally, we provide concluding remarks in Section 6.

a "relaxed DPLL" approach, which shows that in fact as many as 51 out of the 52 unsatisfiable SAT Race-2008 instances have only one unsatisfied clause in the optimal solution [Kroc *et al.*, 2009].

2 Preliminaries

Let V be a set of propositional (Boolean) variables, which take value in the set $\{0,1\}$. We think of 1 as True and 0 as False. Let F be a propositional formula over V. A *solution* to F (also referred to as a satisfying assignment for F) is a 0-1 assignment to all variables in V such that F evaluates to 1. *Propositional Satisfiability* or SAT is the decision problem of determining whether an input formula F has any solutions. This is the canonical NP-complete problem. In practice, one is also interested in finding a solution, if there exists one.

Instances of the SAT problem are often specified in the Conjunctive Normal Form (CNF). Here F is given as a conjunction of *clauses*, each clause is a disjunction of *literals*, and each literal is either a variable or its negation. For example, $F = (a \vee \neg b) \wedge (\neg a \vee c)$ is a CNF formula.

When a CNF formula F is unsatisfiable, i.e., there is no truth assignment to the variables in V for which all clauses of F are satisfied, one is often interested in solving the problem as much as possible. Formally, *Maximum Satisfiability* or MaxSAT is the optimization problem of finding a truth assignment that satisfies as many clauses of an input formula F as possible. We will refer to such truth assignments as *optimal MaxSAT solutions*. One natural quantity of interest when performing a search for an optimal MaxSAT solution is the *number of unsatisfied clauses* found at the end of a search procedure. As we will see in Section 4, the optimal MaxSAT solutions for many interesting industrial problem instances happen to have only one unsatisfied clause, and the proposed hybrid method is often able to find such solutions very efficiently.

Most of the successful search methods for SAT can be classified into two categories: systematic complete search and heuristic local search. For SAT, systematic complete search takes the form of the Davis-Putnam-Logemann-Loveland or DPLL procedure [Davis and Putnam, 1960; Davis *et al.*, 1962]. The idea is to do a standard branch-and-backtrack search in the space of all partial truth assignments. Heuristics are used to set variables to promising values until either a solution is found or a contradiction is reached; in the latter case, the solver backtracks, flips the value of a variable higher up in the search tree, and systematically continues the search for a solution—now in a previously unexplored part of the search space. Modern SAT solvers based on DPLL employ additional techniques such as clause learning, restarts, highly efficient data structures, etc. While the systematic solver, `Minisat`, that forms one half of our hybrid approach does implement these advanced techniques, the details of these techniques are not crucial for understanding the rest of this paper.

Local search SAT solvers, also referred to as stochastic lo-

cal search or SLS solvers, work with complete truth assignments which, of course, violate some number of clauses before a solution is found. The idea here is to do local modifications to the current complete truth assignment, guided essentially by the currently unsatisfied clauses. The local modifications often take the form of heuristically selecting one variable to *flip*, based often on how many of the currently unsatisfied clauses will become satisfied and how many of the currently satisfied clauses will become unsatisfied. Refined local search solvers employ techniques such as selecting variables mostly only from currently unsatisfied clauses, clause re-weighting and stochastic noise to escape local minima, adaptively adjusting the noise level, etc.

3 Using DPLL to Guide Local Search

Our hybrid MaxSAT solver, `MiniWalk`, has two parts that are very independent except for sharing a small amount of memory for information exchange: a DPLL solver and a local search solver. The main idea is the following: *`MiniSat` informs the local search which part of the search space it is currently searching in, and `Walksat` loosely restrains itself to the same part of the search space by not flipping a literal against `MiniSat`.*

Both DPLL and local search are performed *simultaneously*, so there are literally two processes (solvers) running at the same time. This does not slow down the performance of either, given the multi-core architecture that is becoming a standard in the computer industry and the very low communication overhead involved. The details will follow shortly, but let us remark already that any DPLL heuristic and any local search heuristic can be instrumented to create a hybrid solver in this fashion, and the actual implementation requires only a few additional lines of code.

A DPLL solver proceeds by successively selecting variables and their polarities (truth values), fixing those variables accordingly, and simplifying the instance. A lot of effort has been invested in designing heuristics that guide the search into parts of the search space where solutions are likely to be found. A good DPLL heuristic often guides the search in the direction where as many clauses as possible are satisfied. We use this search bias even on formulas that are not satisfiable, assuming that good near-solutions will lie in the regions that look attractive to a DPLL heuristic. The information about the region of the search space that the DPLL solver is currently exploring is communicated using a *shared memory array*. In this array shared between the two solver processes, each variable of the problem instance has either the value "unassigned" or the polarity (0 or 1) that is currently assigned to it by the DPLL search. The only change to the code of the DPLL solver is thus a line that writes the correct polarity every time a variable is fixed (branched on or set, e.g., by unit propagation). The variable value in the shared array is reverted back to "unassigned" upon backtracking.[2]

On the local search side, the modification is also only very slight. The standard local search procedure has two steps that keep repeating: pick a variable to flip, and flip the selected variable. The only modification we make is in the second step: we flip the truth value of a variable *only if* the new value does not violate the setting the DPLL search has for that variable in the shared array. In other words, we flip a variable from, say, TRUE to FALSE only if it is either unassigned by the DPLL search or is assigned FALSE. Otherwise we simply do nothing and `Walksat` selects a different variable to flip in the next step. This is depicted formally as Algorithm 1. The two steps marked with "***" are literally the only change that needs to be made to pure `Walksat`.

Algorithm 1: Hybrid-Walksat part of `Miniwalk`

```
begin
    *** Initialize shared memory array M
    σ ← a randomly generated truth assignment for F
    for j ← 1 to MAX-FLIPS do
        if σ satisfies F then return σ
        Select a variable v using a heuristic
        *** if M[v] ≠ value(v) then
            Flip the value of v in σ
end
```

This way, the information flows only in one direction: from DPLL to the local search. The local search is responsible for reporting the best-so-far achieved assignment, which is in turn used as an estimated solution to the MaxSAT instance. If the DPLL search finishes very quickly (i.e., easily determines unsatisfiability within a few seconds), then the local search has no time to take advantage of the guidance provided by the shared array, and the hybrid method does not improve upon plain local search. If, on the other hand, we have a sufficiently hard instance at hand, we found that this strategy is remarkably successful at finding very good solutions.

There is, of course, the question of which DPLL and local search solvers to select. The `Minisat` and `Walksat` combination turned out to perform the best. We also considered `Rsat` DPLL solver, which adds the concept of "restart memory" to the search. While successful for SAT, we found that `Rsat` did not perform as well as `Minisat` for our purposes, perhaps because the memory constrained the search to too local a region and also because `Rsat` generally terminates quicker than `Minisat`, providing less guidance to `Walksat`. On the local search side, we tried using more powerful algorithms, namely `adaptg2wsat+p` and `saps`, which performed best on our problem suite as stand-alone local search solvers (see Section 4). But we found that neither performed as well as `Walksat` when coupled with a DPLL solver, presumably because their decisions/flips were much more focused than `Walksat`'s, and it was harder for the solvers to follow the DPLL guidance.

4 Experimental Results

We conducted experiments on all 52 unsatisfiable formulas from the SAT Race-2008 suite [Sinz (Organizer), 2008]. The reason for choosing these instances rather than the Max-SAT Evaluation 2007 [Argelich *et al.*, 2007] instances is that the latter are all too easy for `MiniSat`, thus limiting the DPLL

[2]There are a few lines of initial code for declaring the shared memory array; see Appendix for completeness.

Table 1: Comparison of MAXSAT results for exact, local search, and hybrid methods. Timelimit: 1 hour. If a sure optimum was achieved (i.e., 1 unsatisfied clause), the time is reported in parenthesis. Note: the superscript "(2)" for babic-dspam-vc973.cnf denotes that this instance gets down to 267 unsatisfied clauses using `MiniWalk` in the 1 hour time limit, and is solved to optimality within three hours.

			Exact Methods	Local Search Methods			**Hybrid**
			#unsat	best #unsat			best #unsat
Instance	#vars	#clauses	`maxsatz` or `msuf`	`Adapt-g2wsat+p`	`SAPS`	`Walksat`	`MiniWalk`
anbul-dated-5-15-u	152K	687K	—	12	22	266	**1 (15m)**
een-pico-prop05-75	77K	248K	—	2	47	325	**1 (4s)**
fuhs-aprove-15	21K	74K	—	35	31	430	**1 (0s)**
fuhs-aprove-16	52K	182K	—	437	246	1993	**1 (1s)**
ibm-2002-25r-k10	61K	302K	—	111	95	1122	**1 (9s)**
ibm-2002-31_1r3-k30	44K	194K	—	78	101	182	**1 (2s)**
ibm-2004-29-k25	17K	78K	—	14	12	170	**1 (6m)**
manol-pipe-c10nid_i	253K	751K	—	678	695	5211	**1 (20m)**
manol-pipe-c10nidw	434K	1292K	—	1013	1363	22554	**1 (16s)**
manol-pipe-c6bidw_i	96K	284K	—	239	274	924	**1 (24s)**
manol-pipe-c8nidw	269K	800K	—	697	742	13463	**1 (7s)**
manol-pipe-c9n_i	35K	104K	—	214	66	184	**1 (3s)**
manol-pipe-g10bid_i	266K	792K	—	723	822	7622	**1 (103s)**
post-cbmc-aes-d-r2	278K	1608K	—	834	734	5234	**1 (69s)**
post-cbmc-aes-ee-r2	268K	1576K	—	839	760	5160	**1 (37s)**
post-cbmc-aes-ee-r3	501K	2928K	—	1817	1822	10776	**1 (37m)**
schup-l2s-abp4-1-k31	15K	48K	—	7	16	155	**1 (0s)**
schup-l2s-bc56s-1-k391	561K	1779K	—	5153	26312	12882	**1 (168s)**
velev-vliw-uns-4.0-9-i1	96K	1814K	—	12	10	7	**1 (23s)**
velev-vliw-uns-4.0-9	154K	3231K	—	2	3	3	**1 (10s)**
babic-dspam-vc1080	118K	375K	—	728	306	11857	**20**
babic-dspam-vc973	274K	908K	—	2112	1412	32783	**1**$^{(2)}$
ibm-2002-22r-k60	209K	851K	—	198	409	2204	**10**
ibm-2002-24r3-k100	148K	550K	—	205	221	1294	**2**
manol-pipe-f7nidw	310K	923K	—	810	797	15431	**7**
manol-pipe-f9b	183K	547K	—	756	600	9827	**177**
manol-pipe-g10nid	218K	646K	—	585	727	6047	**27**
manol-pipe-g8nidw	121K	358K	—	356	336	1151	**7**
simon-s03-fifo8-400	260K	708K	—	89	289	5939	**13**
goldb-heqc-dalumul	9426	60K	—	11	10	1 (48m)	**1 (0s)**
goldb-heqc-frg1mul	3230	21K	—	**1 (0s)**	**1 (0s)**	**1 (0s)**	**1 (0s)**
goldb-heqc-x1mul	8760	56K	—	**1 (0s)**	**1 (0s)**	**1 (0s)**	**1 (0s)**
post-c32s-ss-8	54K	148K	—	1 (2s)	1 (8s)	1 (4s)	**1 (0s)**
simon-s02-f2clk-50	35K	101K	—	1 (110s)	32	652	**1 (12s)**
velev-vliw-uns-2.0-iq1	25K	261K	—	1 (40m)	4	1 (22s)	**1 (0s)**
velev-vliw-uns-2.0-iq2	44K	542K	—	2	2	1 (6s)	**1 (1s)**
velev-vliw-uns-2.0-uq5	152K	2466K	—	40	11	1 (310s)	**1 (18s)**
aloul-chnl11-13	286	1742	—	**4**	**4**	**4**	**4**
hoons-vbmc-lucky7	8503	25K	—	**1 (0s)**	3	**1 (0s)**	9
post-c32s-col400-16	286K	840K	—	**88**	111	973	698
post-c32s-gcdm16-23	136K	404K	—	**25**	225	3038	127
post-cbmc-aes-ele	277K	1601K	—	864	**781**	5390	2008
cmu-bmc-barrel6	2306	8931	1 (19m)	**1 (0s)**	**1 (0s)**	**1 (0s)**	**1 (0s)**
cmu-bmc-longmult13	6565	20K	1 (171s)	5	12	36	**1 (1s)**
cmu-bmc-longmult15	7807	24K	1 (137s)	6	4	41	**1 (5s)**
goldb-heqc-alu4mul	4736	30K	1 (14m)	1 (105s)	1 (47m)	45	**1 (1s)**
jarvi-eq-atree-9	892	3006	1 (158s)	**1 (0s)**	**1 (0s)**	**1 (0s)**	**1 (0s)**
marijn-philips	3641	4456	1 (336)	**1 (0s)**	**1 (0s)**	**1 (0s)**	**1 (0s)**
post-cbmc-aes-d-r1	41K	252K	1 (177s)	7	10	30	**1 (1s)**
velev-engi-uns-1.0-4nd	7000	68K	1 (76s)	1 (3s)	2	1 (19m)	**1 (0s)**
babic-dspam-vc949	113K	360K	**1 (315s)**	797	216	11818	250
een-pico-prop00-75	94K	324K	**1 (253s)**	23	108	1334	276

guidance provided to the hybrid method to just a few seconds, turning it into essentially local search. The SAT Race instances well illustrate the strengths and promises of the approach, and by using all unsatisfiable ones, we did not bias our selection to only "good" instances. Although not traditional in the MaxSAT domain, we believe that useful information can be obtained from the near-solutions which our technique finds. The usefulness of such information, in the sense of identifying bottleneck constraints, is relatively limited if the minimum number of unsatisfied clauses is large. That is why we do not discuss in depth the performance of our algorithm on instances with no "near solutions," i.e., where the optimal solution has a large number of unsatisfied clauses.

The solvers used in the comparison were from three families: exact MaxSAT solvers `maxsatz` [Li *et al.*, 2007a] and `msuf` [Marques-Silva and Manquinho, 2008]; local search SAT solvers `saps`, `adaptg2wsat+p`, and `walksat`; and our hybrid solver `MiniWalk`. We used a cluster of 3.8 GHz Intel Xeon computers running Linux 2.6.9-22.ELsmp. The time limit for the main experiments was set to 1 hour and the memory limit to 2 GB. The main findings are reported in Table 1.

The two local search algorithms `saps` and `adaptg2wsat+p` were selected as the best performing ones on our suite from a wide pool of choices offered by the `UBCSAT` solver [Tompkins and Hoos, 2004]. Pure `Walksat` was added to contrast performance of an unguided local search with the guided version introduced in this paper. Three runs for each problem and 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.

The exact MaxSAT solvers selected were those that performed exceptionally well in Max-SAT Evaluation 2007, on industrial instances in particular. Nevertheless, only 10 instances in our suite were small enough to be solved by these solvers, and are reported as the bottom two sets of instances in the table. While 8 of these are still solved by `MiniWalk`, such instances are often too easy for `MiniSat` to provide more than a couple of seconds of useful guidance in the hybrid strategy.

More interestingly, out of the 42 remaining harder instances, `MiniWalk` is the only solver that found surely optimal solutions (i.e., with 1 unsatisfied clause) in as many as 20 instances, out of which 13 instances were solved by it in under a minute. These are reported as the first set of instances in Table 1. Note that the previously best known MaxSAT solutions for, e.g., schup-12s-bc56s-1-k391 and post-cbmeaes-ee-r3 had over 5,000 and 1,000 unsatisfied clauses, resp. The second set of instances includes the 9 instances on which `MiniWalk` was able to find significantly better quality solutions than any other technique, often with two orders of magnitude fewer unsatisfied clauses.

The third set of instances in Table 1 includes 9 instances on which other local search methods were able to find equally good solutions as `MiniWalk`, although sometimes taking much longer. Finally, for the fourth set with 4 instances, either `saps` or `adaptg2wsat+p` was able to find a better quality solution than `MiniWalk`.

In summary, we see that on a vast majority of the instances, the hybrid MaxSAT solver, `MiniWalk`, performed the best. It solved 37 out of the 52 unsatisfiable SAT-Race 2008 instances, i.e., 71%, to optimality.[3] In contast, all other solvers could solve to optimality somewhere between 7 and 12 instances, i.e., only 13%-21%. Finally, 29 out of the 52 instances, i.e., 56%, were solved by `MiniWalk` in under one minute, highlighting the efficiency of the hybrid method.

5 Further Insights: Hybrid Search Pattern

We now explore a little deeper into the search behavior of `MiniWalk` and contrast it with the local search heuristics (`adaptg2wsat+p` and `saps` heuristics). Figure 1 shows a comparison of the behavior for the babic-dspam-vc973 instance from our suite, with x-axis showing the time elapsed since the solver started and the y-axis the number of unsatisfied clauses at a given time (log-scale). The instance was chosen because it highlights some of the key features of the search methods. It is solved to optimality (one unsatisfied clause) by `MiniWalk` within a few hours, although the data shown in the plot has it come down to two unsatisfied clauses. The three curves that level-off represent, in descending order, `Walksat`, `saps`, and `adaptg2wsat+p`. The remaining curve with steep drops depicts `MiniWalk`. While the local search algorithms initially descent rapidly and then stabilize at around 1,000 unsatisfied clauses with some natural noise, the hybrid method stays relatively high during the entire search (as high as the unguided `Walksat`, nearly 35,000 unsatisfied clauses), with occasional but extremely steep drops into promising regions. These regions are exactly where the best solutions are found, thanks to the DPLL guidance. While the local search often gets stuck in a plateau, the hybrid method keeps trying new promising regions as the DPLL search continues its systematic exploration. Even DPLL does not make very informed choices at the beginning of its search, but due to restarts, which are an integral part of the state-of-the-art DPLL solvers, these decisions are revised and a promising region is found relatively quickly. (A similar plot for an instance on which `MiniWalk` finishes much faster may be found in the Appendix.)

Figure 2 shows a more detailed look at the internals of the hybrid solver. The y-axis shows a comparison of the depth of the DPLL search (number of choice points, scaled down by a constant factor) and the quality of the current assignment found by the solver (number of unsatisfied clauses). The x-axis is again time and the instance is, as before, babic-dspam-vc973 (although the data is from a different, shorter run than in Figure 1). The curves show some amount of correlation between the DPLL depth and the quality of solution, suggesting that indeed when a brand new region is explored by the DPLL search, a good quality solution can be discovered.

Relative speed of the two solvers plays an important role in the process. The slower the DPLL search, the more time the local search has to explore given regions, but on the other hand, the whole search space is traversed more

[3] As noted earlier, we have been able to improve these numbers to 51 out of the 52 instances having only one unsatisfied clause in the optimal solution [Kroc *et al.*, 2009].

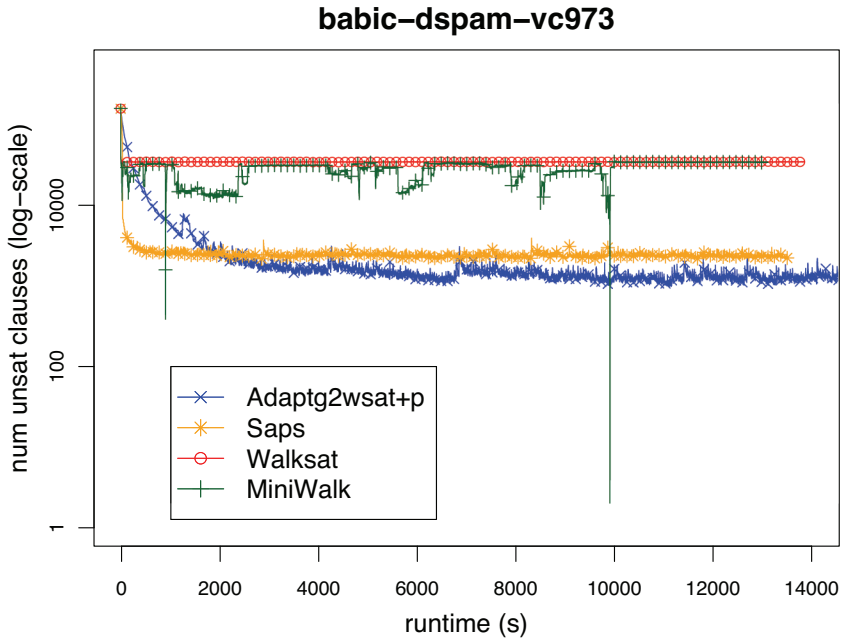

Figure 1: Qualitative search behavior in terms of the number of unsatisfied clauses (y-axis, log scale) as runtime progresses (x-axis). Both state-of-the-art pure local search methods, unguided `Walksat` and `MiniWalk` are shown. Note the deep drops of `MiniWalk`, which distinguish it from the other techniques. Instance: babic-dspam-vc973.cnf.

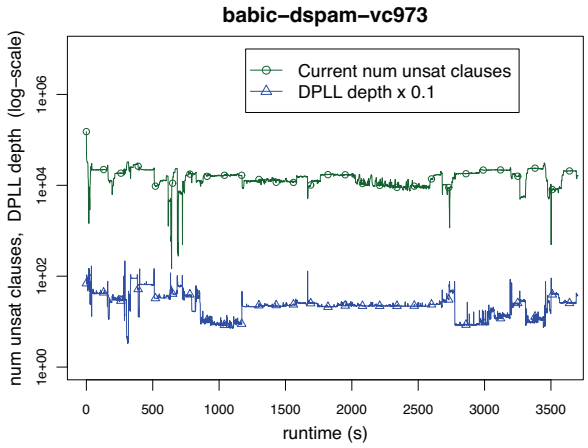

Figure 2: Number of unsatisfied clauses in `MiniWalk` and DPLL search depth vs. time in babic-dspam-vc973.cnf.

slowly. We experimented with this effect by artificially slowing down `MiniSat`, and running the resulting solver on the instances. There appeared to be no obvious setting that is clearly better than all others, and we used the default speed of `Minisat` in the results presented here. It is the case, however, that some instances were solved better when `MiniSat` was slowed down. Other parameters of the DPLL solver (the restart frequency in particular) have also mixed effect on the results. We left these parameters to `MiniSat`'s default setting in our experiments.

6 Conclusion

This paper presents a novel approach to solving MaxSAT instances that combines strengths of both DPLL and local search SAT solvers. The proposed hybrid solver can be easily constructed from any pair of such solvers, and we found that `Minisat` and `Walksat` work exceptionally well, solving many hard problems to optimality that were not solved by any other state-of-the-art technique. The ideas presented here can be explored also for other related problems, such as weighted and partial MaxSAT. There is a clear potential of using the DPLL search to satisfy all hard constraints (or those with large weight) and leaving the "fine tuning" of the lower-weight constraints to the local search. Finally, further exploration of information flow in the other direction—from local search to DPLL—beyond what is known in the context of SAT (e.g., [Mazure *et al.*, 1998]) is left as future work.

Acknowledgments

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

References

[Argelich *et al.*, 2007] J. Argelich, C. M. Li, F. Manyà, and J. Planes. Max-SAT Evaluation 2007, May 2007. http://www.maxsat07.udl.es.

[Biere *et al.*, 2009] A. Biere, M. J. H. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability*, volume 185 of

Frontiers in Artificial Intelligence and Applications. IOS Press, Feb 2009.

[Davis and Putnam, 1960] M. Davis and H. Putnam. A computing procedure for quantification theory. *CACM*, 7:201–215, 1960.

[Davis *et al.*, 1962] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *CACM*, 5:394–397, 1962.

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

[Grégoire *et al.*, 2007a] E. Grégoire, B. Mazure, and C. Piette. Boosting a complete technique to find MSS and MUS thanks to a local search oracle. In *20th IJCAI*, pg. 2300–2305, Hyderabad, India, Jan 2007.

[Grégoire *et al.*, 2007b] E. Grégoire, B. Mazure, and C. Piette. MUST: Provide a finer-grained explanation of unsatisfiability. In *13th CP*, volume 4741 of *LNCS*, pg. 317–331, Providence, RI, Sep 2007.

[Hirsch and Kojevnikov, 2005] E. A. Hirsch and A. Kojevnikov. UnitWalk: A new SAT solver that uses local search guided by unit clause elimination. *Annals Math. and AI*, 43(1):91–111, 2005.

[Hoos and Stützle, 2004] H. H. Hoos and T. Stützle. *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann, San Francisco, CA, 2004.

[Hutter *et al.*, 2002] F. Hutter, D. A. D. Tompkins, and H. H. Hoos. Scaling and probabilistic smoothing: Efficient dynamic local search for SAT. In *8th CP*, volume 2470 of *LNCS*, pg. 233–248, Ithaca, NY, Sep 2002.

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

[Kroc *et al.*, 2009] L. Kroc, A. Sabharwal, and B. Selman. Relaxed DPLL search for MaxSAT. In *12th SAT*, Swansea, Wales, U.K., Jun 2009. To appear.

[Li *et al.*, 2007a] C. M. Li, F. Manyà, and J. Planes. New inference rules for Max-SAT. *JAIR*, 30:321–359, 2007.

[Li *et al.*, 2007b] C. M. Li, W. Wei, and H. Zhang. Combining adaptive noise and look-ahead in local search for SAT. In *10th SAT*, volume 4501 of *LNCS*, pg. 121–133, Lisbon, Portugal, May 2007.

[Marques-Silva and Manquinho, 2008] J. P. Marques-Silva and V. M. Manquinho. Towards more effective unsatisfiability-based maximum satisfiability algorithms. In *11th SAT*, volume 4996 of *LNCS*, pg. 225–230, Guangzhou, China, May 2008.

[Mazure *et al.*, 1998] B. Mazure, L. Sais, and E. Grégoire. Boosting complete techniques thanks to local search methods. *Annals Math. and AI*, 22(3-4):319–331, 1998.

[Prestwich, 2007] S. D. Prestwich. Variable dependency in local search: Prevention is better than cure. In *10th SAT*, volume 4501 of *LNCS*, pg. 107–120, Lisbon, Portugal, May 2007.

[Selman *et al.*, 1996] B. Selman, H. Kautz, and B. Cohen. Local search strategies for satisfiability testing. In D. S. Johnson and M. A. Trick, editors, *Cliques, Coloring and Satisfiability: the Second DIMACS Implementation Challenge*, volume 26 of *DIMACS Series in DMTCS*, pg. 521–532. Amer. Math. Soc., 1996.

[Sinz (Organizer), 2008] C. Sinz (Organizer). SAT-race 2008, May 2008. http://www-sr.informatik.uni-tuebingen.de/sat-race-2008.

[Tompkins and Hoos, 2003] D. A. D. Tompkins and H. H. Hoos. Scaling and probabilistic smoothing: Dynamic local search for unweighted MAX-SAT. In *16th Canadian Conf. on AI*, volume 2671 of *LNCS*, pg. 145–159, Halifax, Canada, Jun 2003.

[Tompkins and Hoos, 2004] D. A. D. Tompkins and H. H. Hoos. UBCSAT: An implementation and experimentation environment for SLS algorithms for SAT and MAX-SAT. In *7th SAT*, Vancouver, BC, May 2004. Solver description.

A Appendix

A.1 Using Shared Memory Array

We use the IPC shared memory framework in both systematic and local search solver as follows (a C code snippet):

```
#include <sys/shm.h>
int *sharedMem = NULL;
long shmKey = 0x11112222;   //any unique id
int shmId = shmget( shmKey, \
   sizeof(int)*(nVars+1), IPC_CREAT|0600 );
sharedMem = (int*)shmat( shmId, NULL, 0 );
```

This memory can now be written to (within DPLL) and read from (within local search) by `sharedMem[verId]`. At the end of each program, `shmctl(shmid, IPC_RMID, 0)` frees up the shared memory.

A.2 Additional Instance Analysis

Figure 3 shows the comparison between local search and `MiniWalk` (the bottom-most curve, finishing early), this time for the ibm-2002-31_1r3-k30 instance. This instance is one where `MiniWalk` is clearly the best approach and very quickly discovers an assignment with only one unsatisfied clause (and thus an optimal MaxSAT solution for this unsatisfiable instance). In fact, this is the more common mode of operation of the hybrid solver: it is able to very quickly find assignments that are optimal or close to optimal.

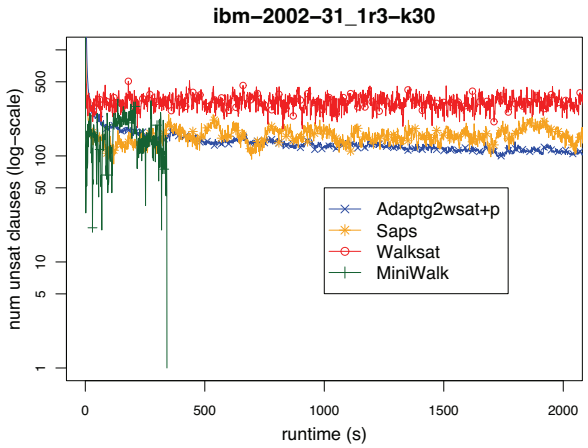

Figure 3: Qualitative search behavior in terms of the number of unsatisfied clauses (y-axis, log scale) vs. runtime (x-axis). Both state-of-the-art pure local search methods, unguided `Walksat` and `MiniWalk` are shown. Note the unique steep drops of `MiniWalk`. Instance: ibm-2002-31_1r3-k30.cnf.