Report Number: WUCSE-2004-16

2004-02-15

An Efficient Algorithm for Maximum Boolean Satisfiability Based on Unit Propagation, Linear Programming, and Dynamic Weighting

Zhao Xing and Weixiong Zhang

Maximum Boolean satisfiability (max-SAT) is the optimization counterpart of Boolean satisfiability (SAT), in which a variable assignment is sought to satisfy the maximum number of clauses in a logical formula. A branch-and-bound algorithm based on the Davis-Putnam-Logemann-Loveland procedure (DPLL) is one of the most efficient complete algorithms for solving max-SAT. In this paper, We propose and investigate a number of new strategies for max-SAT. Our first strategy is a set of unit propagation rules for max-SAT. As unit propaga-tion is a very efficient strategy for SAT, we show that it can be extended to max-SAT, and can greatly improve... **Read complete abstract on page 2.**

An Efficient Algorithm for Maximum Boolean Satisfiability Based on Unit Propagation, Linear Programming, and Dynamic Weighting

Zhao Xing and Weixiong Zhang

Complete Abstract:

Maximum Boolean satisfiability (max-SAT) is the optimization counterpart of Boolean satisfiability (SAT), in which a variable assignment is sought to satisfy the maximum number of clauses in a logical formula. A branch-and-bound algorithm based on the Davis-Putnam-Logemann-Loveland procedure (DPLL) is one of the most efficient complete algorithms for solving max-SAT. In this paper, We propose and investigate a number of new strategies for max-SAT. Our first strategy is a set of unit propagation rules for max-SAT. As unit propaga-tion is a very efficient strategy for SAT, we show that it can be extended to max-SAT, and can greatly improve the performance of an extended DPLL-based algorithm. Our second strategy is an effective lookahead heuristic based on linear programming. We show that the LP heuristic can be made effective as the number of clauses increases. Our third strategy is a dynamic-weight variable ordering, which is based on a thorough analysis of two well-known existing branching rules. Based on the analysis of these strategies, we develop an integrated, constrainedness-sensitive max-SAT solver that is able to dynamically adjust strategies ac-cording to problem characteristics. Our experimental results on random max-SAT and some instances from the SATLIB show that our new solver outperforms most of the existing com-plete max-SAT solvers, with orders of magnitude of improvement in many cases.

Department of Computer Science & Engineering

Washington University in St. Louis
School of Engineering & Applied Science

2004-16

An Efficient Algorithm for Maximum Boolean Satisfiability Based on Unit Propagation, Linear Programming, and Dynamic Weighting

Authors:

April 5, 2004

Abstract: Maximum Boolean satisfiability (max-SAT) is the optimization counterpart of Boolean satisfiability (SAT), in which a variable assignment is sought to satisfy the maximum number of clauses in a logical formula. A branch-and-bound algorithm based on the Davis-Putnam-Logemann-Loveland procedure (DPLL) is one of the most efficient complete algorithms for solving max-SAT. In this paper, We propose and investigate a number of new strategies for max-SAT. Our first strategy is a set of unit propagation rules for max-SAT. As unit propagation is a very efficient strategy for SAT, we show that it can be extended to max-SAT, and can greatly improve the performance of an extended DPLL-based algorithm. Our second strategy is an effective lookahead heuristic based on linear programming. We show that the LP heuristic can be made effective as the number of clauses increases. Our third strategy is a dynamic-weight variable ordering, which is based on a thorough analysis of two well-known existing branching rules. Based on the analysis of these strategies, we

An Efficient Algorithm for Maximum Boolean Satisfiability Based on Unit Propagation, Linear Programming, and Dynamic Weighting

Zhao Xing and *Weixiong Zhang*
Department of Computer Science and Engineering
Washington University in St. Louis
St. Louis, MO 63130, USA
Email: {zx2,zhang}@cs.wustl.edu

February 15, 2004

Abstract

Maximum Boolean satisfiability (max-SAT) is the optimization counterpart of Boolean satisfiability (SAT), in which a variable assignment is sought to satisfy the maximum number of clauses in a logical formula. A branch-and-bound algorithm based on the Davis-Putnam-Logemann-Loveland procedure (DPLL) is one of the most efficient complete algorithms for solving max-SAT. In this paper, We propose and investigate a number of new strategies for max-SAT. Our first strategy is a set of unit propagation rules for max-SAT. As unit propagation is a very efficient strategy for SAT, we show that it can be extended to max-SAT, and can greatly improve the performance of an extended DPLL-based algorithm. Our second strategy is an effective lookahead heuristic based on linear programming. We show that the LP heuristic can be made effective as the number of clauses increases. Our third strategy is a dynamic-weight variable ordering, which is based on a thorough analysis of two well-known existing branching rules. Based on the analysis of these strategies, we develop an integrated, constrainedness-sensitive max-SAT solver that is able to dynamically adjust strategies according to problem characteristics. Our experimental results on random max-SAT and some instances from the SATLIB show that our new solver outperforms most of the existing complete max-SAT solvers, with orders of magnitude of improvement in many cases.

1 Introduction and Overview

Boolean satisfiability (SAT) is an archetypical decision problem in artificial intelligence, logic, theory of computation, and many related areas. SAT with more than two literals (variables or their negations) per clause is NP-complete [4, 11]. Maximum Boolean satisfiability (max-SAT) is the optimization counterpart of SAT, which maximizes the number of satisfied clauses. Max-SAT is more general than SAT; the solution to max-SAT can be used to answer the question of its decision counterpart, but not necessarily vice versa. Therefore, max-SAT is more difficult to solve than SAT as well. In contrast to SAT, max-SAT is NP-hard [11] even when there are at most two literals per clause. Similar to SAT, max-SAT also has many real-world applications, such as scheduling, configuration problems, probabilistic reasoning, and pattern recognition [10, 13].

It has been shown that a branch-and-bound (BnB) algorithm based on the Davis-Putnam-Logemann-Loveland procedure (DPLL) [30] is one of the most efficient complete algorithms for max-SAT. Many efforts have been devoted to improve the performance of such a BnB-based DPLL algorithm for max-SAT by combining the techniques previously developed for SAT [30, 21, 3] and many methods used in operations research (OR), such as integer linear programming (ILP) and cutting plane methods [16, 21, 8]. However, these efforts have enjoyed a limited success, especially on large, complex problems. In particular, the current OR-based approaches are more effective than the DPLL-based algorithms only on max-2-SAT [21]. On the other hand, even though a BnB-based DPLL algorithm remains one of the most competitive algorithms for max-SAT, it can handle relatively small problems with moderate degrees of constrainedness.

Therefore, despite the previous efforts, much work is still needed for developing efficient algorithms for max-SAT, and special care is required to extend the techniques previously developed for SAT. In principle, most techniques developed for SAT can be extended to Max-SAT [10, 13, 30]. However, these techniques for SAT take advantage of the fact that SAT is a decision problem, so that a search avenue can be abandoned as soon as a constraint violation becomes evident. This fact has been properly captured in the unit resolution or propagation methods and different variable orderings used by the DPLL algorithm and its variants. In contrast, the study of unit propagation methods and variable orderings for max-SAT is limited, expect the work of [10, 30]. It is important to note that max-SAT has its own intrinsic features that are remarkably different from its decision counterpart. Many existing techniques for SAT must be carefully reconsidered when being applied to max-SAT. Overall, it is much harder to develop an effective and efficient algorithm for max-SAT than that for SAT, and the research of developing efficient max-SAT solver deserves much attention, due to the generality and importance of the problem.

Aiming to solve difficult max-SAT problems, we develop an efficient exact max-SAT algorithm based on the DPLL algorithm for SAT. Our algorithm has three ingredients, which can be viewed as novel extensions to the main ideas behind the existing methods for SAT. The first is a set of unit propagation rules for max-SAT, which are important because many existing unit propagation rules for max-SAT are not effective. The lack of strong unit propagation rules in the existing max-SAT solvers is perhaps one of the main culprits of their poor performance. We propose an effective new unit propagation rule based on an integer nonlinear programming formulation of max-SAT; and present a set of unit propagation rules whose combined effect is very prominent when applied to max-SAT.

The second element of our max-SAT algorithm is an effective lookahead heuristic for estimating the minimum number of clauses unsatisfiable at a node during the search. Our heuristic is based on linear programming (LP) [14]. This is a remarkable contribution, as it is perhaps the first successful application of LP to max-SAT, despite of similar (but not successful) previous efforts of applying integer LP (ILP) to max-SAT [16, 21, 8].

The third ingredient is a dynamic-weight variable ordering. We experimentally analyze two popular SAT variable orderings, i.e., the Mom's rule [5, 22] and two-side Jeroslow-Wang rule [17], on max-SAT. Our empirical study shows that these branching rules are constrainedness-sensitive. No single branching rule is dominant over the other; and different branching rules should be adopted according to different problem characteristics. To address this problem, we propose a dynamic-weight variable ordering, which can switch its branching rule from the Mom's rule to two-sided Jeroslow-Wang rule as constrainedness increases.

The paper is organized as follows, we first discuss max-SAT and describe two mathematical formulations of max-SAT in section 2. In section 3, we extend the DPLL algorithm for SAT to max-SAT, and discuss various factors that affect its performance, including initial upper bound, value ordering, lower bound from unit clauses, and two existing variable ordering rules. In section 4, we present a set of four unit propagation rules for max-SAT; and prove their correctness. In section 5, we develop a lower bound heuristic based on linear programming, and show that LP-heuristic is effective on highly constrained problem instances. In section 6, we propose a dynamic-weight variable ordering, which can adjust the weights of its branching rule as constrainedness increases. We present experimental results of our new strategies, and then describe an integrated max-SAT solver that combines all our new strategies in section 7. We also systematically compare our new solver with the existing max-SAT solvers in section 7. Finally, we discuss some related work in section 8, and give conclusions in section 9.

2 Formulations of Maximum Satisfiability

A satisfiability problem (SAT) is a Boolean formula involving a set of Boolean variables and a conjunction of a set of disjunctive clauses of literals, which are variables and their negations. A clause is satisfied if at least one of its literals takes value T, and a formula is satisfied if all the clauses are satisfied. The conjunction defines constraints on the possible combinations of variable assignments. SAT is to find a variable assignment that satisfies all the clauses. Specially, 3-SAT is SAT where each clause has three literals. When there exists no variable assignment to satisfy all clauses, it is required to find an assignment such that the total number of satisfied clauses is maximized [10]. This is maximum satisfiability, maximum SAT, or max-SAT.

2.1 Linear programming

max-SAT can be formulated by an integer linear program (ILP) [21] or pseudo-Boolean formula [31, 8]. We map a Boolean variable v to an integer variable x that takes value 1 when v is True and 0 when it is False. We then map $\overline{v}$ to $1 - x$. Therefore, we have $x = 1$ when $v = T$; and $1 - x = 0$ when $\overline{v} = F$. With these mappings, we can then formulate a clause as a linear inequality. For example, clause $(v_1 \vee \overline{v}_2 \vee v_3)$ can be mapped to $x_1 + (1 - x_2) + x_3 \geq 1$. Here, the inequality means that the clause must be satisfied in order for the left side of the inequality to have a value no less than one.

However, a clause in a max-SAT may not be satisfied so that its corresponding inequality may be violated. To address this issue, we introduce an auxiliary integer variable w to the left side of a mapped inequality. Variable $w = 1$ if the corresponding clause is unsatisfied, making the inequality valid; otherwise, $w = 0$. Since the objective is to minimize the number of violated clauses, it is then to minimize the number of auxiliary variables that are forced to take value 1. To be concrete, $(v_1 \vee \overline{v}_2 \vee v_3), and (v_2 \vee \overline{v}_4)$ can be written as an ILP of minimizing $W = w_1 + w_2$, subject to

$$\begin{cases} x_1 & +(1-x_2) & +x_3 & & & +w_1 & \geq 1 \\ & & x_2 & & +(1-x_4) & +w_2 & \geq 1 \end{cases} \tag{1}$$

The linear 0-1 programming formulation of max-SAT suggests that this problem can be solved by integer linear programming (ILP). However, ILP is still NP-hard [11]. Furthermore, as shown in [21], except for max-2-SAT, ILP for max-k-SAT problems ($k \geq 3$) is inferior to a simply extended DPLL algorithm for max-SAT.

2.2 Nonlinear programming

The ILP formulation of max-SAT can be extended to a nonlinear programming formulation by applying the inclusion-exclusion principle [23] so that inequalities in ILP formulation can be turned into equalities in integer nonlinear formulation. Here we introduce an integer expression to represent a *literal*. For example, for $(v_1 \vee \overline{v}_2 \vee v_3)$, we introduce integer expressions $x_1, 1 - x_2$ and x_3 for the literals $v_1, \overline{v}_2$ and v_3. Such an integer expression takes value 1 if its corresponding literal is set to true, or value 0 otherwise. Using the inclusion-exclusion principle, for the i-th 3-literal clause of a given formula $(v_1 \vee \overline{v}_2 \vee v_3)$, we then write a nonlinear equation,

$$x_1 + 1 - x_2 + x_3 - x_1(1 - x_2) - x_1x_3 - (1 - x_2)x_3 + x_1(1 - x_2)x_3 + w_i = 1 \qquad (2)$$

It is important to note that $f = x_1 + 1 - x_2 + x_3 - x_1(1 - x_2) - x_1x_3 - (1 - x_2)x_3 + x_1(1 - x_2)x_3$ can take either value 1 or 0. Specifically, $f = 0$ if no literal in the clause is set to true, or $f = 1$ otherwise. As in the ILP formulation, we introduce auxiliary variables, w_i's, to count for unsatisfied clauses. Here, $w_i = 1$ if $f = 0$, and $w_i = 0$ if $f = 1$. For a binary clause $(v_1 \vee v_3)$ or a unit clause $(\overline{v}_2)$, the corresponding nonlinear equation becomes

$$x_1 + x_3 - x_1x_3 + w_i = 1 \qquad \text{or} \qquad 1 - x_2 + w_i = 1 \qquad (3)$$

The objective of this nonlinear integer program is the same as ILP, i.e., minimizing $W = \sum_{i=1}^{m} w_i$, where an auxiliary variable w_i is introduced to the i-th clause of a formula of m clauses.

3 DPLL Algorithm for Maximum Satisfiability

The Davis-Putnam–Logemann-Loveland (DPLL) [7] algorithm for SAT is a backtracking algorithm that progressively instantiates one variable at a time in order to search for a satisfying variable assignment. In each step, the algorithm selects a variable and branches it off to two possible values, T and F. Whenever a clause is violated after setting a variable to T and F, the algorithm backtracks to a previous variable. The process continues until either a satisfying assignment is found or it can conclude that no such assignment exists.

DPLL for SAT can be extended to max-SAT using depth-first branch-and-bound (DFBnB). DFBnB is a special branch-and-bound that explores nodes in a depth-first order. DFBnB uses an upper bound α on the minimal number of clauses that cannot be satisfied, whose initial value can be infinity or the sub-optimal value generated by a local search algorithm. Starting at the root node, DFBnB always selects a recently generated node to examine next. If in the current node, all the variables have been instantiated, and the number of clauses violated (g value) is less than

the current upper bound α, α is revised to the g value; if some variables are still un-instantiated, and the g value got so far is greater than or equal to α, the current node is pruned.

For simplicity, here we point out the two main differences between DFBnB for max-SAT and backtracking for SAT. First, the upper bound α may not be zero for max-SAT. Therefore, backtracking for SAT can be viewed as a special case of DFBnB for max-SAT where $\alpha = 0$ throughout the search, disallowing any clause violation and resulting at a much reduced search cost. In fact, this special condition of $\alpha = 0$ makes unit propagation (discussed in Section 4) very effective for SAT. Second, DFBnB for max-SAT can abandon a node during the search only if the g value plus a lower bound on the minimal number of clauses that must be violated in the remaining clauses (h value in A* algorithm) at the node exceeds the upper bound α. This indicates that max-SAT becomes more difficult when the constrainedness increases, causing more clauses unsatisfied and a higher upper bound α. This also implies that one method to reduce the search cost of DFBnB is to estimate the number of clauses that cannot be satisfied among the remaining clauses at a node (h value), so as to increase the possibility of pruning the node if it indeed cannot lead to a better variable assignment. This last observation motivated the work on LP-based heuristic (discussed in Section 5).

3.1 Initial upper bound

One way to improve DPLL on max-SAT is to obtain a good initial upper bound α. The smaller the initial α, the more nodes will be pruned. Ideally, the initial α should be set to the cost of an optimal solution, which is typically unknown before the problem is solved. An initial upper bound α can be obtained by an approximation algorithm. A local search algorithm such as WalkSAT [24, 28], one of the best local search algorithms for SAT, is a good choice. In our experiments in Section 7, we apply WalkSAT multiple times to improve the quality of the initial upper bound. Such a combination of local search and systematic search is called a two-phase algorithm [3].

3.2 Lower bounds from unit clauses

Another way to improve DPLL on max-SAT is to compute a lower bound on the minimal number of clauses that cannot be satisfied at the current node of the search. One simple lower bound uses only unit clauses. At a node during the search, if a literal and its negation appear in a set of unit clauses, not every unit clause in this set can be satisfied. The minimal number of such conflicting positive and negative literals in all unit clauses is the minimal number of clauses to be violated, regardless how this variable is instantiated. The sum of all such values for all un-instantiated

variables gives a lower bound on the minimal number of clauses to be violated. It has been shown that this simple lower bound can significantly improve the performance of the DPLL algorithm for max-SAT [10]. We adopt this lower bound in our implementation of the DPLL algorithm.

3.3 Variable ordering

Each recursive call of DPLL may involve a choice of a variable for instantiation. Strategies for making such choices are referred to as *branching rules*. The performance of the DPLL algorithm is greatly affected by the branching rule used. A well-known rule for 3-SAT is two-sided Jeroslow-Wang (J-W) rule [17]. Let $\{C_1, C_2, \cdots, C_m\}$ be the set of clauses to be satisfied. Two sided Jeroslow-Wang rule selects a variable v that maximizes $J(v) + J(\overline{v})$ over all un-instantiated variables, where

$$J(v) = \sum_{v \in C_i} 2^{-n_i} \tag{4}$$

and n_i is the number of literals in C_i.

This branching rule is based on the intuition that shorter clauses are more important than longer ones. It gives the variables that appear in shorter clauses higher weights so that a variable appearing more often in unit clauses is more likely to be selected. It also assumes 4:2:1 to be the ratio of weights for variables in unit, binary and three-literal clauses. (Note that the idea of progressively halving the weighting factors was used by Johnson [20] thirty years earlier in an approximation algorithm for max-SAT.) We call a rule giving different weights to variables in clauses of different sizes a *weighted variable ordering* or a *weighted branching rule*

Weighted variable ordering has been shown to be very effective for 3-SAT [9, 22]. Moreover, experimental results support the scheme of giving the highest weight to variables in the shortest clauses [9, 22]. This scheme has led to another popular SAT heuristic, the Mom's rule, which branches next on the variable having the maximum occurrence in the clauses of minimum size [5, 22]. It has been shown that on 3-SAT, the Mom's rule is better than two-sided Jeroslow-wang rule [9, 22]. In [9, 22], Mom's heuristic was represented as a formula for weighted variable ordering where a clause of length i has a weight that is 5 times as large as the weight of a clause of length $i+1$ instead of 2, namely, the Mom's heuristic has weight ratio of 25:5:1, which is different from that of 4:2:1 in two-sided Jeroslow-wang rule.

3.4 Value ordering

Value ordering is another important element for performance. In a node of a DFBnB search tree explored by the extended DPLL algorithm, the two possible instantiations of a branching

variable need to be explored. Generally, the better the value ordering strategy, the sooner the search process can reach a better solution if it exists, reducing the upper bound more quickly. However, the effect of value ordering is nearly dominated by a good initial upper bound strategy, especially one that is able to provide nearly-optimal value. In our extended DPLL algorithm for max-SAT, because we apply an efficient local search to get a good initial upper bound, we do not use any value ordering strategy, ie, we use a fixed value ordering, first exploring a variable by its true value, and then by its false value.

4 Unit Propagation

A clause with only one literal is call a unit clause. Unit propagation is the most powerful strategy for SAT, and the central piece of a DPLL-based SAT solver. Unit propagation forces the variable in a unit clause to take the value that makes the clause be satisfied immediately and ignores the other value completely. Furthermore, all the clauses containing the literal equal to the forced value of the variable can be removed (satisfied) and the negated literal can also be eliminated from all clauses. The result is a simplified formula. More importantly, the power of unit propagation largely comes from its chaining reaction, i.e., setting a variable in a unit clause to a fixed value may subsequently generate more unit clauses, which can further simplify the formula at hand. Conversely, if two clauses having opposite literals, e.g., (v) and $(\overline{v})$, appear in the current formula, the formula is obviously unsatisfiable and the current search node can be abandoned.

In max-SAT, a clause may not be satisfied at all. Such an unsatisfiable clause may be simplified to a unit clause during the search. Therefore, we cannot restrict the literal in a unit clause to value T, but have to consider setting it to F as well, as long as doing so does not cause the number of violated clauses to exceed the current upper bound α. Therefore, in principle, unit propagation for SAT in its pure sense does not apply to max-SAT.

Nevertheless, the idea of unit propagation can be extended to max-SAT. For a max-k-SAT problem instance, consider a node N of a DFBnB search tree explored by an extended DPLL algorithm, and an un-instantiated variable v in N. Let g be the number of clauses that have been violated at N. Let $p_i(v)$ and $n_i(v)$ be the numbers of *i*-literal clauses in N which have v as a positive and negative literal, respectively. We then have the following unit propagation (UP) rules.

4.1 UP1 : Pure literal rule

- Pure literal rule: *If* $\sum_{i=1}^{k} n_i(v) = 0$*, force* $v = T$ *and ignore* $v = F$*; If* $\sum_{i=1}^{k} p_i(v) = 0$*, force* $v = F$ *and ignore* $v = T$*.*

The pure literal rule is also known as *monotone variable fixing* [21]. Although an algorithm using this rule can only get a very moderate improvement on SAT [25], experiments done by Wallace showed that improvement of the pure literal rule is remarkable for max-2-SAT [29]. We include this rule in our extended DPLL algorithm for max-SAT.

4.2 UP2 : Upper bound rule

- Upper bound rule: *If* $p_1(v) + g \geq \alpha$*, force* $v = T$ *and ignore* $v = F$*; If* $n_1(v) + g \geq \alpha$*, force* $v = F$ *and ignore* $v = T$*; If both conditions hold, prune the current node.*

The upper bound rule is self evident. When setting $v = F$, at least $p_1(v) + g$ clauses will be violated, making it unfavorable comparing to the best variable assignment found so far.

4.3 UP3 : Dominating unit-clause rule

- Dominating unit-clause rule: *If* $p_1(v) \geq \sum_{i=1}^{k} n_i(v)$*, set* $v = T$ *and ignore* $v = F$*; If* $n_1(v) \geq \sum_{i=1}^{k} p_i(v)$*, set* $v = F$ *and ignore* $v = T$*; If both conditions hold, i.e.,* $p_1(v) = n_1(v)$*, set* $v = T$ *or* $v = F$ *and ignore the other value.*

The dominating unit-clause rule was first proposed by Niedermeier in [27]. It has been applied to max-2-SAT in [32]. The dominating unit-clause rule is also self-evident, because setting $v = F$ causes p_1 clauses to be violated immediately, which is no better than violating $\sum_{i=1}^{k} n_1(v)$ clauses if $v = T$. In the rest of this subsection, we give a formal proof to this rule.

Proof: Our proof starts with the *nonlinear formulation* of max-SAT introduced in section 2. Due to space limitations, we only consider max-2-SAT in this proof. Specifically, we only prove that when $p_1(v) \geq n_1(v) + n_2(v)$, setting $v = T$ and ignoring $v = F$ will not miss an optimal solution. The case for $n_1(v) \geq p_1(v) + p_2(v)$ is symmetric. The proof is essentially the same but lengthy for max-k-SAT($k \geq 3$).

Let C_1 and C_2 be the sets of unit and binary clauses, and $C_k(v_j)$ (or $C_k(\overline{v}_j)$) the set of k-literal clauses that contain literal v_j (or $\overline{v}_j$). For simplicity, we also use i to denote the i-th clause. Summing up the nonlinear equalities for all clauses, we can write,

$$\sum_{i=1}^{m} w_i = m \quad - \quad \sum_{i \in C_2} (l_{i1} + l_{i2} - l_{i1} l_{i2}) - \sum_{i \in C_1} l_{i1} \tag{5}$$

Where l_{i1}, l_{i2} are literals in the i-th clause. To minimize the objective $W = \sum_{i=1}^{m} w_i$, consider the items in (5) that contain variable v_j, which may be in positive literal, corresponding to integer variable x_j, or negative literal, corresponding to $1 - x_j$. v_j may be in unit and binary clauses. We now consider these clauses with v_j in turn.

- v_j is in positive literal (corresponding to integer variable x_j) and in unit clauses

$$-\sum_{i \in C_1(v_j)} l_{i1} = -\sum_{i \in C_1(v_j)} x_j = -p_1(v_j)x_j \tag{6}$$

- v_j is in positive literal (corresponding to integer variable x_j) and in binary clauses

$$\begin{aligned} -\sum_{i \in C_2(v_j)} (l_{i1} + l_{i2} - l_{i1}l_{i2}) &= -\sum_{i \in C_2(v_j)} (x_j + l_{i2} - x_j l_{i2}) \\ &= -\sum_{i \in C_2(v_j)} l_{i2} + \sum_{i \in C_1(v_j)} l_{i2}x_j - p_2(v_j)x_j \end{aligned} \tag{7}$$

- For the other three cases where v_j is in negative literal (corresponding to $1 - x_j$), we have

$$-n_1(v_j) + n_1(v_j)x_j \tag{8}$$

$$-n_2(v_j) - \sum_{i \in C_2(\overline{v}_j)} l_{i2}x_j + n_2(v_j)x_j \tag{9}$$

We now focus on the coefficient F_{x_j} of integer variable x_j. Summing up (6) to (9), we have

$$F_{x_j} = n_1(x_j) + n_2(x_j) - p_1(x_j) - p_2(x_j) + \sum_{i \in C_2(v_j)} l_{i2} - \sum_{i \in C_2(\overline{v}_j)} l_{i2}. \tag{10}$$

Because $\sum_{i \in C_2(v_j)} l_{i2} \leq p_2(v_j)$, and $\sum_{i \in C_2(\overline{v_j})} l_{i2} \geq -n_2(x_j)$ we then have

$$n_1(v_j) - p_1(v_j) - p_2(v_j) \leq F_{x_j} \leq n_1(v_j) + n_2(v_j) - p_1(v_j) \tag{11}$$

If $p_1 \geq n_1(v_j) + n_2(v_j)$, F_{x_j} can not be positive, thus to minimize the objective W, x_j should take value 1, i.e., $v_j = T$. If $n_1(v_j) \geq p_1(v_j) + p_2(v_j)$, F_{x_j} can not be negative, to minimize W, x_j should take value 0, i.e., $v_j = F$. This concludes the proof.

4.4 UP4 : Coefficient-determinant propagation rule

Note that we have examined the coefficients of un-instantiated variables in the proof of dominating unit-clause rule. In fact, variable coefficients can be directly exploited in a systematic way, which we refer to as the coefficient-determinant propagation rule. For simplicity, we only consider this rule for max-2-SAT in the following discussion. The same idea applies to max-3-SAT, while it is technically more involved to prove and implement it.

First of all, we represent a max-2-SAT problem instance by a nonlinear formula f in such a way that the final formula only contains variables $x_i(1 \leq i \leq n)$ and auxiliary variables $w_i(1 \leq i \leq m)$.

$$f = \sum_{i=1}^{m} w_i = c \quad + \quad \sum_{i=1}^{n} (coef_i x_i) + \sum_{1 \leq i,j \leq n, i \neq j} (coef_{i,j} x_i x_j) \tag{12}$$

Where *c* is a constant, $coef_{i,j}$ is the coefficient of item $x_i x_j$, and $coef_i$ is the coefficient of item x_i. We now derive the coefficient for variable x_i. We denote F_{x_i} as the coefficient of variable x_i, and $UB(x_i)$ and $LB(x_i)$as the upper bound and the lower bound for F_{x_i}, respectively. Let U be the set of variables that have been instantiated, and V the set of variables that are to be instantiated.

$$F_{x_i} = coef_i + \sum_{x_j \in U \cup V - \{x_i\}} coef_{i,j} x_j \tag{13}$$

Since some of the variables x_j in F_{x_i} are un-instantiated, most of the time during the search, F_{x_i} is not fixed. To this end, we rewrite F_{x_i} as follows:

$$\begin{aligned} F_{x_i} &= coef_i + \sum_{x_j \in U - \{x_i\}} coef_{i,j} x_j + \sum_{x_j \in V - \{x_i\}} coef_{i,j} x_j \\ &= coef_i + \sum_{x_j \in U - \{x_i\}} coef_{i,j} x_j + \sum_{x_j \in V - \{x_i\}, coef_{i,j} > 0} coef_{i,j} x_j + \sum_{x_j \in V - \{x_i\}, coef_{i,j} < 0} coef_{i,j} x_j \end{aligned}$$

Note that $\sum_{x_j \in V - \{x_i\}, coef_{i,j} > 0} coef_{i,j} x_j \geq 0$ and $\sum_{x_j \in V - \{x_i\}, coef_{i,j} < 0} coef_{i,j} x_j \leq 0$, therefore, we have

$$LB(x_i) \leq F_{x_i} \leq UB(x_i) \tag{14}$$

$$UB(x_i) = coef_i + \sum_{x_j \in U - \{x_i\}} coef_{i,j} x_j + \sum_{x_j \in V - \{x_i\}, coef_{i,j} > 0} coef_{i,j} \tag{15}$$

$$LB(x_i) = coef_i + \sum_{x_j \in U - \{x_i\}} coef_{i,j} x_j + \sum_{x_j \in V - \{x_i\}, coef_{i,j} < 0} coef_{i,j} \tag{16}$$

If $UB(x_i) \leq 0$, F_{x_i} can not be positive, thus to minimize the objective $W = \sum_{i=1}^{m} w_i$, x_i should take value 1, i.e., $v_i = T$. If $LB(x_i) \geq 0$, F_{x_i} can not be negative, to minimize W, x_i should take value 0, i.e., $v_i = F$. So, We therefore have the following rule.

- Coefficient-determinant propagation rule: *for each un-instantiated variable v_i and corresponding integer variable x_i, if $LB(x_i) \geq 0$, fix $v_i = F$; If $UB(x_i) \leq 0$, fix $v_i = T$, where $UB(x_i)$ and $LB(x_i)$ are defined in equation 15, 16; If both conditions hold, i.e., $UB(x_i) = LB(x_i)$, set $v_i = T$ or $v_i = F$ and ignore the other value.*

5 Linear Programming Based Lookahead Heuristic

As mentioned in Section 3, one effective way to improve DPLL for max-SAT is to develop a lookahead heuristic function h to estimate the number of clauses that cannot be satisfied at a node of a search tree. If the heuristic estimate h plus the number of clauses already violated, g, is greater than or equal to the current upper bound α, i.e., $g + h \geq \alpha$, the node can be pruned. One of the main contributions of this paper is such an effective lookahead heuristic for max-SAT.

The LP heuristic proposed in this section is very simple. To compute the h value of a node N, we apply the ILP formulation (Section 2) to N. However, rather than solving the remaining max-SAT at N by ILP, we apply linear programming (LP) to it instead. In other words, we do not restrict the mapped variables (e.g., $x_1, x_2, \cdots, w_1, \cdots$) to integers 0 or 1, rather allow them to be real values in $[0, 1]$. As a result, we only obtain an estimate of the actual solution cost of the ILP instance since LP is less restricted than ILP. By relaxing the problem to LP, we can obtain heuristic estimation with less computation.

In principle, applying a stronger heuristic function (i.e., LP-based lookahead heuristic in our case) can reduce the effective branching factor of a search. The complexity of extended DPLL algorithm is exponential in the number of constraints. Assuming that the effective branching factor of the extended DPLL algorithm is b and its average search depth for a given problem is d, we have $d = O(km)$, where k is a constant factor less than 1, and m is the number of constraints. The complexity of extended DPLL is then $T = O(b^d) = O(b^{km})$. Using LP-based heuristic, since more nodes can be pruned, the effective branching factor will be reduced to $b_{LP} < b$, and the total node expended will become $O(b_{LP}^{km})$. However, there is an overhead on the time of computing the LP-based lookahead heuristic. In our implementation of the heuristic, we used the CPLEX package [19] for LP, which runs in proportional to the number of constraints encoded in linear programs [19]. Therefore, the overhead for each LP call is $O(m)$. Thus, the overall time complexity of extended DPLL using LP-based heuristic is $T_{LP} = O(m)O(b_{LP}^{km})$. Combining these factors, $T_{LP} < T$ when the number of constraints m is large. This will be verified by our experiments in section 7. This also means that on under-constrained or modest-constrained problems, the overhead of LP makes the LP-based heuristic ineffective; and explains why it has been difficult to make LP effective on satisfiability problem instances.

However, the application of an LP-based heuristic needs to be handled with further care. Note that the solution to an LP relaxation problem at a node may have too many variables that take values in the middle of the range of $[0, 1]$, i.e., taking a value close to 1/2. Such "fractional" variables are troublesome in binary clauses, e.g., two such variables in a binary clause can take values slightly more than 1/2, forcing the auxiliary variable (w variable in the LP formulation,

Section 2) for the clause to take value 0, yielding no contribution to the overall heuristic function. Similar scenarios can occur to three-literal clauses. Fortunately, such situations will not occur in unit clauses because auxiliary variables can always contribute to the overall heuristic function even setting literals within unit clauses to "fractional" value. So, we only apply the LP-based heuristic function to nodes that have unit clauses. Moreover, during the search, unit clauses do not need to be eliminated, since the increase in expected lower bound from eliminating unit clauses has already been calculated exactly by applying LP heuristic, namely, if we apply LP heuristic to compute h, any expected gain on the g value from unit clauses has already been taken into account in the h value. All in all, DPLL+LP boosts the lower bound value even without increasing the g value.

6 Dynamic-Weight Variable Ordering

The best known existing variable ordering rules, e.g., the Mom's rule and two-sided Jeroslow-Wang rule described in section 3, are static, in that they fix the weights in their rules throughout search regardless of problem constrainedness. As we will see from our experiments discussed in the next section, these heuristic variable orderings are effective within different range of problem constrainedness.

Comparing to SAT, max-SAT can contain problem instances with various constrainedness. The existing variable orderings for SAT may not be effective for max-SAT. To address this problem, we propose a dynamic-weight variable ordering.

Dynamical-Weight Variable Ordering in each node of the DFBnB search tree explored by the extended DPLL algorithm, select a variable v that maximizes $J(v) + J(\overline{v})$ over all uninstantiated variables, where

$$J(v) = \sum_{v \in C_i} w(r)^{-n_i} \tag{17}$$

n_i is the number of literals in C_i, r is the clause / variable ratio, and

$$w(r) = \begin{cases} 5 & \text{if} \quad r < 6.3; \\ 6 - 3.33r & \text{if} \quad 6.3 \leq r \leq 7.2; \\ 2 & \text{if} \quad r > 7.2; \end{cases}$$

In the above function, 6.3 and 7.2 are two values determined empirically. Our experiments in section 7 show that in max-3SAT, when clause / variable ratio is smaller than or bigger than

these two threshold values, Mom's rule or two-sided Jeroslow-Wang rule perform well, respectively. Therefore, we dynamically switch weights from those close to the Mom's rule to those similar to two-sided Jeroslow-Wang rule as C/V ratio increases from 6.3 to 7.2, thus having good performance in all cases.

7 Experimental Evaluation and Applications

The combination of the above three strategies discussed in section 4, 5, and 6 leads to an integrated algorithm for max-SAT, which we shorthand as *MaxSolver*. In this section, we experimentally evaluate the performance of MaxSolver using various problem instances, including those from the SATLIB [18]. Without explicitly stated, all our experiments in this paper obeyed the following conditions: (1) an initial upper bound for each problem was computed by WalkSAT [28, 24] with 100 random restarts and 10,000 flips per try; (2) All experiments were run on PCs with Athlon 1.9 MHZ processor and 756 MB memory; (3) The LP solver we used to compute the h value was CPLEX 8.0 [19]. Note that we used dual-simplex algorithm in CPLEX, which optimizes the computation of the h value of the current node based on the existing solution to its parent node in the search tree. This feature can significantly reduce the number of iterations of the dual-simplex algorithm, particularly if the current problem is similar to the problem solved in the parent node.

We start with an investigation on the efficacy of the three improving strategies, and then compare our MaxSolver directly with some existing max-SAT algorithms that we are aware of and able to get source code from their authors.

7.1 Evaluation of new strategies

We first compared the average running time of the extended DPLL with and without unit propagations (or LP heuristic) in combinations of different branching rules. We ran three algorithms: DPLL, DPLL with different unit propagation (DPLL+UPs), and DPLL with LP heuristic (DPLL+LP), where UP and LP stand for unit propagation and LP heuristic, respectively. Each algorithm was tested with two branching rules, the Mom's rule and two-sided Jeroslow-Wang rule. Due to the complexity of implementation, we only applied our UP4 on max-2-SAT, although it can be implemented in the same way on max-3-SAT.

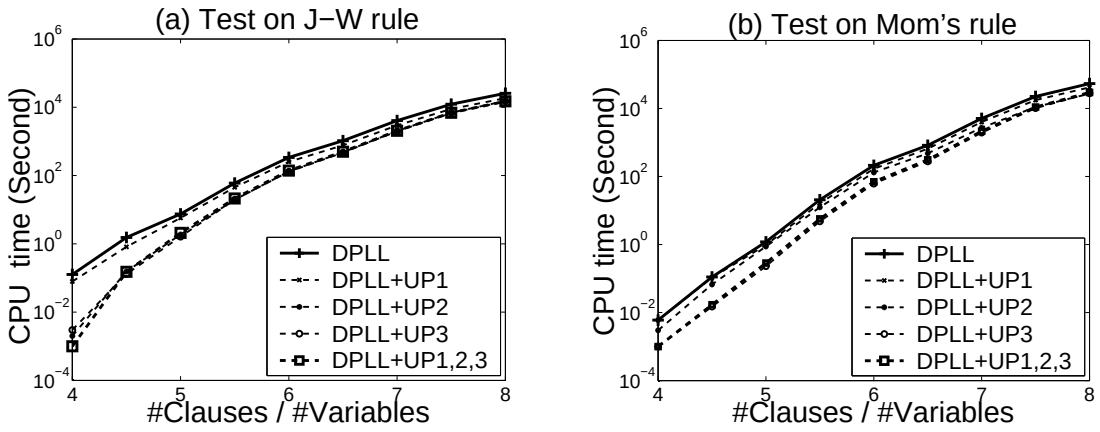

Figure 1: Effects of unit propagation (UP) rules on max-3-SAT problems

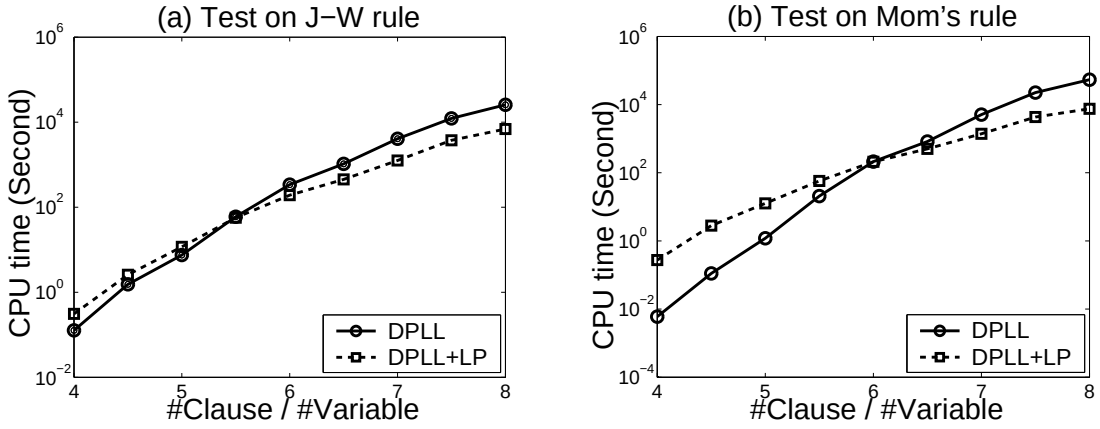

Figure 2: Effects of LP heuristic on max-3-SAT problems

7.1.1 Max-3-SAT problems

The experiments were carried out on random max-3-SAT with 80 variables and clause/variable (C/V) ratios ranging from 4 to 8 in an increment of 0.5. For C/V ratios from 4 to 6 and from 6.5 to 8, 100 and 10 problem instances were used, respectively.

Unit propagation rules are only effective on certain arrange of constrainedness. As shown in Figure 1, each UP rule except UP1 can reduce DPLL's running time by 2-10 times. (detailed running time and speedup for each UP rule is in APPENDIX) When the C/V ratio is low (from 4 to 5.5), the initial upper bound α is close to 0, thanks to the effectiveness of the Walksat algorithm. As a result, solving max-3-SAT is similar to solving 3-SAT. In this case, the percentage of unit clauses is relatively high throughout the search, making the conditions of unit propagations easy to satisfy and unit propagations happen frequently.

DPLL+LP, on the contrary, is ineffective on low-constrainedness regions, due to its overhead

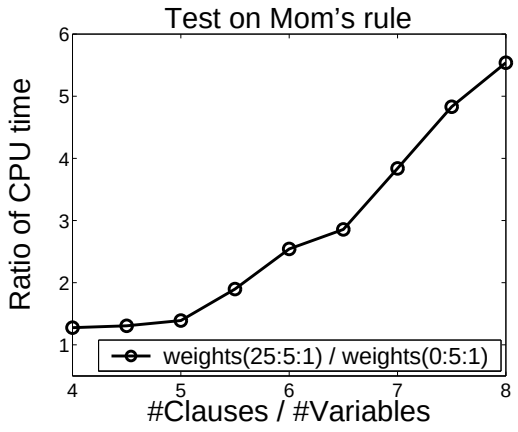

Figure 3: Effects of not assigning weights to unit clauses in DPLL+LP

to the running time. However, as shown in Figure 2, the running time overhead of LP is gradually compensated by the amount of pruning it provides as C/V ratio increases, making LP effective on over-constrained problems. As we mentioned in section 5, the computation time required by an LP call is linear to the number of constraints of the problem at hand [14]. When constrainedness is low, such a linear-time overhead may be still too costly compared to a single DPLL node expansion. On the other hand, in a highly constrained situation where the upper bound α is large, DPLL without LP heuristic may have to search sufficiently deep along a search avenue before it can backtrack, resulting in a large amount of search cost, which is typically exponential in search depth. DPLL+LP, on the other hand, can estimate a reasonably accurate h value with a relatively small overhead for over-constrained problems. The difference in time between using LP and without LP makes DPLL+LP outperform the original DPLL on over-constrained problems.

Note that when running DPLL+LP, we did some modifications to both branching rules. Instead of 4:2:1 and 25:5:1, we assigned 0:5:1 as weight ratio to the Mom's rule and 0:2:1 to two-sided Jeroslow-Wang rule. As discussed in section 5, we need not to eliminate any unit clause in DPLL+LP, so we assign "zero value" to unit clause in weighted variable order. The effect of this "zero unit clause weight" in the Mom's rule can be shown in Figure 3. In DPLL+LP, when we change weight ratio from 25:5:1 to 0:5:1, the CPU time can be reduced by 20 percent in low-constrained regions, e.g.(C/V=4), and 80 percent in high-constrained regions, e.g. (C/V=8). The similar effect also exists for two-sided Jeroslow-Wang rule.

The Mom's and two-sided Jeroslow-Wang rules affect the unit propagations and the LP heuristic differently. As shown in Figures 4(a) and 4(b), the Mom's rule combined with DPLL and DPLL+UP has relatively better performance in not highly constrained regions (C/V < 6); while it is outperformed by the two-sided Jeroslow-Wang rule as C/V ratio increases. (Note that

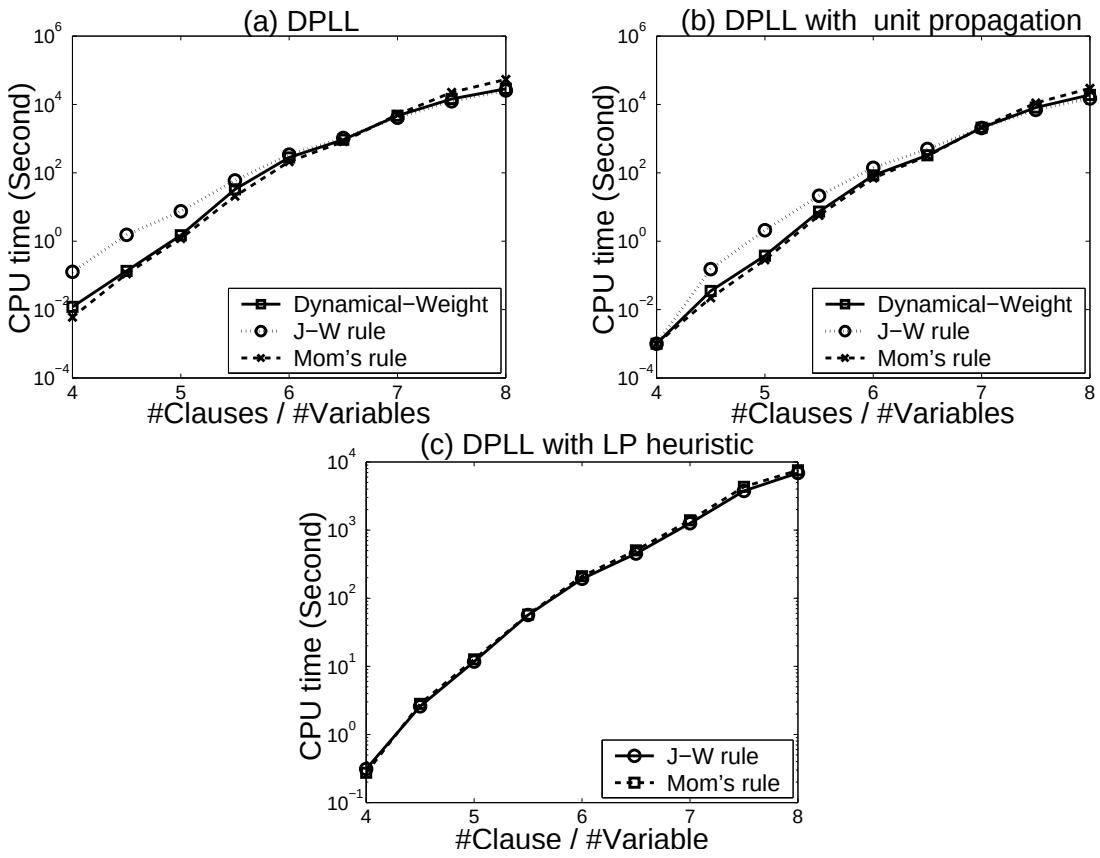

Figure 4: Effects of different branching rules on max-3-SAT problems

the vertical axes of the figures are logarithmic, so the actual difference on running time is substantial.) In DPLL and DPLL+UP, the Mom's rule tends to get ride of unit clauses quickly. If C/V ratio is low, so is the upper bound α. It is more likely that an early increase in the number of violated constraints g will result in a lower bound value exceeding α, forcing the search to backtrack early. However, if the C/V ratio and upper bound α are high, it is not so easy for the value of $g + h$ to exceed α. Therefore, although the Mom's rule can increase the g value in an early stage of the search, it actually produces fewer unit clauses to contribute to the g value as the search progresses. This is mainly because in the Mom's rule, the weights on binary and three literal clauses are smaller than those in two-sided Jeroslow-Wang rule, making it more difficult for non-unit clauses to be turned in to unit clauses. Therefore, the Mom's rule performs better than two-sided Jeroslow-Wang rule in under-constrained regions, but worse in over-constrained regions.

In short, our results showed that the Mom's and two-sided Jeroslow-Wang rules are effective under different problem constrainedness. Our new dynamic-weight variable ordering rule was developed to combine their strengthes under different conditions. Moreover, instead of statically

set the weights, the new rule dynamically adjusts the weights based on the current situation of the search. As the results in Figures 4(a) and 4(b) show, the new rule is nearly the winner under different constraint tightness.

Compared to DPLL and DPLL+UP, the Mom's and two-sided Jeroslow-Wang rules do not make too much difference to DPLL+LP as shown in Figures 4(c). Unlike DPLL and DPLL+UP that use only the g value, DPLL+LP uses both the g value and the h value. The g value is only from unit clauses, while the h value can be contributed by binary and three-literal clauses, making all clauses in DPLL+LP to contribute to the lower bound. Namely, no matter a clause is removed early or later during the search process of a DPLL+LP search tree, it can contribute to the lower bound through the g value (if the clause is removed early) or the h value (if the clause is removed later). As a result, it does not matter whether a variable is branched early or later in DPLL+LP; and DPLL+LP is relatively less sensitive to branching rule than DPLL and DPLL+UP.

7.1.2 Max-2-SAT problems

Compared to max-3-SAT, the scenario on max-2-SAT is relatively simple. Most strategies applicable to max-2-SAT are less sensitive to consrainedness. Because there are only two literals in each clause, any simplification of the problem formula will result in some unit clauses, which, in turn, makes the percentage of unit clauses high and unit propagation happen frequently. In addition, a relatively higher percentage of unit clauses helps to estimate higher h values, which make LP heuristic more efficient.

These arguments can be verified by experimental results. In the experiments, we used random instances with 80 variables and C/V ratios ranging from 2 to 5 in an increment of 0.5. For C/V ratios from 2 to 3 and from 3.5 to 5, 100 and 10 problem instances were used, respectively. As shown in Figure 5, unit propagation rules are very effective on all constrainedness ranges of max-2-SAT. In either branching rule, each unit propagation rule can independently reduce DPLL's running time by 10-1000 times, and their combination makes the greatest effect in most of constrainedness. Moreover, unlike max-3-SAT, the effectiveness of unit propagation rules on max-2-SAT does not degrade as problems become highly constrained. (see APPENDIX for detailed performance of each UP rule) As shown in Figure 6, DPLL+LP is also very effective in all constrainedness ranges. For branching rules, it is clear in Figure 7 that two-sided Jeroslow-Wang rule is the winner for nearly all the situations. All these results suggest that for max-2-SAT, LP heuristic and all the unit propagation rules should be applied and two-sided Jeroslow-Wang rule is preferred over the Mom's rule.

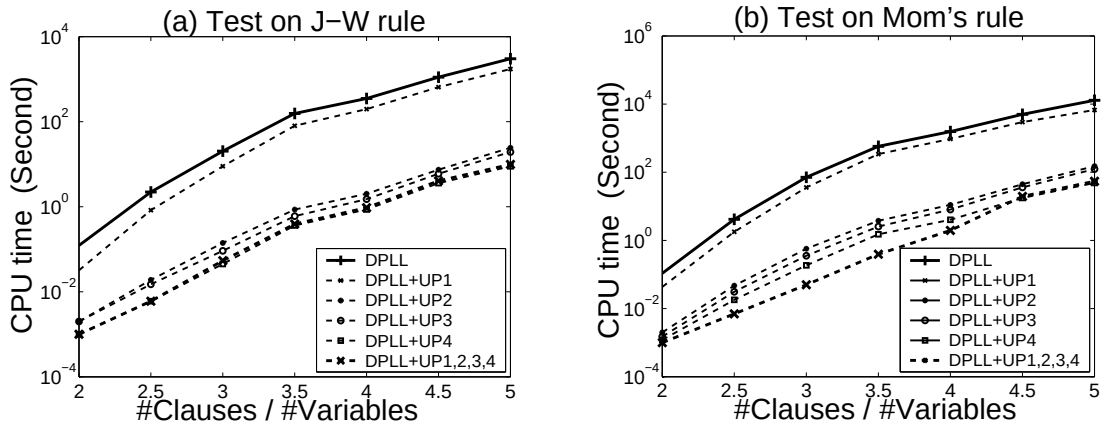

Figure 5: Effects of unit propagation (UP) rules on max-2-SAT problems

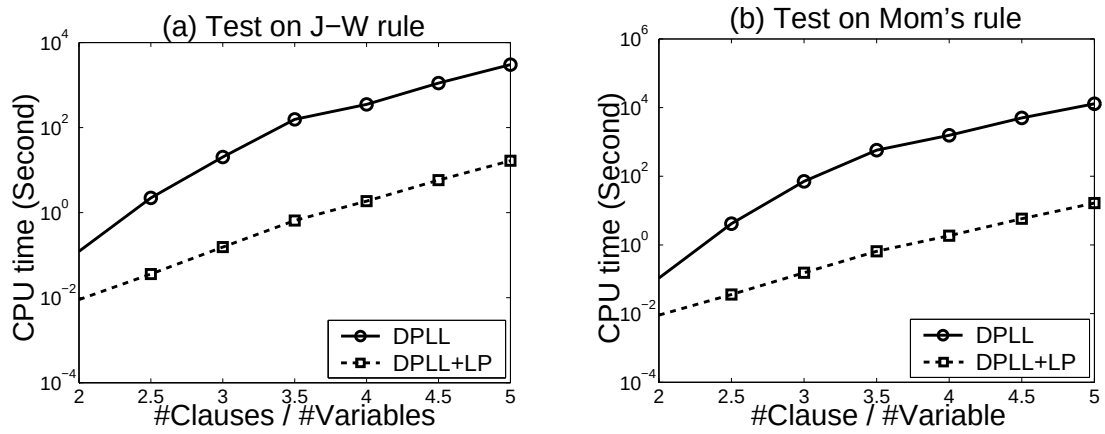

Figure 6: Effects of LP heuristic on max-2-SAT problems

7.2 Integrated algorithm and its performance

Based on our understanding of the effects of the existing strategies and heuristics, in this section, we study the efficacy of our new integrated algorithm, MaxSolver. To reiterate, MaxSolver incorporates in extended DPLL the three new strategies. Based on the results in the previous section, in our experiments with MaxSolver, we applied the unit propagation rules only to max-2-SAT or moderately constrained max-3-SAT, the LP lookahead heuristic to max-2-SAT or highly constrained max-3-SAT, and our new dynamic-weight variable ordering to max-3-SAT.

To fully evaluate its performance, we compared MaxSolver with following existing algorithms for max-SAT and maximum CSP (max-CSP) which we are aware of and whose source codes are available to us:

- The DPLL-based solver BF implemented by Borchers and Furman [3]

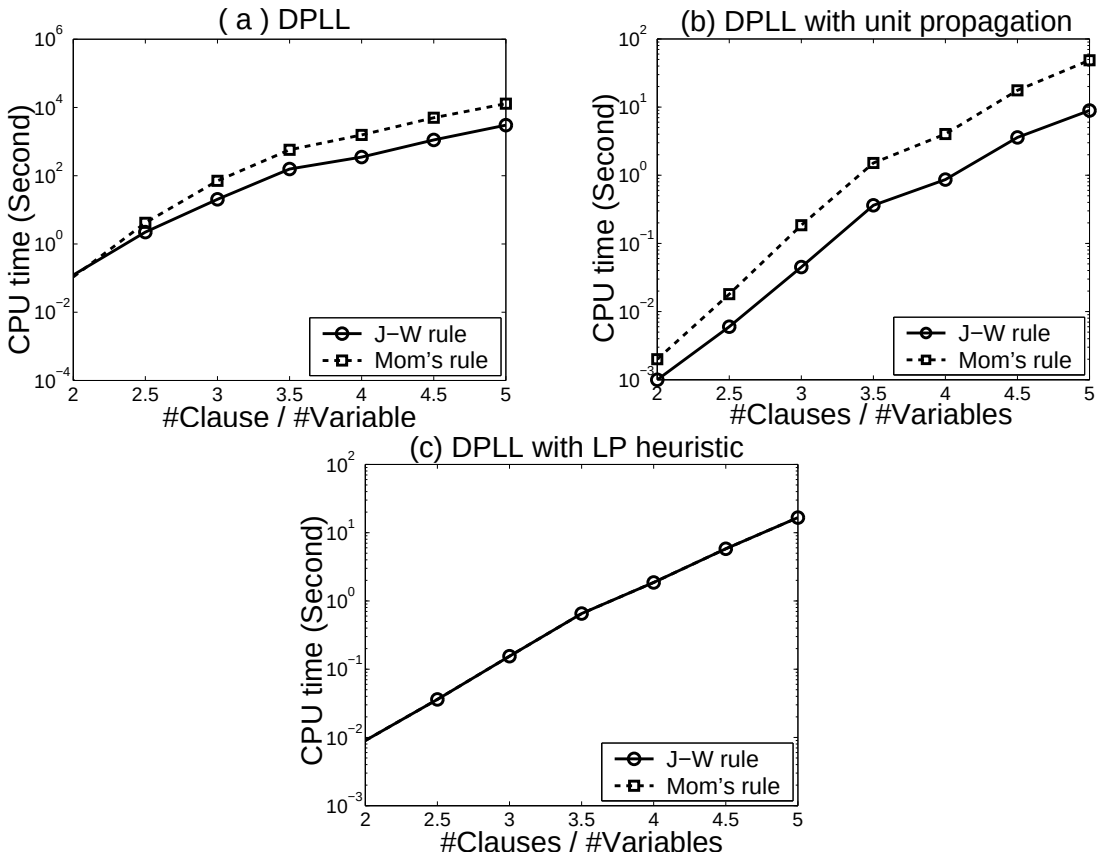

Figure 7: Effects of different branching rules on max-2-SAT problems

- The DPLL-based solver AMF implemented by Alsinet, Manya, and Planes [1]
- The Pseudo Boolean Optimization solver PBS2.1 [8]
- The weighted CSP-based solver WCSP [6]
- The generic commercial ILP solver CPLEX 8.0 [19]

These algorithms contain most of the known techniques for SAT, max-SAT and max-CSP. To the best of our knowledge, BF and AMF are the only exact max-SAT solvers implemented in C language that are variants of DPLL method. Another earlier exact max-SAT solver implemented by wallace [30] was in Lisp, so we do not include it in our comparison. BF is an extended DPLL with the Mom's rule and a simple unit propagation that is similar but weaker than our UP2. AMF is derived from BF which includes the lower bound described in section 3.2 and uses two-sided Jeroslow-Wang rule. PBS is a specialized 0-1 ILP solver and uses advanced techniques such as conflict diagnosis, random restarts, improved backtracking, and cutting plane. WCSP encodes a max-CSP (and max-SAT) into a weighted constraint network and solves the problem using state-

C/V	MaxSolver	BF		AMF		PBS		WCSP		CPLEX	
2.0	**0.00**	0.04	(36)	0.07	(66)	3.01	(3013)	1	(1000)	0.01	(14)
2.5	**0.01**	1.21	(207)	1.04	(179)	186.00	(320612)	13	(2167)	0.12	(12)
3.0	**0.04**	51.79	(1300)	11.87	(298)	—	—	109	(2725)	0.27	(7)
3.5	**0.18**	687.55	(3900)	80.00	(449)	—	—	744	(4133)	0.60	(3)
4.0	**0.85**	12228.00	(14000)	485.10	(575)	—	—	4577	(5385)	1.61	(2)
4.5	**3.89**	—	—	2073.52	(532)	—	—	—	—	5.88	(1)
5.0	13.00	—	—	4617.56	(355)	—	—	—	—	**12.82**	(**1**)

Table 1: Average CPU times on max-2-SAT of 80 variables.

of-art algorithms for weighted CSP. We used the default settings for all the solvers, except for PBS which used VSIDS decision heursitic [26] (as advised by the author). The results presented below can be viewed as a comprehensive evaluation of these existing algorithms on max-SAT.

We used random max-2-SAT, max-3-SAT, and unsatisfiable instances in SATLIB [18], which were generated from applications such as planning and model checking. The results are respectively in Tables 1 to 3, where "-" indicates an incomplete run after 5 hours of CPU time. For each problem class, the tables list either the C/V ratio or the numbers of variables V and clauses C, followed by columns for the running times of the six solvers in seconds. The numbers in parentheses are MaxSolver's relative speedups over the existing methods. #Unsat in Table 3 is the number of violated clauses.

C/V	MaxSolver	BF		AMF		PBS		WCSP		CPLEX	
4.0	**0.00**	**0.00**	(**1.0**)	**0.00**	(**1.0**)	0.01	(16)	0	—	0.91	(113.8)
4.5	**0.01**	0.01	(1.0)	1.14	(87.3)	44.90	(3563)	19	(1461)	6.29	(499.2)
5.0	**0.15**	0.19	(1.3)	7.43	(50.5)	—	—	321	(2183)	21.06	(140.4)
5.5	**4.26**	6.95	(1.6)	64.79	(15.2)	—	—	—	—	88.41	(20.8)
6.0	**38.00**	104.00	(2.7)	386.00	(10.2)	—	—	—	—	290.84	(7.6)
6.5	**228.00**	629.00	(2.8)	1342.52	(5.9)	—	—	—	—	1197.38	(5.2)
7.0	**1723.00**	9498.00	(5.5)	7937.17	(4.6)	—	—	—	—	3061.91	(1.8)
7.5	**7493.00**	—	—	—	—	—	—	—	—	14665.30	(2.0)

Table 2: Average CPU times on max-3-SAT of 80 variables.

For random max-2-SAT (Table 1), BF degrades quickly as C/V ratio increases. As BF is the only solver for max-2-SAT in which the Mom's rule is applied, its poor performance means that the Mom's rule alone is ineffective on max-2-SAT. MaxSolver is also much faster than AMF, which indicates that our unit propagation rules can dramatically reduce the node expansions, and that our LP heuristic is effective as well. CPLEX performance second best, which indicates that ILP solver is efficient for max-2-SAT problems. The other two non-DPLL solvers, PBS and WCSP, perform much worse than MaxSolver. PBS is unable to solver problems with more than 240 clauses.

Instances	V:C	#Unsat	MaxSolver	BF		AMF		PBS		WCSP		CPLEX	
jnh8	100:850	2	**0.001**	0.03	(30.0)	0.04	(40.0)	439.80	(439800.000)	2	(2000)	37.16	(37160)
jnh9	100:850	2	**0.010**	0.04	(4.0)	0.53	(5.3)	40.69	(4069.000)	1	(100)	302.36	(30236)
jnh14	100:850	2	**0.001**	0.03	(30.0)	0.04	(41.0)	13.21	(13210.000)	2	(2000)	193.72	(193720)
jnh211	100:800	2	**0.001**	0.03	(30.0)	0.04	(37.0)	26.77	(26770.000)	3	(3000)	42.62	(42620)
jnh307	100:900	3	**0.010**	0.32	(32.0)	0.41	(41.0)	1391.60	(139160.000)	5	(500)	158.22	(15822)
aim50−2.0no1	50:100	1	0.040	0.02	(0.5)	0.07	(1.8)	**0.00**	(**0.000**)	1	(25)	1.13	(28)
aim50−2.0no2	50:100	1	0.010	0.01	(1.0)	0.03	(3.0)	0.01	(1.000)	0	—	5.12	(512)
aim50−2.0no3	50:100	1	0.010	0.02	(2.0)	0.05	(5.0)	0.01	(1.000)	0	—	3.77	(377)
pret60−40	60:160	1	3.850	5.65	(1.5)	14.25	(3.7)	**0.02**	(**0.005**)	75	(19)	—	—
pret60−60	60:160	1	3.970	5.67	(1.4)	14.26	(3.6)	**0.01**	(**0.003**)	75	(19)	—	—
pret60−75	60:160	1	4.830	5.63	(1.2)	14.19	(2.9)	**0.02**	(**0.004**)	75	(16)	—	—
dubois25	75:200	1	18.600	121.80	(6.6)	319.70	(17.2)	**0.27**	(**0.001**)	600	(32)	—	—

Table 3: Average CPU times on unsatisfiable SATLIB instances.

For random max-3-SAT (Table 2), BF performs better than what it does on max-2-SAT and is sometimes competitive when C/V ratio is low. However, it still degrades faster than MaxSover and even AMP as C/V ratio increases, indicating that not only the Mom's rule on max-3-SAT becomes less effective, but also the LP heuristic becomes effective as C/V ratio increases. ILP solver CPLEX is not as good as on max-2-SAT, especially when C/V ratio is low. PBS and WCSP are still not competitive at all on max-3-SAT.

MaxSolver also outperforms the other solvers on many instances from SATLIB. As shown in Table 3, jnh instances are best solved using MaxSolver. For pret instances and dubois25, PBS is the winner. Note that PBS is many orders of magnitude slower than MaxSolver jnh instances, each of which has at least 2 unsatisfiable clauses. This matches the results in Tables 1 and 2, where PBS is the worst on highly over-constrained problems. Therefore, PBS is not suitable for hard max-SAT. CPLEX performs much worse than our MaxSolver, which indicates that ILP solver is not suitable for low-constrained or special structure instances. WCSP is also much worse than MaxSolver in all the instances, as it was originally developed for max-CSP. Finally, MaxSolver still outperforms BP and AMP for nearly every problems, and solves every of them in a reasonable amount of time. Therefore, all of the results indicate that our MaxSolver, being developed based on random max-SAT, works fairly well on these instances with special structures embedded.

In summary, our results show that MaxSolver and its three improving strategies are effective on max-SAT problems, outperforming the five existing algorithms on random max-SAT and many instances from SATLIB, often with orders of magnitude reduction on running time.

8 Related Work and Discussions

Hansen gave a complete summary of different mathematical formulations and heuristics on max-SAT in [13]. Freuder and Wallace carried out an early and the most significant research on over-constrained satisfaction problems by directly extending the techniques for constraint satisfaction [10, 30]. They proposed a number of basic ideas of how to construct a DPLL-based exact max-SAT solver, most of which we have discussed in section 3.

Our UP1 and a rule similar to UP2 were mentioned in [32, 29, 1]. UP3 was first proposed by Niedermeier and Rossmanith in [27], and was applied to max-2-SAT in [32]. In [27], Niedermeier and Rossmanith also presented a set of transformation and splitting rules in order to provide a worst case complexity for max-SAT. However, conditions for using most of those rules are too hard to satisfy. Our new UP4 was developed based on an idea of formulating max-SAT into nonlinear program. (The first nonlinear 0-1 formulation for max-SAT was proposed by Hammer and Rudeanu in [12]) The combination of all these four rules has been shown very efficient and powerful in our experiments.

[21] is perhaps the first to apply ILP to max-SAT. It showed that an ILP-based solver is able to outperform DPLL-based solvers on max-2-SAT. However, when applied to max-3-SAT, the ILP-based solver is much slower than a DPLL-based algorithm. In [2], Blair, Jeroslow and Lowe applied LP to study SAT, although bounds so obtained would be poor, compared to applying ILP. Better bounds might be obtained, as done by Hooker in [15]. In this paper, we proposed to use LP for max-SAT, and successfully showed its power on max-3-SAT for the first time.

Little work has been done on variable ordering for max-SAT, except the work in [30] on the effects of applying in-most-unit-clause and in-most-shortest-clause heuristics on small random max-SAT of 25 variables. Our dynamic-weight variable ordering is novel in that it is able to adjust itself according to problem characteristics to cope with different constraint situations.

9 Conclusions

Max-SAT is an important problem with many real-world applications. However, the existing algorithms for max-SAT are typically restricted to simple problems with small numbers of variables and low clause/variable ratios. The main contributions of this paper are three effective methods for max-SAT and an integrated algorithm for solving hard max-SAT instances, which include a set of unit propagation rules, a linear-programming based lookahead heuristic and a dynamic-weight variable ordering rule, and constitute the main building blocks of a new max-SAT solver, called MaxSolver, developed in this paper.

C/V	DPLL	DPLL+UP1		DPLL+UP2		DPLL+UP3		DPLL+UP1,2,3	
4.0	0.127	0.079	(1.6)	0.002	(63.5)	0.003	(42.3)	**0.001**	(**127.0**)
4.5	1.538	0.817	(1.9)	0.155	(9.9)	**0.131**	(**11.7**)	0.152	(10.1)
5.0	7.504	5.730	(1.3)	1.874	(4.0)	**1.619**	(**4.6**)	2.089	(3.6)
5.5	59.876	45.141	(1.3)	19.506	(3.1)	**19.495**	(**3.1**)	21.300	(2.8)
6.0	339.796	252.501	(1.3)	132.012	(2.6)	**129.476**	(**2.6**)	139.925	(2.4)
6.5	1046.228	767.516	(1.4)	479.744	(2.2)	**470.264**	(**2.2**)	498.510	(2.1)
7.0	4074.524	2945.308	(1.4)	2031.124	(2.0)	**1980.976**	(**2.1**)	2043.560	(2.0)
7.5	12311.832	8930.277	(1.4)	6940.387	(1.8)	**6777.374**	(**1.8**)	6911.136	(1.8)
8.0	25633.211	18493.927	(1.4)	15266.256	(1.7)	**14903.538**	(**1.7**)	14973.617	(1.7)

Table 4: Effects of unit propagation (UP) rules on max-3-SAT problems, $\mid V \mid$ = 80, tested in two-sided Jeroslow-Wang rule.

We experimentally showed that these new strategies and MaxSolver are effective on difficult max-2-SAT and max-3-SAT problems. MaxSolver, is significantly superior to five existing algorithms for max-SAT that we considered. MaxSolver is able to significantly reduce the running time of the existing algorithms, sometimes with orders of magnitude reduction, on random max-SAT instances and many max-SAT instances converted from real application domains. MaxSover will be made available to the public on our webpage.

Acknowledgment

This research was supported in part by NSF grants IIS-0196057 and ITR/EIA-0113618, and in part by DARPA Cooperative Agreement F30602-00-2-0531. We would like to thank Zhongsheng Guo for an early implementation of the DPLL algorithm. We would also like to thank Fadi Aloul, Javier Larrosa, and Jordi Plane for making their codes available to us.

Appendix: Numerical Results on the Effects of Unit Propagation Rules

Tables 4 to 7 list effects of different unit propagation rules on extended DPLL algorithm. Tables 4, 5 are corresponding to Fig 1; and Tables 6, 7 are corresponding to Fig 5 . For each unit propagation rule(DPLL+UPs), the running time in seconds is given, followed by its relative speedup over extended DPLL (DPLL/DPLL+UPs) in parentheses.

C/V	DPLL	DPLL+UP1		DPLL+UP2		DPLL+UP3		DPLL+UP1,2,3	
4.0	0.006	0.007	(0.9)	0.003	(2.0)	**0.001**	(**6.0**)	**0.001**	(**6.0**)
4.5	0.111	0.124	(0.9)	0.157	(0.7)	**0.015**	(**7.4**)	0.022	(5.0)
5.0	1.197	0.964	(1.2)	1.877	(0.6)	**0.231**	(**5.2**)	0.285	(4.2)
5.5	20.641	16.354	(1.3)	19.440	(1.1)	**4.804**	(**4.3**)	5.680	(3.6)
6.0	211.885	167.644	(1.3)	131.511	(1.6)	**61.716**	(**3.4**)	69.827	(3.0)
6.5	823.916	655.768	(1.3)	478.722	(1.7)	**277.262**	(**3.0**)	308.516	(2.7)
7.0	5074.660	3992.982	(1.3)	2024.720	(2.5)	**1951.556**	(**2.6**)	2134.304	(2.4)
7.5	22493.912	17568.003	(1.3)	**6925.618**	(**3.2**)	10202.106	(2.2)	10997.884	(2.0)
8.0	53867.456	41687.331	(1.3)	**15228.462**	(**3.5**)	27862.787	(1.9)	29578.631	(1.8)

Table 5: Effects of unit propagation (UP) rules on max-3-SAT problems, $\mid V \mid = 80$, tested on the Mom's rule.

C/V	DPLL	DPLL+UP1		DPLL+UP2		DPLL+UP3		DPLL+UP4		DPLL+UP1,2,3,4	
2.0	0.123	0.032	(3.8)	0.002	(61.5)	0.002	(61.5)	**0.001**	(**123.0**)	**0.001**	(**123.0**)
2.5	2.217	0.828	(2.7)	0.019	(116.7)	0.015	(147.8)	**0.006**	(**369.5**)	**0.006**	(**369.5**)
3.0	20.337	8.952	(2.3)	0.141	(144.2)	0.092	(221.1)	0.054	(376.6)	**0.045**	(**451.9**)
3.5	156.406	80.082	(2.0)	0.860	(181.9)	0.600	(181.9)	0.392	(399.0)	**0.362**	(**432.1**)
4.0	351.004	197.558	(1.8)	2.020	(173.8)	1.504	(233.4)	0.950	(369.5)	**0.864**	(**406.3**)
4.5	1116.926	648.690	(1.7)	7.418	(150.6)	5.974	(187.0)	4.050	(275.8)	**3.586**	(**311.5**)
5.0	3022.160	1731.054	(1.7)	24.378	(124.0)	19.378	(124.0)	9.918	(304.7)	**8.918**	(**338.9**)

Table 6: Effects of unit propagation (UP) rules on max-2-SAT problems, $\mid V \mid = 80$, tested on two-sided Jeroslow-Wang rule.

References

[1] T. Alsinet, F. Manya, and J. Planes. Improved branch and bound algorithms for Max-SAT. In *6th International Conference on Theory and Applications of Satisfiability Testing - SAT2003*, pages 408–415, 2003.

[2] C. E. Blair, R. G. Jeroslow, and J. K. Lowe. Some results and experiments in programming techniques for prepositional logic. *Computers and Operations Research*, 13(5):633–645, 1986.

[3] B. Borchchers and J. Furman. A two-phase exact algorithm for Max-SAT and weighted Max-SAT problems. *Journal of Combinatorial Optimization*, 2(4):299–306, 1999.

[4] S. A. Cook. The complexity of theorem-proving procedures. In *Proc. 3rd IEEE Symposium on the Foundations of Computer Science, FOCS-71*, pages 151–158, 1971.

C/V	DPLL	DPLL+UP1		DPLL+UP2		DPLL+UP3		DPLL+UP4		DPLL+UP1,2,3,4	
2.0	0.108	0.043	(2.5)	0.002	(54.0)	0.001	(108.0)	0.002	(54.0)	**0.001**	(**108.0**)
2.5	4.155	1.789	(2.3)	0.047	(88.4)	0.031	(134.0)	0.018	(230.8)	**0.007**	(**593.6**)
3.0	70.922	35.827	(2.0)	0.569	(124.6)	0.355	(199.8)	0.185	(383.4)	**0.050**	(**1418.4**)
3.5	574.818	340.998	(1.7)	3.766	(152.6)	2.552	(225.2)	1.514	(379.7)	**0.394**	(**1458.9**)
4.0	1562.466	949.422	(1.6)	11.002	(142.0)	8.058	(193.9)	4.022	(388.5)	**0.962**	(**1624.2**)
4.5	5000.200	2967.804	(1.7)	44.520	(112.3)	36.046	(138.7)	**17.626**	(**283.7**)	19.586	(255.3)
5.0	12910.389	6716.592	(1.9)	150.612	(85.7)	124.118	(104.0)	**48.638**	(**265.4**)	54.470	(237.0)

Table 7: Effects of unit propagation (UP) rules on max-2-SAT problems, $\mid V \mid = 80$, tested on the Mom's rule.

[5] J. M. Crawford and L. D. Auton. Experimental results on the crossover point in satisfiability problems. In *Proceedings of the 11th National Conference on Artificial Intelligence*, pages 21–27, 1993.

[6] S. d. Givry, J. Larrosa, P. Meseguer, and T. Schiex. Solving Max-SAT as weighted CSP. In *Principles and Practice of Constraint Programming - CP2003*, pages 363–376, 2003.

[7] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the Association for Computing Machinery*, 7:201–215, 1960.

[8] H. Dixon and M. L. Ginsberg. Inference methods for a pseudo-boolean satisfiability solver. In *Proceedings of the 18th National Conference on Artificial Intelligence*, pages 635–640, 2002.

[9] J. W. Freeman. *Improvements to Propositional Satisfiability Search Algorithms*. PhD thesis, University of Pennsylvania, 1995.

[10] E. C. Freuder and R. J. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58:21–70, 1992.

[11] M. R. Garey and D. S. Johnson. *Computers and Intractability*. Freeman, 1979.

[12] P. L. Hammer and S. Rudeanu. *Boolean methods in Opeartions Research and Related Areas*. Springer-Verlag, 1968.

[13] P. Hansen and B. Jaumard. Algorithm for the maximum satisfiability problem. *Computing*, 44:279–303, 1990.

[14] F. S. Hillier and G. J. Lieberman. *Introduction to Operations Research*. McGraw-Hill, 2001.

[15] J. N. Hooker. Input proofs and rank one cutting-planes. *ORSA Journal on Computing*, 1:137–145, 1989.

[16] J. N. Hooker and G. Fedjki. Branch-and-cut solution of inference problems in prepositional logic. *Annals of Mathematics and Artificial Intelligence*, 1:123–139, 1990.

[17] J. N. Hooker and V. Vinay. Branching rules for satisfiability. *Journal of Automated Reasoning*, 15:359–383, 1995.

[18] H. H. Hoos and T. Stuzle. http://www.satlib.org, 1999.

[19] http://www.ilog.com/products/cplex.

[20] D. S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9:256–278, 1974.

[21] S. Joy, J. Mitchell, and B. Borchers. A branch and cut algorithm for Max-SAT and weighted Max-SAT. In D. Du, J. Gu, and P.M. Pardalos, editors, *Satisfiability Problem: Theory and Applications*, pages 519–536. 1997.

[22] C. M. Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pages 366–371, 1997.

[23] C. L. Liu. *Introduction to Combinatorial Mathematics*. McGraw-Hill, 1968.

[24] D. McAllester, B. Selman, and H. Kautz. Evidence for invariants in local search. In *Proceedings of the 14th National Conference on Artificial Intelligence*, pages 321–326, 1997.

[25] D. B. Mitchell, B. Selman, and H. Levesque. Hare and easy distributions of SAT problems. In *Proceedings of the 11th National Conference on Artificial Intelligence*, pages 459–465, 1993.

[26] M. Moskewics, C. Madigan, Y. Zhao, L. Zhang, , and S. Malik. Chaff: Engineering an efficient sat solver. In *Proceedings of the Design Automation Conference*, 2001.

[27] R. Niedermeier and P. Rossmanith. New upper bounds for maximum satisfiability. *Journal Algorithm*, 36:63–88, 2000.

[28] B. Selman, H. Kautz, and B. Cohen. Noise strategies for local search. In *Proceedings of the 12th National Conference on Artificial Intelligence*, pages 337–343, 1994.

[29] R. J. Wallace. Enhancing maximum satisfiability algorithms with pure literal strategies. In *11th Canadian Conference on Artificial Intelligence*, 1996.

[30] R. J. Wallace and E. C. Freuder. Comparative study of constraint satisfaction and davis-putnam algorithms for maximum satisfiability problems. In D. Johnson and M. Trick, editors, *Cliques, Coloring, and Satisfiability*, pages 587–615. 1996.

[31] J. P. Walser. *Integer Optimization Local Search*. Springer, 1999.

[32] H. Zhang, H. Shen, and F. Manya. Exact algorithms for Max-SAT. In *Workshop on First-Order Theorem Proving (FTP-03)*, 2003.