Available online at www.sciencedirect.com
SCIENCE DIRECT®
ELSEVIER
Artificial Intelligence 164 (2005) 47–80
Artificial Intelligence
www.elsevier.com/locate/artint

MaxSolver: An efficient exact algorithm for (weighted) maximum satisfiability

Zhao Xing, Weixiong Zhang *

Department of Computer Science and Engineering, Washington University in Saint Louis, Saint Louis, MO 63130, USA

Received 17 April 2004

Available online 2 March 2005

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 Boolean formula. A branch and bound algorithm based on the Davis–Putnam–Logemann–Loveland procedure (DPLL) is one of the most competitive exact algorithms for solving max-SAT. In this paper, we propose and investigate a number of strategies for max-SAT. The first strategy is a set of unit propagation or unit resolution rules for max-SAT. We summarize three existing unit propagation rules and propose a new one based on a nonlinear programming formulation of max-SAT. The second strategy is an effective lower bound based on linear programming (LP). We show that the LP lower bound can be made effective as the number of clauses increases. The third strategy consists of a binary-clause first rule and a dynamic-weighting variable ordering rule, which are motivated by a thorough analysis of two existing well-known variable orderings. Based on the analysis of these strategies, we develop an exact solver for both max-SAT and weighted max-SAT. Our experimental results on random problem instances and many instances from the max-SAT libraries show that our new solver outperforms most of the existing exact max-SAT solvers, with orders of magnitude of improvement in many cases.

Keywords: Weighted maximum satisfiability; DPLL; Unit propagation; Linear programming; Nonlinear programming; Variable ordering

* Corresponding author.
E-mail addresses: zx2@cse.wustl.edu (Z. Xing), zhang@cse.wustl.edu (W. Zhang).

0004-3702/$ – see front matter
doi:10.1016/j.artint.2005.01.004

1. Introduction and overview

Boolean satisfiability (SAT) is an archetypical decision problem in artificial intelligence, logic and theory of computation. SAT with more than two literals (variables or their negations) per clause is NP-complete [6,29]. Maximum Boolean satisfiability (max-SAT) is the optimization counterpart of SAT, whose aim is to maximize 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 vice versa. Therefore, max-SAT is more difficult to solve than SAT. Max-SAT is NP-hard [15] even when each clause has no more than two literals, while SAT with two literals per clause (2-SAT) is polynomial soluble.

Weighted max-SAT is an extension of max-SAT in which a clause carries a weight, representing the significance of the clause or an induced penalty if it is violated. In weighted max-SAT, the objective is to maximize the total weight of the satisfied clauses. Max-SAT and weighted max-SAT have many real-world applications in domains such as scheduling, configuration problems, probabilistic reasoning, auction, and pattern recognition [14,20]. For simplicity, in this paper, when we mention max-SAT, we refer to both weighted and unweighted max-SAT. Following the convention for SAT, we refer to the ratio of the number of clauses to the number of variables as the "constrainedness" of max-SAT.

The Davis–Putnam–Logemann–Loveland (DPLL) algorithm for SAT [10] can be extended to a branch-and-bound (BnB) algorithm for max-SAT. A BnB-based DPLL algorithm has been shown to be among the most competitive for max-SAT [43]. Much effort has been devoted to improving the performance of such a BnB-based DPLL algorithm for max-SAT by combining the techniques previously developed for SAT [4,28,43] and many methods used in Operations Research (OR), such as integer linear programming (ILP) and cutting plane methods [12,23,28]. However, these efforts have enjoyed 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 [28], which is max-SAT with no more than two literals per clause. On the other hand, even though a BnB-based DPLL algorithm is an efficient algorithm for max-SAT, it can handle relatively small problems with moderate degrees of constrainedness.

Therefore, despite the previous effort, much work is still needed in order to develop efficient algorithms for both max-SAT and weighted max-SAT, and special care is required when extending SAT techniques to max-SAT. In principle, most techniques developed for SAT can be extended to max-SAT [14,20,43]. However, the SAT techniques 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 explicitly captured in the unit propagation or unit resolution 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. 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 for SAT, and the research of developing efficient exact max-SAT solver deserves much attention, due to the generality and importance of the problem.

Aiming at solving difficult max-SAT and weighted max-SAT problems *optimally*, we review the previous research on max-SAT, those taking the DPLL framework for SAT in particular, and develop an efficient exact max-SAT algorithm based on DPLL. 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 combination of four unit propagation rules for max-SAT. Three of these rules were proposed by others in previous studies; we analyze them extensively in this research. The fourth, new unit propagation rule is developed in this research based on an integer nonlinear programming formulation of max-SAT. This is an innovative contribution, enlarging our arsenal of unit propagations for max-SAT. We also consider different ways to combine these four propagation rules in our study.

The second element of our max-SAT algorithm is an effective lookahead lower bound to estimate the minimum number of clauses unsatisfiable at a node during the search. Our lower bound is based on linear programming (LP) [21]. This is a remarkable contribution; it is perhaps the first successful application of LP to max-SAT, despite similar (but not successful) previous efforts to apply integer LP (ILP) to max-SAT [12,23,28].

The third ingredient consists of two new variable-ordering or branching rules, which were inspired by the results of a close examination of two popular variable-ordering rules for SAT, i.e., the Mom's rule [8,30] and the two-side Jeroslow–Wang rule [24], on max-SAT. The first new variable-ordering rule is designed for max-2-SAT. As its name, binary-clause first rule, indicates, this rule gives a higher priority to a variable in binary clauses than those in unit clauses. The second new rule is designed to cope with large range of constrainedness values of max-3-SAT instances. It is a dynamic variable-ordering heuristic that is able to dynamically change its variable ordering from close to the Mom's rule to close to the two-sided Jeroslow–Wang rule as the constrainedness increases.

The paper is organized as follows, we first discuss max-SAT and describe two types of mathematical formulation of the problem in Section 2. In Section 3, we review the DPLL algorithm for SAT and how it can be extended to max-SAT. We 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 four unit propagation rules for max-SAT. In Section 5, we develop a lower bound function based on linear programming, and discuss why LP-based lower bound is effective on highly constrained problem instances. In Section 6, we propose the binary-clause first and dynamic-weighting variable ordering rules. We present experimental results of our new strategies, and describe an efficient max-SAT algorithm that combines all our new strategies in Section 7. We also systematically compare our new solver with the most existing max-SAT solvers in Section 7. Finally, we discuss some related work in Section 8, and conclude in Section 9.

Preliminary results of the research and an extended abstract of this paper appeared in [45].

2. Formulation 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 with only one literal is called *unit clause*, a clause with two literals is named *binary clause*, a clause with three literals is referred to as a clause of size three, and so on. 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 that maximizes the total number (or weight) of satisfied clauses [14]. This is maximum satisfiability, maximum SAT, or max-SAT for short.

In general, a weighted max-SAT can be formulated as a minimization problem. Given a set of m clauses defined on n Boolean variables, $\{v_1, v_2, \ldots, v_n\}$, it is to minimize objective

$$W = \sum_{i=1}^{m} w_i y_i,$$

subject to

$$y_i = \begin{cases} 1, & \text{if the } i\text{th clause is unsatisfied,} \\ 0, & \text{otherwise,} \end{cases}$$

where w_i is the weight of the ith clause, and y_i is a decision variable [21] corresponding to the ith clause, for $i = 1, 2, \ldots, m$. When the problem is unweighted, $w_i = 1$.

2.1. Linear programming

Max-SAT can be formulated as an integer linear program (ILP) [28] or a pseudo-Boolean formula [12,44]. We map a Boolean variable v_i to an integer variable x_i that takes value 1 when v_i is True or 0 when it is False, i.e., $x_i = 1$ or 0 when $v_i = T$ or F, respectively. We then map $\bar{v}_i$ to $1 - x_i$. With these mappings, we can formulate a clause as a linear inequality. For example, clause $(v_1 \vee \bar{v}_2 \vee v_3)$ can be mapped to $x_1 + (1 - x_2) + x_3 \geqslant 1$. Here, the inequality means that the clause must be satisfied in order for the left side of the inequality to have a value not smaller than one.

However, a clause in a max-SAT may not be satisfied at all, so that the corresponding inequality may be violated. To address this issue, we introduce an auxiliary integer variable y_i (or decision variable) to the left side of the ith mapped inequality. Variable $y_i = 1$ if the corresponding clause is unsatisfied, making the inequality valid; otherwise, $y_i = 0$. Since the objective is to minimize the total weight of violated clauses, it is equivalent to minimizing the sum of the products of the clause weights and the decision variables that are forced to take value 1. For example, $(v_1 \vee \bar{v}_2 \vee v_3)$ (weight 2), $(v_2 \vee \bar{v}_4)$ (weight 3) can be written as an ILP of minimizing $W = 2y_1 + 3y_2$, subject to the constraints of $x_1 + (1 - x_2) + x_3 + y_1 \geqslant 1$ and $x_2 + (1 - x_4) + y_2 \geqslant 1$.

The linear 0-1 program formulation of max-SAT suggests that the problem could be solved by integer linear programming (ILP). However, ILP is NP-hard. Furthermore, as shown in [28], except for max-2-SAT, a direct application of ILP to other max-SAT problems does not seem to be effective.

2.2. Nonlinear programming

The ILP formulation of max-SAT can be extended to a nonlinear program formulation. We will use this formulation to derive a new unit resolution rule for max-SAT in Section 4.4. This extension can be achieved by applying the inclusion-exclusion principle [31] to turn the inequalities in an ILP formulation into equalities. Here, we introduce an integer expression to represent a *literal*. For example, given $(v_1 \vee \bar{v}_2 \vee v_3)$, we introduce integer expressions x_1, $1-x_2$ and x_3 for the literals v_1, $\bar{v}_2$ 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, we then write a nonlinear equation $f_i + y_i = 1$ for the ith clause of a given formula, where y_i is a decision variable taking value 0 or 1. Taking $(v_1 \vee \bar{v}_2 \vee v_3)$ as an example, we have

$$f_i = [x_1 + 1 - x_2 + x_3] - \big[x_1(1-x_2) + x_1x_3 + (1-x_2)x_3\big] + x_1(1-x_2)x_3.$$

Note that f_i can take value 1 or 0. Specifically, $f_i = 0$ if no literal in the clause is set to true, or $f_i = 1$ otherwise. As in the ILP formulation, we introduce decision variables, y_i's, to count for unsatisfied clauses. Here, $y_i = 1$ if $f_i = 0$, and $y_i = 0$ if $f_i = 1$. For a binary clause, e.g., $(v_1 \vee v_3)$, or a unit clause, e.g., $(\bar{v}_2)$, the corresponding nonlinear equation becomes $x_1 + x_3 - x_1x_3 + y_i = 1$ or $1 - x_2 + y_i = 1$, respectively.

In general, f_i is a function of $x_u, x_v, x_w, \ldots, x_ux_v, x_ux_w, \ldots, x_ux_vx_w, \ldots$, where $x_u, x_v, x_w, \ldots$ are integer variables introduced for the Boolean variables in the ith clause. By using $y_i = 1 - f_i$, a max-SAT problem is to minimize the following nonlinear objective function

$$\begin{aligned} W &= \sum_{i=1}^{m} w_i y_i = \sum_{i=1}^{m} w_i(1 - f_i) \\ &= c + \sum_{x_i \in V} \pi_i x_i + \sum_{x_i, x_j \in V} \pi_{i,j} x_i x_j + \sum_{x_i, x_j, x_k \in V} \pi_{i,j,k} x_i x_j x_k + \cdots, \end{aligned} \tag{1}$$

where $V = \{x_1, x_2, \ldots, x_n\}$ is a set of variables to be instantiated to 0 or 1, c is a constant, and $\pi_i, \pi_{i,j}, \pi_{i,j,k}, \ldots$ are the coefficients of items $x_i, x_ix_j, x_ix_jx_k, \ldots$, respectively.

3. DPLL algorithm for maximum satisfiability: a brief review

The Davis–Putnam–Logemann–Loveland (DPLL) algorithm for SAT [11] is a backtracking algorithm that progressively instantiates one variable at a time in searching for a satisfying variable assignment. In each step, the algorithm selects a variable and branches off to assign two possible values, T and F, to the variable. Whenever a clause is violated after setting a variable to T and F, the algorithm backtracks to the 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 a search tree in a depth-first order. DFBnB uses an upper bound α on the minimum total weight of clauses that cannot

be satisfied, whose initial value can be infinity or the value of a sub-optimal solution generated by an approximation algorithm. Starting at the root node, DFBnB always selects a recently generated node to examine next. If all the variables of the current node have been instantiated, and the total weight of clauses violated so far (the g value in the A* algorithm) is less than the current upper bound α, α is revised to the g value; if some variables are still un-instantiated in the current node, and the g value accumulated 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, forbidding any clause violation and resulting in a much reduced search cost. In fact, the 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 minimum total weight of clauses that must be violated in the remaining clauses (the h value in the A* algorithm) at the node exceeds the current upper bound α. This indicates that max-SAT becomes more difficult when the constrainedness increases, causing more clauses unsatisfied and resulting in a higher upper bound α. This also implies that one method to reduce the search cost of DFBnB is to accurately estimate the total weight of the clauses that cannot be satisfied in 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 has motivated our work on LP-based lower bound (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 α can be obtained by an approximation algorithm. A local search algorithm such as WalkSAT [32,38], one of the best local search algorithms for SAT, is a good choice. In our experiments in Section 7, for example, we apply WalkSAT multiple times to reduce the initial upper bound. Such a combination of local search and systematic search is called a two-phase algorithm [4].

3.2. Lower bounds from unit clauses

Another way to improve DPLL on max-SAT is to compute a lower bound on the minimum total weight of clauses that cannot be satisfied at the current node of the search.

3.2.1. Freuder and Wallace's lower bound

One simple lower bound uses only unit clauses. At a node during the search, if the literals in two unit clauses negate each other, one of them must be violated. Let $\mathcal{C}_v$ and $\mathcal{C}_{\bar{v}}$ be the sets of unit clauses with literal v and $\bar{v}$, respectively. Then the minimum weight of violated clauses due to variable v is the smaller of the total weight of the clauses in $\mathcal{C}_{\bar{v}}$ and the total weight of the clauses in $\mathcal{C}_{\bar{v}}$. A lower bound to a node of a search tree

can be obtained by summing up all such minimum weights associated with the variables appearing in all the unit clauses of the node. It has been shown that this simple lower bound can significantly improve the performance of the DPLL algorithm for max-SAT [14]. We adopt this lower bound function in our implementation of DPLL algorithm to deal with all the max-SAT problems except unweighted max-2-SAT.

3.2.2. Shen and Zhang's lower bound for max-2-SAT

Recently, Shen and Zhang proposed several efficient lower bound functions for max-2-SAT [39]. These functions are developed from Freuder and Wallace's lower bound. By analyzing the number of unit clauses and where the literals in the unit clauses appear in binary clauses, Shen and Zhang have deduced three new lower bound functions, LB3, LB4, and LB4a, and shown that they are stronger than Freuder and Wallace's lower bound. The detail of the new lower bounds are left to their original paper [39]. In our implementation of the integrated DPLL algorithm (Section 7.2), we adopted Shen and Zhang's strongest LB4a for solving unweighted max-2-SAT. Note that it is unclear if these new lower bound functions can be extended to max-3-SAT and weighted max-2-SAT.

3.3. Variable ordering

Each step of DPLL chooses a variable to be instantiated next. Strategies for making such choices are referred to as *variable orderings*. The performance of the DPLL algorithm is greatly affected by the variable ordering used.

3.3.1. The two sided Jeroslow–Wang rule

A well-known rule for 3-SAT is the two-sided Jeroslow–Wang (J–W) rule [24]. Let $\{C_1, C_2, \ldots, C_m\}$ be the set of clauses to be satisfied. The two sided J–W rule selects a variable v that maximizes $J(v) + J(\bar{v})$ over all un-instantiated variables, where

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

and n_i the number of literals in the ith clause.

This variable ordering 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 a ratio of 4:2:1 for the weights for variables in unit, binary and three-literal clauses. (It is interesting to note that the idea of progressively halving the weighting factors was used by Johnson [27] thirty years ago in an approximation algorithm for max-SAT.) We call a rule giving different weightings to variables in clauses of different sizes a *weighted variable ordering* or a *weighted branching rule*.

3.3.2. The Mom's rule

Weighted variable ordering has been shown to be very effective for 3-SAT [13,30]. Moreover, experimental results supported the scheme of giving the highest weighting to variables in the shortest clauses [13,30]. This scheme has led to another popular SAT heuristic, the Mom's rule (or the shortest clauses first rule), which branches next on the

variable having the maximum occurrence in the clauses of minimum size [8,30]. The Mom's rule is better than the two-sided J–W rule on 3-SAT [13,30]. In [13,30], the Mom's heuristic was represented as a formula for weighted variable ordering where a clause of length i has a weighting that is 5 times (rather than 2) as large as the weighting of a clause of length $i+1$, namely, the Mom's heuristic selects a variable v that maximizes $J(v) + J(\bar{v})$ over all un-instantiated variables, where

$$J(v) = \sum_{v \in C_i} 5^{-n_i}.$$

The Mom's rule is successful on 3-SAT, because it tends to get rid of unit clauses soon, and forces the lower bound to increase and the search to backtrack early.

3.4. Value ordering

Value ordering, which determines which of the two possible instantiations of a branching variable to be explored first, is another element affecting performance. Different value ordering very often results in different search complexity. Generally, the better a value ordering strategy, the sooner a search process can reach a better solution, if it exists, so that the upper bound can be reduced more quickly and the total search cost will be smaller. On the other hand, if the initial upper bound is the same as the cost of an optimal solution, search is merely to verify that the optimal solution at hand is indeed optimal. In this case, exploring either one of the two branches of a variable instantiation will not affect the number of nodes to be visited in the other branch. Therefore, the effect of value ordering is in large part dominated by an effective initial upper bound strategy, especially one that is able to provide an optimal or nearly-optimal solution. Our experimental analysis supported this observation (data not shown). In our extended DPLL algorithm for max-SAT, because we apply an efficient local search, the WalkSAT algorithm [32,38], to get a good initial upper bound, we do not use any value ordering strategy, i.e., we use a fixed value ordering, first setting a variable to T and then to F.

4. Unit propagation

Unit propagation for SAT, which recursively sets literals in unit clauses to T, 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 satisfies the clause 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 cascade effect, 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 unit clauses having opposite literals, e.g., (v) and $(\bar{v})$, appear in the current formula, the formula is obviously unsatisfiable and the current search avenue 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 force the literal in a unit clause to value T, but rather have to consider setting it to F as well, as long as doing so does not cause the total weight of violated clauses to exceed the current upper bound α. Therefore, unit propagation for SAT in its pure form does not apply to max-SAT.

Nonetheless, the principle of unit propagation can be extended to max-SAT. Indeed, three unit propagation rules have been suggested before by many different groups of researchers. We summarize them all in this section and experimentally analyze them in Section 7. Moreover, we develop a new unit propagation rule that is significantly different from the existing ones. In the rest of this section, we describe these four rules. We experimentally examine their effects and the effect of their combination in Section 7.

To make our presentation of the three existing rules simply, we first introduce some terms. For a max-k-SAT problem where each clause has k literals, consider a node N of a search tree, and an uninstantiated variable v in N. Let g be the total weight of clauses that have been violated at N, and $p_i(v)$ and $n_i(v)$ be the total weights of clauses of i literals in N which have v as positive and negative literals, respectively.

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* [28]. Although an algorithm using this rule can only get a very moderate improvement on SAT [33], experiments done by Wallace showed that improvement of the pure literal rule is considerable for max-2-SAT [42]. We include this rule in our extended DPLL algorithm for max-SAT.

4.2. UP2: Upper bound rule

- Upper bound rule: *If* $p_1(v) + g \geqslant \alpha$, *force* $v = T$ *and ignore* $v = F$; *if* $n_1(v) + g \geqslant \alpha$, *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* $p_1(v) \geqslant \sum_{i=1}^{k} n_i(v)$, *set* $v = T$ *and ignore* $v = F$; *if* $n_1(v) \geqslant \sum_{i=1}^{k} p_i(v)$, *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 [35]. It has been applied to max-2-SAT in [46]. This rule is 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 $v = T$. Nevertheless, for a pedagogical purpose and to simplify the proof to the next,

new propagation rule, we prove UP3 in Appendix A using the nonlinear formulation of max-SAT.

4.4. UP4: Coefficient-determining propagation rule

UP4 is a new unit propagation rule that was derived from a nonlinear programming formulation of max-SAT. It significantly differs from the other three propagation rules; it focuses on individual variables rather than collectively on all clauses of certain lengths. The main idea of UP4 is to infer whether the coefficient F_{x_i} of a (single) variable x_i is nonpositive or nonnegative; if $F_{x_i} \geqslant 0$ or $F_{x_i} \leqslant 0$, we need to fix variable x_i to false or true, respectively, in order to minimize the objective function of the problem. However, to ensure $F_{x_i} \geqslant 0$ or $F_{x_i} \leqslant 0$ is not straightforward, particularly for max-3-SAT and beyond, because F_{x_i} is no longer a linear function of the variables of the problem. To circumvent the difficulty, we consider an upper bound $UB(x_i)$ and a lower bound $LB(x_i)$ of F_{x_i}. If $UB(x_i) \leqslant 0$, F_{x_i} cannot be positive; likewise, if $UB(x_i) \geqslant 0$, F_{x_i} cannot be negative. UP4 can be summarized as follows.

- Coefficient-determining propagation rule: *For each un-instantiated variable v_i and its corresponding integer variable x_i, if $LB(x_i) \geqslant 0$, set $v_i = F$ and ignore $v_i = T$; if $UB(x_i) \leqslant 0$ set $v_i = T$ and ignore $v_i = F$; 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.*

For max-2-SAT,

$$UB(x_i) = \pi_i + \sum_{x_j \in V \setminus \{x_i\}, \pi_{i,j} > 0} \pi_{i,j} \quad \text{and}$$

$$LB(x_i) = \pi_i + \sum_{x_j \in V \setminus \{x_i\}, \pi_{i,j} < 0} \pi_{i,j},$$

where π_i and $\pi_{i,j}$ are as defined in Eq. (1). The detail for the derivation of these bounds is in Appendix B.

For max-3-SAT, it seems to be difficult to obtain closed forms for $UB(x_i)$ and $LB(x_i)$. However, upper and lower bounds can be still computed at each node during the search by simplifying quadratic terms into linear terms. The simplification process involves different ways of turning quadratic terms into linear ones under different conditions to get tight bounds. In other words, the process computes something like piece-wise linear functions. The detail is provided in Appendix B.

Note that simplying a quadratic term to a linear term loosens the tightness of the corresponding bounds, making them less effective. There are also interactions among quadratic terms as well as interactions between quadratic and linear terms, which may further degrade the tightness of the bounds. Their ramification is that the UP4 rule becomes less effective as problem constrainedness increases. Such consequences have been observed in our experiments, including those reported in Section 7.

5. Linear programming based lower bound

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

The new LP lower bound is 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., $x_1, x_2, \ldots, y_1, \ldots$) to integers 0 or 1, but rather allow them to be real values in $[0, 1]$. As a result, we obtain an admissible 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 lower bound estimation with less computation.

However, the application of an LP-based lower bound 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. For example, two such variables in a binary clause can take values slightly more than $1/2$, forcing the auxiliary variable (y variable in the LP formulation, Section 2) for the clause to take value 0, yielding no contribution to the overall lower bound function. Similar scenarios can occur to three-literal clauses. Fortunately, such situations will not occur in unit clauses because decision variables can always contribute to the overall lower bound function even setting literals within unit clauses to "fractional" value. Therefore, *we only apply the LP lower bound to the nodes that have unit clauses*, making it more accurate and more efficient to compute. Moreover, during the search, unit clauses do not need to be eliminated, since the increase in the expected lower bound from eliminating unit clauses has already been calculated exactly by applying the LP lower bound, namely, if we apply the LP lower bound 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.

In principle, applying a stronger lower bound function (i.e., LP-based lookahead lower bound 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 $d = \mathrm{O}(km)$, where k is a constant factor less than 1, and m is the number of constraints. The complexity of extended DPLL is then $T = \mathrm{O}(b^d) = \mathrm{O}(b^{km})$. Using LP-based lower bound, since more nodes can be pruned, the effective branching factor will be reduced to $b_{LP} < b$, and the total node expended will become $\mathrm{O}(b_{LP}^{km})$. However, there is an overhead on the time of computing the LP-based lookahead lower bound. In our implementation of the LP-based lower bound, we used the Simplex algorithm in CPLEX package [26] for LP. Theoretically, the worst-case complexity of the Simplex algorithm is exponential [9]. However, in practice, the Simplex algorithm can efficiently solve most problems in nearly linear time of the dimension of a

problem or the number of constraints encoded in a linear program [16,41]. Therefore, we can consider the overhead for each LP call to be approximately $O(m)$. Thus, the overall time complexity of extended DPLL using an LP-based lower bound 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 lower bound ineffective. This observation may explain why it has been difficult to make LP effective on satisfiability problem instances.

6. Variable ordering, revisited

Variable ordering has not been very well studied for max-SAT. The two most popular variable ordering heuristics, the Mom's rule and the two-side J–W rule (see Section 3.3), were originally developed for SAT. To take advantage of the power of unit propagations in SAT, these rules focus on variables appearing more often in the shortest clauses, which may not be effective for max-SAT. In addition, as our empirical analysis in Section 7 shows, they do not perform well on problems with various constrainedness; neither of these two rules dominates the others under all conditions. Motivated to address these issues, we propose the following two new variable ordering rules for max-2-SAT and max-3-SAT, respectively.

6.1. Binary-clause first variable ordering for max-2-SAT

Due to the efficient Lower bound functions and unit propagation rules in max-2-SAT, the conflicting unit clauses in max-2-SAT, e.g., (v) and $(\bar{v})$, can cause the lower bound to increase without being branched off. Therefore, a plausible strategy for max-2-SAT is to generate, rather than remove, as many unit clauses as possible so as to produce more conflicting unit clauses to increase lower bound. An effective implementation of this strategy is to give a higher weighting to the variables appearing often in binary clauses than the variables in unit clauses, since the instantiation of this variable may give rise to the maximum number of new unit clauses. We call this variable selection rule *binary clauses first rule*. As we will see in Section 7, binary clauses first rule is effective on max-2-SAT; with the weighting ratio of 1:25, binary clause first rule is more effective than the Mom's and the two-side J–W rules in max-2-SAT.

6.2. Dynamic-weighting variable ordering for max-3-SAT

The Mom's and the two-side J–W rules described in Section 3 use static weightings, in that the weighting ratios in the variable ordering are fixed throughout a search regardless of problem constrainedness. As we will see in the next section, in max-3-SAT, they are effective within different ranges of problem constrainedness. Compared to SAT, max-3-SAT can contain problem instances with various constrainedness. These two variable orderings for SAT may not be effective for max-3-SAT. To address this problem, we propose a dynamic-weighting variable ordering.

Dynamical-weighting variable ordering for a max-3-SAT problem with the total number of clauses C and the total number of variables V, in each node of a DFBnB search tree explored by the extended DPLL algorithm, select a variable v that maximizes $J(v) + J(\bar{v})$ over all un-instantiated variables, where

$$J(v) = \sum_{v \in C_i} w_i \beta(r)^{-n_i}$$

n_i is the number of literals in the ith clause C_i, w_i is the weight of the ith clause, r is the clause/variable ratio ($r = C/V$), and

$$\beta(r) = \begin{cases} 5, & \text{if } r < 6.3; \\ 26 - 3.33r, & \text{if } 6.3 \leqslant r \leqslant 7.2; \\ 2, & \text{if } r > 7.2. \end{cases}$$

In the above function, the values 6.3 and 7.2 are determined empirically on randomly generated problem instances (see Section 7 for information of how they were generated). To obtain these empirical values, we tested the different weighting ratios βs on problem instances of different constrainedness, recorded the best β value for each individual constrainedness, and then built a linear function to best fit these data points. Our experiments in Section 7 show that in max-3-SAT, when clause/variable ratio is smaller than 6.3, the Mom's rule performs better; when clause/variable ratio is bigger than 7.2, the two-sided J–W rule works better. Therefore, this method can switch weighting ratio β in variable ordering from that close to the Mom's rule to that similar to the two-side J–W rule as constrainedness increases, thus having good performance in nearly all cases.

7. Experimental evaluation and applications

The combination of the three strategies discussed in Sections 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 [25]. When not explicitly stated otherwise, all our experiments obeyed the following conditions: (15) an initial upper bound for each problem was computed by WalkSAT [32,38] with 10 random restarts and $100 * |V|$ flips per try, where $|V|$ is the number of variables for a given problem instance; (2) all experiments were run on PCs with Pentium 2.4 GHZ processor and 512 MB cache; (3) The LP solver we used to compute the h value was CPLEX 8.0 [26]. 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 all 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 the LP lower bound) in combinations of different variable orderings. We ran three algorithms: DPLL, DPLL with different unit propagation (DPLL + UPs), and DPLL with the LP lower bound (DPLL + LP), where UP and LP stand for unit propagations and the LP lower bound, respectively. Each algorithm was tested with two variable orderings, the Mom's rule and the two-sided J–W rule. In the following figures, the average running time of each experiment was given with a 95% confidence interval.

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 arrangement of constrainedness. As shown in Fig. 1, each UP rule except the UP1 can reduce DPLL's running time by 2–10 times. Detailed running time and speedup for each UP rule are in Tables 1 and 2. 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

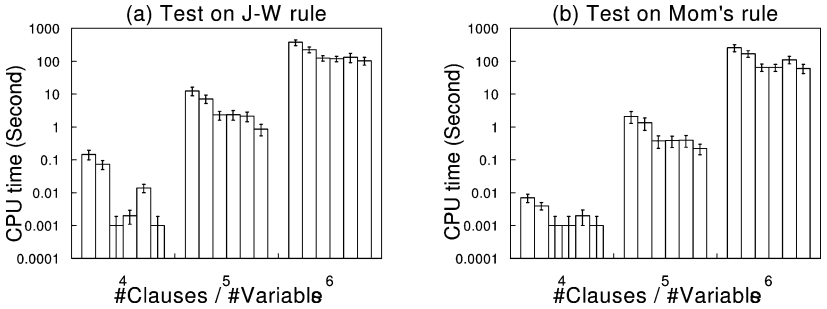

Fig. 1. Effects of unit propagation (UP) rules on unweighted max-3-SAT. For each group of error bars, from left to right are the results from DPLL, DPLL with UP1, UP2, UP3 and UP4, respectively, and DPLL with all four UP rules.

Table 1
Effects of unit propagation (UP) rules on unweighted max-3-SAT, tested on two-sided J–W rule. The running time in seconds is given, followed by its relative speedup (DPLL/DPLL + UPs) in parentheses

C/V	DPLL	DPLL + UP1	DPLL + UP2	DPLL + UP3	DPLL + UP4	DPLL + UP1,2,3,4
4.0	0.148	0.074 (2.0)	0.001 (148.0)	0.002 (74.0)	0.004 (37.0)	**0.001 (148.0)**
4.5	1.477	0.878 (1.7)	0.118 (12.5)	0.144 (10.0)	0.172 (8.6)	**0.039 (37.9)**
5.0	12.554	7.251 (1.7)	2.316 (5.4)	2.372 (5.3)	2.153 (5.8)	**0.870 (14.4)**
5.5	90.411	54.697 (1.7)	23.859 (3.8)	23.410 (3.9)	21.453 (4.2)	**12.950 (7.0)**
6.0	376.390	227.258 (1.6)	123.904 (3.0)	119.434 (3.2)	132.326 (2.8)	**103.533 (3.6)**
6.5	1157.851	704.714 (1.6)	433.563 (2.7)	**431.936 (2.7)**	478.864 (2.4)	465.809 (2.5)
7.0	3643.094	2203.376 (1.6)	1502.187 (2.4)	1496.634 (2.4)	1732.674 (2.1)	**1443.292 (2.5)**
7.5	10005.426	6076.131 (1.6)	4637.295 (2.2)	**4514.648 (2.2)**	5734.217 (1.7)	4533.212 (2.2)
8.0	22153.242	13526.077 (1.6)	11053.094 (2.0)	10656.930 (2.1)	13643.647 (1.6)	**10323.329 (2.1)**

Table 2
Effects of unit propagation (UP) rules on unweighted max-3-SAT, tested on the Mom's rule. The running time in seconds is given, followed by its relative speedup (DPLL/DPLL + UPs) in parentheses

C/V	DPLL	DPLL + UP1	DPLL + UP2	DPLL + UP3	DPLL + UP4	DPLL + UP1,2,3,4
4.0	0.007	0.004 (1.8)	0.001 (7.0)	0.001 (7.0)	0.002 (3.5)	**0.001** **(7.0)**
4.5	0.108	0.073 (1.5)	0.015 (7.2)	0.017 (6.4)	0.034 (3.2)	**0.012** **(9.5)**
5.0	2.110	1.339 (1.5)	0.377 (5.6)	0.384 (5.5)	0.394 (5.4)	**0.221** **(9.4)**
5.5	33.843	21.932 (1.5)	7.229 (4.7)	7.343 (4.6)	10.164 (3.3)	**4.632** **(7.3)**
6.0	261.843	171.548 (1.5)	64.875 (4.0)	64.755 (4.0)	112.176 (2.3)	**60.335** **(4.3)**
6.5	1141.163	717.444 (1.6)	309.602 (3.7)	380.333 (3.0)	419.782 (2.7)	**303.323** **(3.8)**
7.0	5136.232	3928.692 (1.3)	**1859.716** **(2.8)**	1975.385 (2.6)	2912.623 (1.8)	2142.346 (2.4)
7.5	22737.991	17490.939 (1.3)	**9060.268** **(2.5)**	10335.454 (2.2)	12303.489 (1.8)	11016.214 (2.1)
8.0	51183.832	39371.538 (1.3)	29720.075 (1.7)	**26938.421** **(1.9)**	29934.167 (1.7)	32912.432 (1.6)

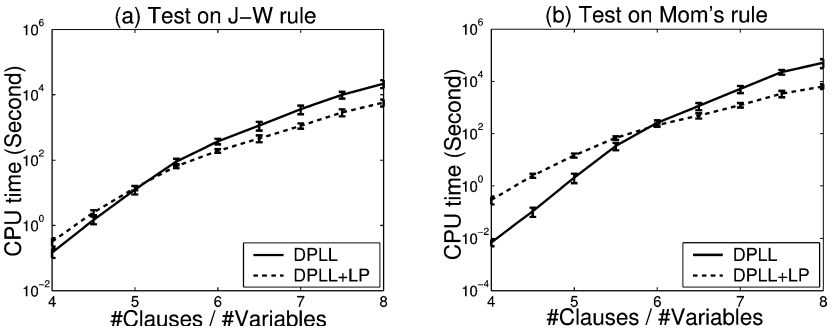

Fig. 2. Effects of the LP lower bound on unweighted max-3-SAT.

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 to the running time. However, as shown in Fig. 2, where we directly compared DPLL + LP and DPLL without unit propagation, the running time overhead of LP is gradually compensated by the amount of pruning it provides as the 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. 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 lower bound 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. As shown in Fig. 3(a), the number of expanded nodes with LP grows more slowly than that without LP. The different growth rates in the number of expanded nodes between using LP and not using LP make DPLL + LP outperform the original DPLL on over-constrained problems.

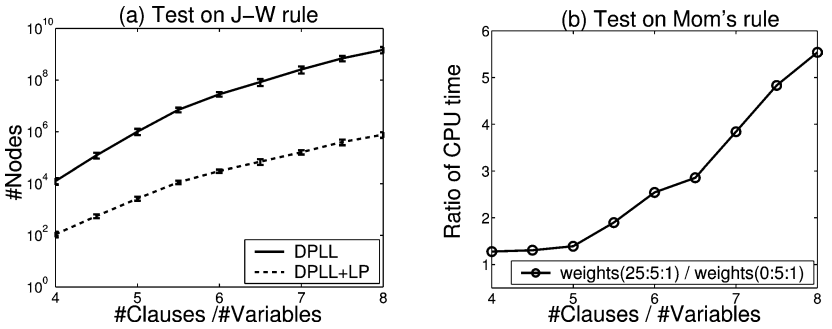

Fig. 3. (a) Expanded nodes of DPLL + LP, (b) effects of not assigning weightings to unit clauses in DPLL + LP. Both (a) and (b) are on unweighted max-3-SAT.

Note that when running DPLL + LP, we modified both Mom's and the two-sided J–W rules. Instead of using weighting ratios of 4:2:1 and 25:5:1, we assigned 0:5:1 as weighting ratio to the Mom's rule and 0:2:1 to the two-sided J–W rule. As discussed in Section 5, we need not 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 weighting" in the Mom's rule is shown in Fig. 3(b). In DPLL + LP, when we change weighting 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 the two-sided J–W rule.

The Mom's and the two-sided J–W rules affect unit propagations and the LP lower bound differently. As shown in Figs. 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 J–W rule as C/V ratio increases. (Note that the vertical axes of the figures are logarithmic, so the actual difference in running time is substantial.) In DPLL and DPLL + UP, the Mom's rule tends to get rid of unit clauses quickly. If the 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 weightings on binary and three literal clauses are smaller than those in the two-sided J–W rule, making it more difficult for non-unit clauses to be turned into unit clauses. Therefore, the Mom's rule performs better than the two-sided J–W rule in under-constrained regions, but worse in over-constrained regions.

In short, our results showed that the Mom's and the two-sided J–W rules are effective under different problem constrainedness. Our new dynamic-weighting variable ordering rule was developed to combine their strengths under different conditions. Moreover, instead of statically setting the weightings, the new rule dynamically adjusts the weightings based on the current situation of the search. As the results in Figs. 4(a) and 4(b) show, the new rule is nearly always the winner under different constraint tightness.

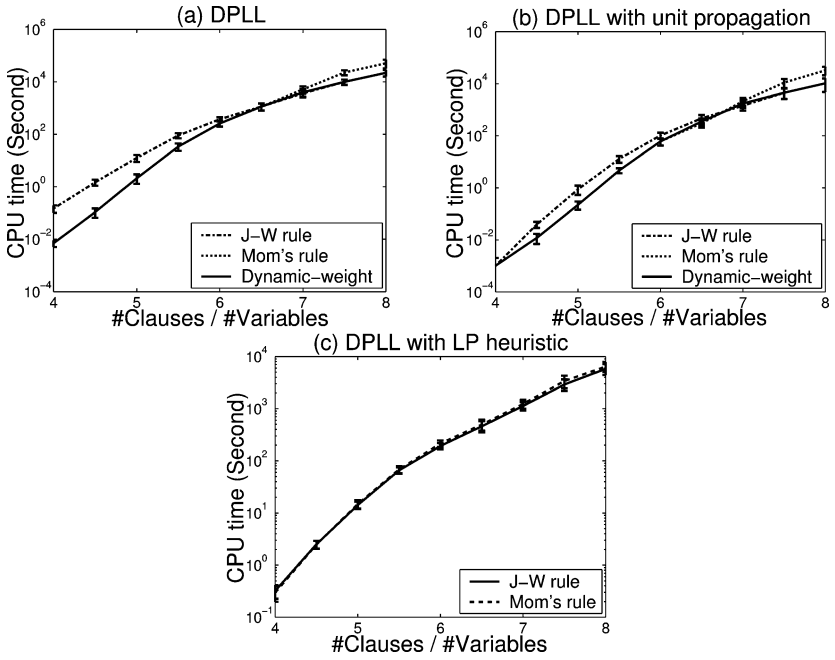

Fig. 4. Effects of different variable orderings on unweighted max-3-SAT.

Compared to DPLL and DPLL + UP, the Mom's and the two-sided J–W rules do not make much difference to DPLL + LP as shown in Fig. 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 contribute to the lower bound. Namely, no matter whether 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 variable ordering 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 constrainedness. Because there are only two literals in each clause, any simplification to a problem formula will result in some unit clauses, which, in turn, make unit propagations happen frequently. In addition, a relatively higher percentage of unit clauses gives rise to higher h values, which make the LP lower bound 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 Fig. 5, unit propagation rules are very effective on all con-

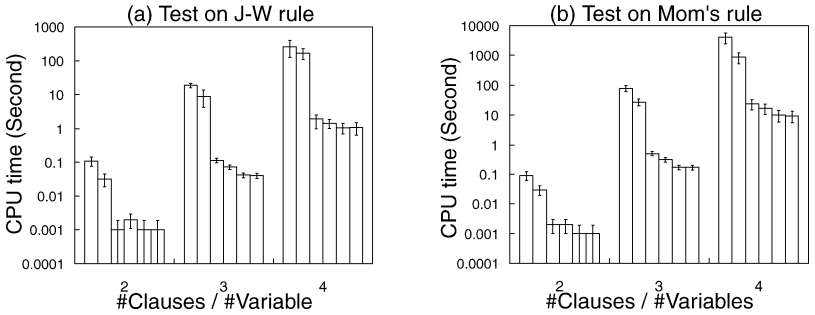

Fig. 5. Effects of unit propagation (UP) rules on unweighted max-2-SAT. For each group of error bars, from left to right are the results from DPLL, DPLL with UP1, UP2, UP3 and UP4, respectively, and DPLL with all four UP rules.

Table 3
Effects of unit propagation (UP) rules on unweighted max-2-SAT, tested on two-sided J–W rule. The running time in seconds is given, followed by its relative speedup (DPLL/DPLL + UPs) in parentheses

C/V	DPLL	DPLL + UP1	DPLL + UP2	DPLL + UP3	DPLL + UP4	DPLL + UP1,2,3,4
2.0	0.109	0.032 (3.4)	0.001 (109.0)	0.002 (54.5)	0.001 (109.0)	**0.001 (109.0)**
2.5	2.166	0.828 (2.6)	0.019 (114.0)	0.013 (166.6)	0.007 (309.4)	**0.007 (309.4)**
3.0	18.820	8.952 (2.1)	0.113 (144.2)	0.073 (257.8)	0.042 (448.1)	**0.040 (470.5)**
3.5	127.651	50.205 (2.5)	0.657 (194.3)	0.432 (295.5)	0.276 (462.5)	**0.254 (502.6)**
4.0	394.274	167.938 (2.3)	1.965 (200.6)	1.449 (272.1)	**1.068 (369.2)**	1.090 (361.7)
4.5	1061.148	482.873 (2.2)	6.284 (168.9)	4.762 (222.8)	4.067 (260.9)	**3.936 (269.6)**
5.0	3086.905	1442.981 (2.1)	20.246 (152.5)	15.846 (194.8)	10.457 (295.2)	**9.634 (320.4)**

Table 4
Effects of unit propagation (UP) rules on unweighted max-2-SAT, tested on the Mom's rule. The running time in seconds is given, followed by its relative speedup (DPLL/DPLL + UPs) in parentheses

C/V	DPLL	DPLL + UP1	DPLL + UP2	DPLL + UP3	DPLL + UP4	DPLL + UP1,2,3,4
2.0	0.090	0.030 (3.3)	0.002 (45.0)	0.002 (45.0)	0.001 (90.0)	**0.001 (90.0)**
2.5	4.377	1.589 (2.8)	0.042 (104.2)	0.028 (156.3)	0.015 (291.8)	**0.014 (312.6)**
3.0	78.910	27.526 (2.8)	0.505 (156.3)	0.310 (254.5)	0.171 (461.5)	**0.169 (466.9)**
3.5	1015.202	234.585 (4.3)	5.795 (175.2)	3.694 (274.3)	2.377 (427.1)	**2.246 (451.9)**
4.0	4058.255	871.081 (4.6)	24.543 (165.4)	17.092 (237.4)	10.064 (403.2)	**9.585 (423.4)**
4.5	8425.353	2161.377 (3.9)	73.602 (114.5)	57.354 (146.9)	42.926 (196.3)	**40.899 (206.0)**
5.0	23822.247	6383.872 (3.7)	261.755 (89.6)	203.894 (116.8)	124.990 (190.6)	**110.876 (216.6)**

strainedness ranges of max-2-SAT. In either variable ordering, each unit propagation rule can independently reduce DPLL's running time by 10–1000 times, and their combination makes the greatest effect under most 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 Tables 3 and 4 for detailed performance of each UP rule.) As shown in Fig. 6, DPLL + LP is also very effective in all constrainedness ranges. For the variable orderings in Fig. 7, although binary clause first rule is the worst one in DPLL experiments, it is the winner for nearly all the situations in DPLL + UP experiments. Since it is DPLL + UP but not DPLL that we will implement in our integrated max-2-SAT algo-

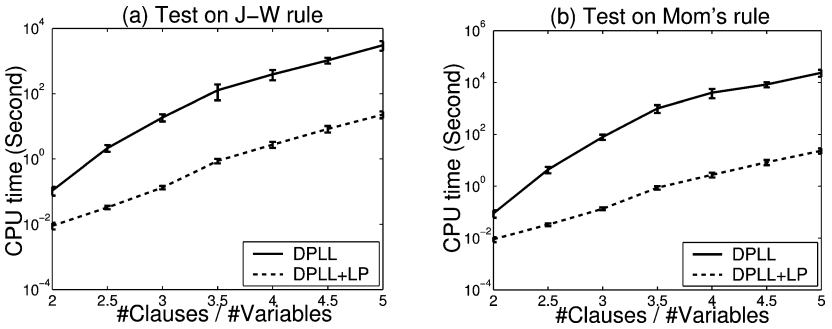

Fig. 6. Effects of the LP lower bound on unweighted max-2-SAT.

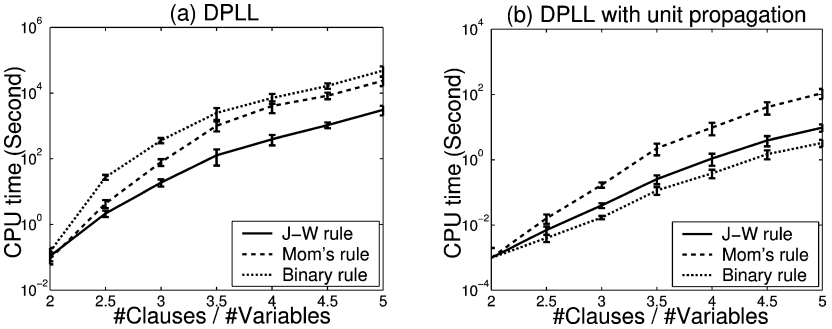

Fig. 7. Effects of different variable orderings on unweighted max-2-SAT.

rithm, we will adopt binary clause first rule for max-2-SAT. All these results suggest that for max-2-SAT, the LP lower bound and all the unit propagation rules should be applied and binary clause first rule is our final choice.

7.1.3. Weighted max-SAT

We used the same set of random max-SAT problems that we experimented with in the unweighted case, except that each clause was given a random integer weighting uniformly distributed between one and ten. We show the results of combined effects of unit propagation rules on weighted max-3-SAT (Fig. 8), and on weighted max-2-SAT (Fig. 9). The results show that our conclusions on unweighted max-SAT are almost equally valid on weighted max-SAT, i.e., unit propagation rules are effective on weighted max-2-SAT or moderately constrained weighted max-3-SAT, LP lookahead lower bound is effective on weighted max-2-SAT or highly constrained weighted max-3-SAT, and the new dynamic-weighting variable ordering is still effective on weighted max-3-SAT. One additional observation is that for the same problem size, weighted problems are usually easier than the corresponding unweighted problems, which can be seen by comparing Figs. 1 and 5 with Figs. 8 and 9, respectively.

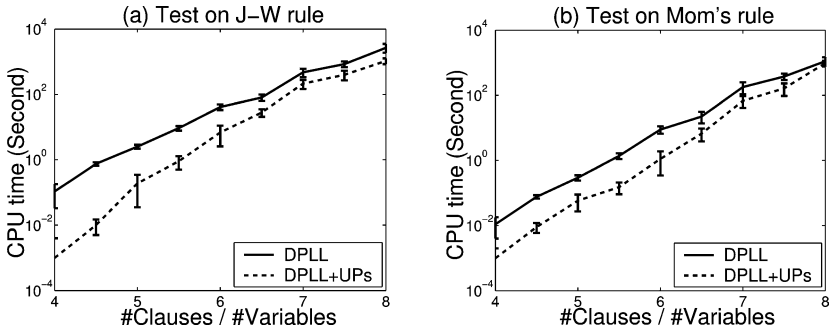

Fig. 8. Effects of unit propagations (UP) on weighted max-3-SAT.

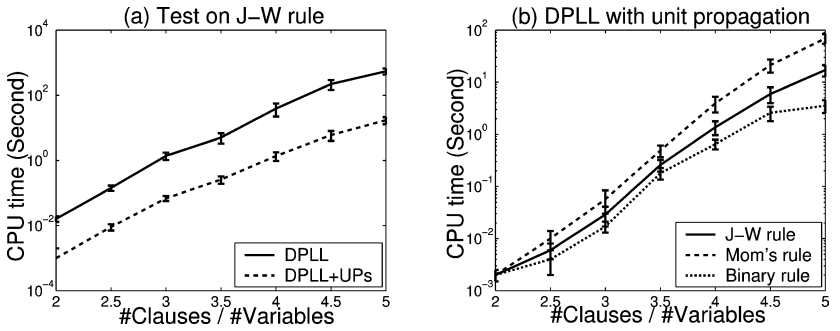

Fig. 9. Effects of (a) unit propagations, (b) variable ordering on weighted max-2-SAT.

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. In our experiments with MaxSolver, we applied the unit propagation rules only to max-2-SAT or moderately constrained max-3-SAT (with c/v ratio of 3 to 6), the LP lookahead lower bound to max-2-SAT or highly constrained max-3-SAT (with c/v ratio more than 6), and our new dynamic-weighting 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:

- A DPLL-based solver BF developed by Borchers and Furman [4].
- A DPLL-based solver AMP developed by Alsinet, Manya, and Planes [2].
- A DPLL-based max-2-SAT solver SZ_LB4a developed by Shen and Zhang [39].
- A Pseudo Boolean Optimization solver PBS2.1 developed by Aloul [1].
- A Weighted CSP-based solver WCSP developed by Givry and Larrosa [17].

These algorithms contain most of the known techniques for SAT, max-SAT and max-CSP. To the best of our knowledge, BF, AMP, and SZ_LB4a 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 [43] 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. AMP is derived from BF and includes a lower bound function described in Section 3.2 and uses the two-sided J–W rule. SZ_LB4a is a specialized max-2-SAT solver with a powerful max-2-SAT lower bound. However, it is not applicable to weighted max-2-SAT. PBS is a specialized 0-1 ILP solver and uses advanced techniques such as conflict diagnosis, random restarts, improved backtracking, and cutting planes. WCSP encodes a max-CSP (and max-SAT) into a weighted constraint network and solves the problem using the state-of-art algorithms for weighted CSP. We used the default settings for all these solvers, except for PBS which used VSIDS decision heuristic [34] (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 unweighted max-SAT instances generated by the MWFF package of Selman [37], random max-SAT instances from Borcher's max-SAT library [5], and unsatisfiable instances in SATLIB [25], which were generated from applications such as planning and model checking. The results are respectively in Tables 5–11, where "–" indicates an incomplete run after 5 hours of CPU time. For each problem class, the tables list either the C/V ratio r or the numbers of variables V and clauses C, followed by columns for the running times of all solvers in seconds. #Unsat in Tables 7, 8, and 11 are the number of violated clauses in unweighted max-SAT, and cost in Tables 9 and 10 are the total weight

Table 5
Average CPU times on unweighted max-2-SAT of 80 variables

C/V	MaxSolver	BF		AMP		PBS		WCSP		SZ_LB4a	
2.0	**0.00**	0.04	(36)	0.07	(66)	3.01	(3013)	0.03	(27)	**0.00**	(–)
2.5	**0.01**	1.21	(207)	1.04	(179)	186.00	(320612)	0.14	(14)	**0.01**	(**1.0**)
3.0	**0.04**	51.79	(1300)	11.87	(298)	–	–	0.57	(14)	0.05	(1.3)
3.5	**0.18**	687.55	(3900)	80.00	(449)	–	–	1.59	(9)	0.34	(1.9)
4.0	**0.85**	12228.00	(14000)	485.10	(575)	–	–	5.80	(7)	1.62	(1.9)
4.5	**3.89**	–	–	2073.52	(532)	–	–	17.28	(4)	8.23	(2.1)
5.0	**13.00**	–	–	4617.56	(355)	–	–	45.47	(3)	32.73	(2.5)

Table 6
Average CPU times on unweighted max-3-SAT of 80 variables

C/V	MaxSolver	BF		AMP		PBS		WCSP	
4.0	**0.00**	**0.00**	(**1.0**)	**0.00**	(**1.0**)	0.01	(16)	0.03	(48.0)
4.5	**0.01**	0.01	(1.0)	1.14	(87.3)	44.90	(3563)	1.18	(90.4)
5.0	**0.15**	0.19	(1.3)	7.43	(50.5)	–	–	6.60	(44.0)
5.5	**4.25**	6.95	(1.6)	64.79	(15.2)	–	–	27.54	(6.5)
6.0	**38.00**	104.00	(2.7)	386.00	(10.2)	–	–	107.25	(2.8)
6.5	**228.00**	629.00	(2.8)	1342.52	(5.9)	–	–	379.49	(1.7)
7.0	1723.00	9498.00	(5.5)	7937.17	(4.6)	–	–	**877.17**	**(0.5)**
7.5	7493.00	–	–	–	–	–	–	**3792.67**	**(0.5)**

Table 7
Computation results for unweighted max-2-SAT test problems from Borcher's library

Instance	$\|V\|$	$\|C\|$	#Unsat	MaxSolver	BF		AMP		PBS		WCSP		SZ_LB4a	
p100	50	100	4	**0.01**	**0.01**	**(1.0)**	0.16	(16.0)	0.06	(6.0)	**0.01**	**(1.0)**	**0.01**	**(1.0)**
p150	50	150	8	**0.01**	0.04	(4.0)	0.07	(7.0)	1.64	(164.0)	**0.01**	**(1.0)**	0.03	(3.0)
p200	50	200	16	**0.02**	4.81	(240.5)	0.83	(41.5)	–	–	**0.02**	**(1.0)**	0.03	(1.5)
p250	50	250	22	**0.02**	28.16	(108.0)	0.57	(28.5)	–	–	**0.02**	**(1.0)**	0.04	(2.0)
p300	50	300	32	0.07	394.09	(5629.9)	10.61	(151.6)	–	–	0.19	(2.7)	**0.06**	**(0.9)**
p350	50	350	41	0.12	2875.61	(23963.4)	22.47	(187.3)	–	–	0.29	(2.4)	**0.10**	**(0.8)**
p400	50	400	45	0.09	2592.49	(28805.4)	9.72	(108.0)	–	–	0.15	(1.7)	**0.06**	**(0.7)**
p450	50	450	63	0.65	–	–	95.81	(147.4)	–	–	1.52	(2.3)	**0.18**	**(0.3)**
p500	50	500	66	0.42	–	–	39.78	(94.8)	–	–	0.80	(1.9)	**0.14**	**(0.3)**
p2200	100	200	5	0.08	0.34	(4.25)	0.88	(11.0)	0.10	(1.3)	0.13	(1.6)	**0.03**	**(0.4)**
p2300	100	300	15	**0.04**	575.69	(14392.3)	106.16	(2654.0)	–	–	1.67	(41.8)	0.33	(8.3)
p2400	100	400	29	**0.32**	–	–	2261.25	(7066.4)	–	–	13.99	(43.7)	0.88	(2.8)
p2500	100	500	44	**11.82**	–	–	–	–	–	–	1539.56	(130.3)	50.72	(4.3)
p2600	100	600	65	106.22	–	–	–	–	–	–	2762.36	(26.0)	**95.64**	**(0.9)**
p2300	150	300	4	**0.06**	0.08	(1.3)	0.51	(8.5)	0.99	(16.5)	1.28	(21.3)	0.07	(1.2)
p2450	150	450	22	**1.93**	–	–	–	–	–	–	154.96	(80.3)	5.59	(2.9)
p2600	150	600	38	**10.41**	–	–	–	–	–	–	2987.56	(287.0)	40.41	(3.9)

Table 8
Computation results for unweighted max-3-SAT test problems from Borcher's library

Instance	$\|V\|$	$\|C\|$	#Unsat	MaxSolver	BF		AMP		PBS		WCSP	
p3250	50	250	2	0.03	**0.01**	**(0.3)**	3.96	(132.0)	0.22	(7.3)	0.02	(0.7)
p3300	50	300	4	0.12	**0.10**	**(0.8)**	3.15	(26.2)	121.30	(1010.8)	0.18	(1.4)
p3350	50	350	8	2.84	6.53	(2.3)	7.81	(2.8)	–	–	**1.12**	**(0.4)**
p3400	50	400	11	12.73	44.71	(3.5)	23.74	(1.9)	–	–	**2.99**	**(0.2)**
p3450	50	450	15	34.49	250.69	(7.3)	42.86	(1.2)	–	–	**3.86**	**(0.1)**
p3500	50	500	15	20.69	150.74	(7.3)	29.84	(1.4)	–	–	**2.48**	**(0.1)**
p3500	100	500	4	8.87	**8.27**	**(0.9)**	–	–	–	–	103.59	(11.7)
p3550	100	550	5	**37.19**	41.01	(1.1)	–	–	–	–	221.97	(6.0)
p3600	100	600	8	**2913.41**	6385.55	(2.2)	–	–	–		3149.86	(1.1)
p3675	150	675	2	8.18	**3.48**	**(0.4)**	–	–	–	–	1419.54	(173.8)
p3750	150	750	5	**2343.04**	2775.47	(1.2)	–	–	–	–	–	–

Table 9
Computation results for weighted max-2-SAT test problems from Borcher's library

Instance	$\|V\|$	$\|C\|$	Cost	MaxSolver	BF		AMP		PBS		WCSP	
wp2100	50	100	16	0.07	0.03	(0.4)	0.04	(0.6)	0.03	(0.4)	**0.01**	**(0.1)**
wp2150	50	150	34	0.09	0.05	(0.6)	0.04	(0.4)	0.68	(7.6)	**0.01**	**(0.1)**
wp2200	50	200	69	0.11	0.58	(5.3)	0.16	(1.5)	220.25	(2002.3)	**0.01**	**(0.1)**
wp2250	50	250	96	0.17	5.97	(35.1)	0.88	(5.2)	–	–	**0.03**	**(0.2)**
wp2300	50	300	132	0.23	22.77	(99.0)	1.26	(5.5)	–	–	**0.04**	**(0.2)**
wp2350	50	350	211	0.92	1078.74	(1172.5)	28.32	(30.8)	–	–	**0.29**	**(0.3)**
wp2400	50	400	211	0.38	532.97	(1402.6)	9.16	(24.1)	–	–	**0.07**	**(0.2)**
wp2450	50	450	257	0.67	1720.42	(2567.8)	8.12	(12.1)	–	–	**0.09**	**(0.1)**
wp2500	50	500	318	1.88	5141.21	(2734.7)	42.30	(22.5)	0.01	–	**0.36**	**(0.2)**
wp2200	100	200	7	0.05	0.10	(2.0)	0.10	(2.0)	–	–	**0.01**	**(0.2)**
wp2300	100	300	67	**0.29**	86.56	(298.5)	23.05	(79.5)	–	–	0.46	(1.6)
wp2400	100	400	119	**6.94**	–	–	3728.47	(537.2)	–	–	15.35	(52.9)
wp2500	100	500	241	532.37	–	–	–	–	–	–	**220.61**	**(0.4)**
wp2600	100	600	266	289.76	–	–	–	–	–	–	**145.37**	**(0.5)**
wp2300	150	300	24	**0.24**	0.43	(1.8)	1.47	(6.1)	11.36	(47.3)	0.94	(3.9)
wp2450	150	450	79	53.48	6857.52	(128.2)	5752.42	(107.6)	–	–	**32.35**	**(0.6)**
wp2600	150	600	189	**3527.52**	–	–	–	–	–	–	5321.10	(1.5)

Table 10
Computation results for the weighted max-3-SAT test problems from Borcher's library

Instance	$\|V\|$	$\|C\|$	Cost	MaxSolver	BF		AMP		PBS		WCSP	
wp3250	50	250	1	0.04	0.06	(1.5)	0.06	(1.5)	**0.01**	**(0.3)**	**0.01**	**(0.3)**
wp3300	50	300	13	**0.07**	0.11	(1.6)	0.14	(2.0)	1.32	(18.9)	**0.07**	**(1.0)**
wp3350	50	350	25	**0.21**	0.69	(3.3)	1.13	(5.4)	510.11	(2429.1)	0.31	(1.5)
wp3400	50	400	33	**0.53**	1.70	(3.2)	3.07	(5.8)	7043.20	(13289.1)	**0.53**	**(1.0)**
wp3450	50	450	35	0.37	1.52	(4.1)	2.03	(5.5)	3053.90	(8253.8)	**0.29**	**(0.8)**
wp3500	50	500	77	21.15	143.44	(6.8)	50.73	(2.4)	–	–	**4.12**	**(0.2)**
wp3500	100	300	6	**0.16**	0.52	(3.2)	–	–	37.37	(233.6)	1.39	(8.7)
wp3600	100	600	26	**45.13**	213.94	(4.7)	–	–	–	–	313.66	(7.0)
wp3675	150	675	2	**0.28**	3.96	(14.1)	–	–	877.06	(3132.4)	8.71	(31.1)
wp3750	150	750	5	**2.17**	17.09	(7.9)	–	–	–	–	94.99	(43.8)

Table 11
CPU times on unsatisfiable SATLIB instances

Instance	$\|V\|$	$\|C\|$	#Unsat	MaxSolver	BF		AMP		PBS		WCSP	
jnh8	100	850	2	**0.01**	0.04	(4.0)	0.32	(32.0)	2.89	(289.0)	1.18	(118.0)
jnh9	100	850	2	**0.02**	0.05	(2.5)	0.32	(16.0)	2.90	(145.0)	1.65	(82.5)
jnh14	100	850	2	**0.01**	0.03	(3.0)	0.31	(31.0)	2.38	(238.0)	3.1	(310.0)
jnh211	100	800	2	**0.01**	0.03	(3.0)	0.31	(31.0)	1.60	(160.0)	0.89	(89.0)
jnh307	100	900	3	**0.02**	0.32	(16.0)	0.76	(38.0)	24.80	(1240.0)	3.97	(198.5)
jnh308	100	900	2	**0.04**	0.06	(1.5)	0.38	(9.5)	6.61	(165.2)	3.59	(89.8)
aim50-2.0no1	50	100	1	0.06	0.02	(0.3)	0.10	(1.7)	**0.01**	**(0.2)**	0.15	(2.5)
aim50-2.0no2	50	100	1	0.03	0.02	(0.7)	0.07	(2.3)	**0.01**	**(0.3)**	0.04	(1.3)
aim50-2.0no3	50	100	1	0.03	0.02	(0.7)	0.09	(3.0)	**0.01**	**(0.3)**	0.06	(2.0)
aim100-1.6no1	100	160	1	649.25	449.55	(0.7)	1047.19	(1.6)	**0.01**	**(0.0)**	670.72	(1.0)
pret60-40	60	160	1	3.27	4.68	(1.4)	10.49	(3.2)	**0.01**	**(0.0)**	85.27	(26.1)
pret60-60	60	160	1	3.35	4.69	(1.9)	10.53	(3.1)	**0.14**	**(0.0)**	85.07	(25.4)
pret60-75	60	160	1	4.12	4.62	(1.1)	10.59	(2.6)	**0.01**	**(0.0)**	84.94	(20.6)
dubois25	75	200	1	16.77	99.31	(5.9)	234.81	(14.0)	**0.21**	**(0.0)**	2358.37	(140.6)
dubois30	90	240	1	2217.63	2947.50	(1.3)	7280.33	(3.3)	**64.42**	**(0.0)**	–	–

of violated clauses in weighted max-SAT. The numbers in parentheses are MaxSolver's relative speedups over the best existing method.

For random unweighted max-2-SAT (Tables 5 and 7), BF degrades quickly as the 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 indicates that the Mom's rule alone is ineffective on max-2-SAT. Maxsolver is also much faster than AMP, which implies that our unit propagation rules can dramatically reduce the node expansions, and that our LP lower bound is effective as well. SZ_LB4a performs the second best for instances from Borcher's library (Table 7), which indicates that SZ_LB4a's special lower bound function is efficient for max-2-SAT. The other two non-DPLL solvers, PBS and WCSP, perform much worse than MaxSolver. PBS is unable to solver problems with more than moderate degree of constrainedness.

For random max-3-SAT (Tables 6 and 8), BF performs better than what it does on max-2-SAT and is sometimes competitive when the C/V ratio is low. However, it still degrades faster than MaxSolver and even AMP as the C/V ratio increases, indicating that not only the Mom's rule on max-3-SAT becomes less effective, but also the LP lower bound becomes effective as the C/V ratio increases. WCSP becomes not as efficient as MaxSolver on max-2-SAT, when the problem size exceeds 100 variables. PBS is not competitive at all on max-3-SAT.

For random weighted max-2-SAT (Table 9) and weighted max-3-SAT (Table 10) instances from Borcher's max-SAT library [5], we compared MaxSolver with BF and WCSP, since the other two algorithms cannot apply. In Table 9, WCSP outperforms MaxSolver and BF on 13 out 17 instances. However, most of the instances that WCSP wins have small sizes and high constrainedness. For large problems with moderate constrainedness, MaxSolver is still the winner. MaxSolver is significantly superior to BF in Table 9, mainly due to the tremendous effects of our new UP4 unit propagation rule. Moreover, UP4 rule becomes increasingly more effective as the constrainedness increases. In Table 10, when the effect of UP4 rule is moderate on weighted max-3-SAT, MaxSolver can still substantially outperform BF and WCSP in all but three cases.

MaxSolver also outperforms the other solvers on many instances from SATLIB. As shown in Table 11, jnh instances are best solved using MaxSolver. For pret instances and dubois25, PBS is the winner. Note that PBS is a few orders of magnitude slower than MaxSolver on jnh instances, each of which has at least two unsatisfiable clauses. This matches the results in Tables 5 and 6, where PBS is the worst on highly over-constrained problems. Therefore, PBS is not suitable for hard max-SAT. worse than our MaxSolver, suitable for low-constrained or special structure instances. WCSP is much worse than MaxSolver on all the instances tested, as it was originally developed for max-CSP. Finally, MaxSolver outperforms BP and AMP on nearly every problem, and solves every one of them in a reasonable amount of time. Therefore, all results indicate that our MaxSolver, although 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 in running time.

8. Related work and discussions

A tremendous amount of research has been devoted to SAT. In this section, we discuss some previous works on max-SAT and exact algorithm for max-SAT.

8.1. Exact algorithms for max-SAT

There are at least three different types of exact algorithms for max-SAT. The most popular among them is an extended DPLL algorithm based on Branch-and-Bound procedure. So far, the known existing DPLL-based max-SAT algorithms include BF [4], AMP [2], and SZ (in which, one of three lower bound functions LB3, LB4, and LB4a can be chosen) [39]. Our MaxSolver belongs to this category. The second type is an OR-based Pseudo Boolean algorithm like PBS [12]. The third type is a weighted CSP-based algorithm like WCSP [17].

Freuder and Wallace carried out an early and significant study of over-constrained satisfaction problems by directly extending the techniques for constraint satisfaction [14,43]. They proposed a number of basic ideas of constructing a DPLL-based exact max-SAT solver, most of which were discussed in Section 3.

In BF algorithm [4], Borchchers and Furman first applied a local search to obtain an initial upper bound for an exact max-SAT algorithm. This idea of obtaining a good initial upper bound has been adopted by nearly every exact max-SAT algorithm. Based on BF algorithm, Alsinet, Manya, and Planes introduced a lower bound function and used the two-sided J–W rule for variable ordering in AMP [2]. In SZ_LB4a, Shen and Zhang developed a novel and very effective lower bound function for max-2-SAT [39]. We extend and improve the DPLL-based max-SAT paradigm in our MaxSolver algorithm in three aspects: unit propagation, lower bound function and variable ordering.

PBS is an OR-based Pseudo Boolean algorithm [12]. It is efficient only on critically constrained problems; its performance degrades greatly on over-constrained problems. WCSP is a weighted CSP-based algorithm [17]. Its performance improves as the constrainedness increases. However, WCSP is still outperformed by MaxSolver on large problems. Moreover, WCSP is more efficient on max-2-SAT than on max-3-SAT.

8.2. Analysis of max-SAT

Niedermeier and Rossmanith analyzed the complexity of a particular exact max-SAT algorithm [35]. They proved that the time complexity of that algorithm is $O(|F| \cdot 1.3803^K)$, where $|F|$ is the total number of literals in a formula F in conjunctive normal form and K is the number of clauses. They also proved a time bounds $O(|F| \cdot 1.3995^k)$, where k is the maximum number of satisfiable clauses, and $O(1.1279^{|F|})$ for the same problem. For max-2-SAT, their results imply a bound of $O(1.2722^k)$.

Zhang studied the relationship between phase transitions of SAT (decision problem) and backbones (variables with fixed values among all optimal solutions) of max-SAT (optimization problem) [47]. His results suggest that the backbone of max-SAT is an order parameter for the problem hardness. Shen and Zhang also studied phase transitions of max-2-SAT [40] and empirically examined the results of phase transitions of [7].

8.3. Unit propagation

The three existing unit propagation rules, UP1, UP2, and UP3, which we summarized in this paper, were considered in many previous studies. The unit propagation rule UP1 and a rule similar to UP2 were studied in [2,42,46]. UP3 was first proposed by Niedermeier and Rossmanith [35], and was applied to max-2-SAT in [46]. Niedermeier and Rossmanith also presented a set of transformation and splitting rules in order to provide a worst case complexity for max-SAT [35]. However, conditions for using most of those rules are too difficult to satisfy. In this paper, we provide an extensive comparative analysis of these rules. Our new unit propagation rule UP4 was developed based on the idea of formulating max-SAT as a nonlinear program. The combination of all these four rules has been shown very efficient in our experiments. Note that the first nonlinear 0-1 formulation of max-SAT was suggested by Hammer and Rudeanu earlier [19].

8.4. Lower bounds and LP and ILP heuristics

Joy, Mitchell, and Borchers are perhaps the first to apply ILP to max-SAT [28]. They showed that an ILP-based solver was able to outperform DPLL-based solvers on max-2-SAT. However, when applied to max-3-SAT, the ILP-based solver was much slower than a DPLL-based algorithm. Blair, Jeroslow and Lowe applied LP to SAT [3]. However, the bounds that they obtained were not tight at all when compared to the bounds from applying ILP. Hooker speculated that better bounds from LP might be possible [22]. In this paper, we proposed to use LP for max-SAT, and successfully showed its power on max-3-SAT for the first time.

8.5. Variable ordering

Little research has been done on variable ordering for max-SAT, except the work in [43] on the effects of applying in-most-unit-clause and in-most-shortest-clause heuristics on small random unweighted max-SAT of 25 variables. Our binary-clause first and dynamic-weighting variable ordering heuristics are novel. The binary-clause first heuristic is able to take advantage of the strong unit propagations and lower bound functions for max-2-SAT problem instances; the dynamic variable ordering heuristic is able to adjust itself according to problem characteristics to cope with different constraint situations for max-3-SAT problem instances.

8.6. Weighted max-SAT

In contrast to the amount of effort devoted to SAT and unweighted max-SAT, research on weighted max-SAT is rather limited. In addition to the BF and WCSP algorithms we compared in this paper, the most relevant previous work is the branch-and-cut algorithm for weighted max-SAT [28]. We did not include this branch-and-cut algorithm in our analysis because it is compatible with the BF algorithm, as discussed in [28].

9. Conclusions and future work

Max-SAT is an important problem that has many real-world applications. However, the existing algorithms for max-SAT are typically restricted to simple problems with small numbers of variables and low constrainedness. The main contributions of this research are a novel unit propagation rule for max-SAT based on an integer nonlinear programming formulation of the problem, an efficient lower bound function based on linear programming, and two effective variable ordering heuristics designed specifically for max-SAT. The key results of this paper are three effective methods for max-SAT and an algorithm that integrates these methods for solving hard max-SAT instances. The three methods are a set of unit propagation rules, a linear-programming based lookahead lower bound, and two new variable ordering rules. We call the new integrated algorithm for max-SAT *MaxSolver*.

We experimentally showed that these new strategies and MaxSolver are effective on different max-2-SAT and max-3-SAT problems. MaxSolver is significantly superior to five existing state-of-the-art algorithms for max-SAT. MaxSolver is able to significantly outperform the existing algorithms, sometimes with orders of magnitude improvement, on many random max-SAT instances and max-SAT instances converted from real application domains.

As our future plan, we will apply MaxSolver to over-constrained real-world applications. For example, the Maximum Probable Explanation (MPE) problem in Bayesian Networks has been formulated as a weighted max-SAT and subsequently solved, approximately, by an approximation max-SAT algorithm [36]. We plan to optimally solve large MPE problems using our new MaxSolver.

Acknowledgements

Many thanks to Zhongsheng Guo for an early implementation of the DPLL algorithm, to Fadi Aloul, Javier Larrosa, Haiou Shen, and Jordi Plane for making their programs available to us for this research, to John Hagar for a careful reading of the paper, and to the reviewers of this paper and an early version in [45] for many constructive comments and suggestions, which helped improve the research and the presentation of the paper. This research was supported in part by US National Science Foundation Grants IIS-0196057 and EIA-0113618 under the ITR program, and in part by US Defense Research Projects Agency and Air Force Research Laboratory, Air Force Material Command, USAF, under Cooperative Agreements F30602-00-2-0531 and F33615-01-C-1897. The views and conclusions herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the US Government.

Appendix A. Proof of dominating unit-clause rule (UP3)

- Dominating unit-clause rule: *If* $p_1(v) \geqslant \sum_{i=1}^{k} n_i(v)$, *set* $v = T$ *and ignore* $v = F$; *If* $n_1(v) \geqslant \sum_{i=1}^{k} p_i(v)$, *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.*

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

Following the discussion on nonlinear formulation of max-SAT in Section 2.2, we can introduce an integer variable that takes value 0 or 1 to present a Boolean variable that takes value F or T. The problem of a max-2-SAT, which may contain unit clauses, is to minimize the objective function

$$W = \sum_{i=1}^{m} w_i y_i, \tag{A.1}$$

where m is the number of clauses, w_i is the weight of the ith clause, and y_i is a decision variable for the ith clause and is subject to the following constraints

$$\begin{cases} y_i = 1 - f_i; & \\ f_i = l_{i1}, & \text{if the } i\text{th clause is a unit clause;} \\ f_i = l_{i1} + l_{i2} - l_{i1} l_{i2}, & \text{if the } i\text{th clause is a binary clause,} \end{cases} \tag{A.2}$$

for $i = 1, 2, \ldots, m$, where l_{i1} and l_{i2} are the 0-1 integers corresponding to the Boolean variables in the ith clause.

The restrictions in (A.2) can be directly used in the objective function in (A.1). Let $\mathcal{C}_k$ be the sets of clauses of k literals, for $k = 1$ or 2, and let c_i be the ith clause. The objective function in (A.1) can be rewritten as

$$W = \sum_{i=1}^{m} w_i(1 - f_i) = \sum_{i=1}^{m} w_i - \sum_{c_i \in \mathcal{C}_2} w_i(l_{i1} + l_{i2} - l_{i1}l_{i2}) - \sum_{c_i \in \mathcal{C}_1} w_i l_{i1}. \tag{A.3}$$

To minimize W, we separate positive and negative literals. Let $\mathcal{C}_k(v_j) \subseteq \mathcal{C}_k$ and $\mathcal{C}_k(\bar{v}_j) \subseteq \mathcal{C}_k$ be the sets of k-literal clauses that contain literal v_j and $\bar{v}_j$, respectively. As discussed in Section 2.1, we can represent positive literal v_i by integer variable x_j and negative literal $\bar{v}_j$ by integer expression $1 - x_j$. Furthermore, Boolean variable v_j may be in unit and binary clauses. We now consider these clauses with v_j in turn.

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

$$-\sum_{c_i \in \mathcal{C}_1(v_j)} w_i l_{i1} = -\sum_{c_i \in \mathcal{C}_1(v_j)} w_i x_j = -p_1(v_j)x_j. \tag{A.4}$$

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

$$\begin{aligned} -\sum_{c_i \in \mathcal{C}_2(v_j)} w_i(l_{i1} + l_{i2} - l_{i1}l_{i2}) &= -\sum_{c_i \in \mathcal{C}_2(v_j)} w_i(x_j + l_{i2} - x_j l_{i2}) \\ &= -\sum_{c_i \in \mathcal{C}_2(v_j)} w_i l_{i2} + \sum_{c_i \in \mathcal{C}_2(v_j)} w_i l_{i2}x_j - p_2(v_j)x_j. \end{aligned} \tag{A.5}$$

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

$$-\sum_{c_i \in \mathcal{C}_1(\bar{v}_j)} w_i l_{i1} = -n_1(v_j) + n_1(v_j)x_j, \tag{A.6}$$

and

$$-\sum_{c_i \in \mathcal{C}_2(\bar{v}_j)} w_i(l_{i1} + l_{i2} - l_{i1}l_{i2}) = -n_2(v_j) - \sum_{c_i \in \mathcal{C}_2(\bar{v}_j)} w_i l_{i2}x_j + n_2(v_j)x_j. \tag{A.7}$$

We now focus on the coefficient F_{x_j} of integer variable x_j. From (A.4) to (A.7), summing up the coefficient of x_j in each case, we have

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

Because $\sum_{c_i \in \mathcal{C}_2(v_j)} w_i l_{i2} \leqslant p_2(v_j)$, and $\sum_{c_i \in \mathcal{C}_2(\bar{v}_j)} w_i l_{i2} \leqslant n_2(x_j)$, we then have

$$n_1(v_j) - p_1(v_j) - p_2(v_j) \leqslant F_{x_j} \leqslant n_1(v_j) + n_2(v_j) - p_1(v_j).$$

If $p_1(v_j) \geqslant n_1(v_j) + n_2(v_j)$, F_{x_j} cannot be positive, thus to minimize the objective function W, x_j should take value 1, i.e., $v_j = T$. If $n_1(v_j) \geqslant p_1(v_j) + p_2(v_j)$, 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.

Appendix B. Derivation of coefficient-determining propagation rule (UP4)

- Coefficient-determining propagation rule: *For each un-instantiated variable v_i and corresponding integer variable x_i, if $LB(x_i) \geqslant 0$, set $v_i = F$ and ignore $v_i = T$; if $UB(x_i) \leqslant 0$ set $v_i = T$ and ignore $v_i = F$; 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.*

To derive rule UP4, we first introduce a lower bound $LB(x_i)$ and an upper bound $UB(x_i)$ for the coefficient F_{x_i} of a variable x_i, for $1 \leqslant i \leqslant n$, i.e., $LB(x_i) \leqslant F_{x_i} \leqslant UB(x_i)$. To derive $LB(x_i)$ and $UB(x_i)$, we first represent the objective function W in such a way that the final nonlinear formula only contains variables x_i. From the objective function in Eq. (1) in Section 2.2, we have

$$W = \sum_{i=1}^{m} w_i y_i = c + \sum_{x_i \in V} \pi_i x_i + \sum_{x_i, x_j \in V} \pi_{i,j} x_i x_j \tag{B.1}$$

for max-2-SAT, and

$$W = \sum_{i=1}^{m} w_i y_i = c + \sum_{x_i \in V} \pi_i x_i + \sum_{x_i, x_j \in V} \pi_{i,j} x_i x_j + \sum_{x_i, x_j, x_k \in V} \pi_{i,j,k} x_i x_j x_k \tag{B.2}$$

for max-3-SAT. We now derive $LB(x_i)$ and $UB(x_i)$ for the coefficient F_{x_i} of x_i. We consider max-2-SAT and max-3-SAT separately.

B.1. Upper and lower bounds for max-2-SAT

For max-2-SAT, from Eq. (B.1), we have

$$\begin{aligned} F_{x_i} &= \pi_i + \sum_{x_j \in V \setminus \{x_i\}} \pi_{i,j} x_j \\ &= \pi_i + \sum_{x_j \in V \setminus \{x_i\}, \pi_{i,j} > 0} \pi_{i,j} x_j + \sum_{x_j \in V \setminus \{x_i\}, \pi_{i,j} < 0} \pi_{i,j} x_j. \end{aligned} \tag{B.3}$$

Note that $\sum_{x_j \in V \setminus \{x_i\}, \pi_{i,j} > 0} \pi_{i,j} x_j \geqslant 0$ and $\sum_{x_j \in V \setminus \{x_i\}, \pi_{i,j} < 0} \pi_{i,j} x_j \leqslant 0$. Therefore, we have

$$\begin{cases} UB(x_i) = \pi_i + \sum_{x_j \in V \setminus \{x_i\}, \pi_{i,j} > 0} \pi_{i,j}, \\ LB(x_i) = \pi_i + \sum_{x_j \in V \setminus \{x_i\}, \pi_{i,j} < 0} \pi_{i,j}, \\ LB(x_i) \leqslant F_{x_i} \leqslant UB(x_i). \end{cases}$$

B.2. Upper and lower bounds for max-3-SAT

For max-3-SAT, from Eq. (B.2), we can write

$$F_{x_i} = \pi_i + \sum_{x_j \in V \setminus \{x_i\}} \pi_{i,j} x_j + \sum_{x_j, x_k \in V \setminus \{x_i\}} \pi_{i,j,k} x_j x_k. \tag{B.4}$$

The main difference between F_{x_i} in Eq. (B.3) and F_{x_i} in Eq. (B.4) is that the former is a linear function of x_j's, while the latter is quadratic.

Since F_{x_i} is quadratic for max-3-SAT, to derive a linear upper bound for F_{x_i} we define a *linear roof* $F^r_{x_i}$ of F_{x_i} which satisfies $F^r_{x_i} \geqslant F_{x_i}$ under all possible assignments. Similarly, we define a *linear floor* $F^f_{x_i}$ satisfying $F^f_{x_i} \leqslant F_{x_i}$ under all possible assignments. Similar definitions of linear roof and linear floor were introduced earlier by Hammer et al. [18]. As a bound on quadratic F_{x_i} is difficult to get, we then use an upper bound of $F^r_{x_i}$ to bound F_{x_i} from above and a lower bound of $F^f_{x_i}$ to bound F_{x_i} from below. In the remaining discussion, we consider how to get $F^r_{x_i}$, $F^l_{x_i}$ and their corresponding bounds.

A simple method to generate $F^r_{x_i}$ (or $F^f_{x_i}$) is to first relax each individual quadratic term $\pi_{i,j,k}x_jx_k$ in F_{x_i} to a linear term, and then sum up all the resulting linear terms. By doing so, we expect some of the linear terms to cancel out each other to arrive at a tighter bound. To relax a quadratic term to linear, we apply a set of inequalities introduced by Hammer et al. [18], specifically,

$$\begin{cases} \pi_{i,j,k}x_jx_k \leqslant \lambda\pi_{i,j,k}x_j + (1-\lambda)\pi_{i,j,k}x_k, & \text{if } \pi_{i,j,k} > 0; \\ \pi_{i,j,k}x_jx_k \leqslant \lambda\pi_{i,j,k}(x_j + x_k - 1), & \text{if } \pi_{i,j,k} < 0; \end{cases} \tag{B.5}$$

and

$$\begin{cases} \pi_{i,j,k}x_jx_k \geqslant \lambda\pi_{i,j,k}(x_j + x_k - 1), & \text{if } \pi_{i,j,k} > 0; \\ \pi_{i,j,k}x_jx_k \geqslant \lambda\pi_{i,j,k}x_j + (1-\lambda)\pi_{i,j,k}x_k, & \text{if } \pi_{i,j,k} < 0, \end{cases} \tag{B.6}$$

where x_j and x_k are 0–1 integer variables and λ is a real value satisfying $0 \leqslant \lambda \leqslant 1$. The proof to these two sets of inequalities are straightforward and can be found in [18]. We can apply inequality (B.5) to relaxing quadratic term $\pi_{i,j,k}x_jx_k$ to a linear term for computing an upper bound and inequality (B.5) for a lower bound. We now consider upper bound first.

Recall that $\pi_{i,j,k}x_jx_k$ is a quadratic term in F_{x_i}, and $\pi_{i,j}x_j$ and $\pi_{i,k}x_k$ are the linear terms involving x_j and x_k. Without loss of generality, assume that $\pi_{i,j} \leqslant \pi_{i,k}$. We can apply inequality (B.5) to relaxing $\pi_{i,j,k}x_jx_k$ to linear function, where λ can be taken as follows,

$$\lambda = \begin{cases} 1, & \text{if } \pi_{i,j,k} > 0 \text{ and } \pi_{i,j,k} \leqslant \pi_{i,k} - \pi_{i,j}; \\ \frac{1}{2} + \frac{\pi_{i,k}-\pi_{i,j}}{2\pi_{i,j,k}}, & \text{if } \pi_{i,j,k} > 0 \text{ and } \pi_{i,j,k} > \pi_{i,k} - \pi_{i,j}\ ; \\ 0, & \text{if } \pi_{i,j,k} < 0 \text{ and } \pi_{i,j} \leqslant 0; \\ \frac{|\pi_{i,j}|}{|\pi_{i,j,k}|}, & \text{if } \pi_{i,j,k} < 0 \text{ and } 0 < \pi_{i,j} < |\pi_{i,j,k}|; \\ 1, & \text{if } \pi_{i,j,k} < 0 \text{ and } \pi_{i,j} \geqslant |\pi_{i,j,k}|. \end{cases} \tag{B.7}$$

The intention behind Eq. (B.7) is to try to cancel (or partially cancel) some resulting linear term with an existing linear term in F_{x_i} as much as possible so as to get a tight linear upper bound $F^r_{x_i}$. If $\pi_{i,j,k} > 0$, one of the first two values in Eq. (B.7) is used, otherwise, one of the last three values is chosen.

By applying λ in Eq. (B.7) to inequality (B.5) and summing up all terms for different x_j and x_k, we then obtain a linear function $F^r_{x_i}$ such that $F_{x_i} \leqslant F^r_{x_i}$. Linear function $F^r_{x_i}$ can be written as $F^r_{x_i} = \pi^r_i + \sum_{x_j \in V\setminus\{x_i\}} \pi^r_{i,j}x_j$. Following the discussion for deriving bounds for max-2-SAT in Section B.1, we take $UB(x_i) = \pi^r_i + \sum_{x_j \in V\setminus\{x_i\}, \pi^r_{i,j}>0} \pi^r_{i,j}$ and thus have $F_{x_i} \leqslant F^r_{x_i} \leqslant UB(x_i)$.

The case for a lower bound is similar and symmetric to the case for the upper bound considered above. Specifically, without loss of generality, assuming that $\pi_{i,j} \leqslant \pi_{i,k}$, we choose λ as follows,

$$\lambda = \begin{cases} 0, & \text{if } \pi_{i,j,k} > 0 \text{ and } \pi_{i,k} \geqslant 0; \\ \frac{|\pi_{i,k}|}{\pi_{i,j,k}}, & \text{if } \pi_{i,j,k} > 0,\ \pi_{i,k} < 0, \text{ and } |\pi_{i,k}| \leqslant \pi_{i,j,k}; \\ 1, & \text{if } \pi_{i,j,k} > 0,\ \pi_{i,k} < 0, \text{ and } |\pi_{i,k}| > \pi_{i,j,k}; \\ 0, & \text{if } \pi_{i,j,k} < 0 \text{ and } |\pi_{i,j,k}| \leqslant \pi_{i,k} - \pi_{i,j}; \\ \frac{1}{2} - \frac{\pi_{i,k} - \pi_{i,j}}{|2\pi_{i,j,k}|}, & \text{if } \pi_{i,j,k} < 0 \text{ and } |\pi_{i,j,k}| > \pi_{i,k} - \pi_{i,j}. \end{cases} \tag{B.8}$$

By applying λ from Eq. (B.8) to inequality (B.6), we write $F^f_{x_i} = \pi^f_i + \sum_{x_j \in V \setminus \{x_i\}} \pi^f_{i,j} x_j$ and have $F^f_{x_i} \leqslant F_{x_i}$. By the same reasoning for the lower bound for max-2-SAT, we have $LB(x_i) = \pi^f_i + \sum_{x_j \in V \setminus \{x_i\}, \pi^f_{i,j} < 0} \pi^f_{i,j}$ and $LB(x_i) \leqslant F^f_{x_i} \leqslant F_{x_i}$.

Putting all the pieces together, we finally have

$$\begin{cases} UB(x_i) = \pi^r_i + \sum_{x_j \in V \setminus \{x_i\}, \pi^r_{i,j} > 0} \pi^r_{i,j}, \\ LB(x_i) = \pi^f_i + \sum_{x_j \in V \setminus \{x_i\}, \pi^f_{i,j} < 0} \pi^f_{i,j}, \\ LB(x_i) \leqslant F_{x_i} \leqslant UB(x_i). \end{cases}$$

B.3. Coefficient-determining propagation rule

For the above $UB(x_i)$ and $LB(x_i)$ in max-2-SAT and max-3-SAT problems, if $UB(x_i) \leqslant 0$, F_{x_i} cannot be positive, thus to minimize the objective $W = \sum_{i=1}^{m} w_i y_i$, x_i should take value 1, i.e., $v_i = T$. If $LB(x_i) \geqslant 0$, F_{x_i} cannot be negative, to minimize W, x_i should take value 0, i.e., $v_i = F$. This concludes the proof.

References

[1] F. Aloul, A. Ramani, I. Markov, K. Sakallah, Generic ILP versus specialized 0-1 ILP: an update, in: International Conference on Computer Aided Design (ICCAD), 2002.
[2] T. Alsinet, F. Manya, J. Planes, Improved branch and bound algorithms for Max-SAT, in: Proceedings of SAT2003, 2003.
[3] C.E. Blair, R.G. Jeroslow, J.K. Lowe, Some results and experiments in programming techniques for propositional logic, Comput. Oper. Res. 13 (5) (1986) 633–645.
[4] B. Borchchers, J. Furman, A two-phase exact algorithm for Max-SAT and weighted Max-SAT problems, J. Combin. Optim. 2 (4) (1999) 299–306.
[5] http://www.nmt.edu/~borchers/maxsat.html.
[6] S. Cook, The complexity of theorem-proving procedures, in: Proceedings of the 3rd ACM Symposium on the Theory of Computing, 1971, pp. 151–158.
[7] D. Coppersmith, D. Gamarnik, M. Hajiaghayi, G.B. Sorkin, Random MAX SAT, random MAX CUT, and their phase transitions, in: Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms, 2003, pp. 364–373.
[8] J.M. Crawford, L.D. Auton, Experimental results on the crossover point in satisfiability problems, in: Proceedings of AAAI-93, Washington, DC, 1993, pp. 21–27.
[9] G.B. Dantzig, M.N. Thapa, Linear Programming 1: Introduction, Springer, New York, 1997.
[10] M. Davis, G. Logemann, D. Loveland, A machine program for theorem proving, C. ACM 5 (1962) 394–397.

[11] M. Davis, H. Putnam, A computing procedure for quantification theory, J. ACM 7 (1960) 201–215.
[12] H. Dixon, M.L. Ginsberg, Inference methods for a pseudo-boolean satisfiability solver, in: Proceedings of AAAI-02, Edmonton, AB, 2002, pp. 635–640.
[13] J.W. Freeman, Improvements to propositional satisfiability search algorithms, PhD thesis, Univ. of Pennsylvania, 1995.
[14] E.C. Freuder, R.J. Wallace, Partial constraint satisfaction, Artificial Intelligence 58 (1992) 21–70.
[15] M.R. Garey, D.S. Johnson, Computers and Intractability, Freeman, New York, 1979.
[16] P.E. Gill, W. Murray, M.A. Saunders, J.A. Tomlin, M.H. Wright, On projected Newton barrier methods for linear programming and an equivalence to Karmarkar's projective method, Math. Programming 26 (1986) 183–209.
[17] S.D. Givry, J. Larrosa, P. Meseguer, T. Schiex, Solving Max-SAT as weighted CSP, in: Proceedings of 9th International Conference on Principles and Practice of Constraint Programming (CP2003), 2003, pp. 363–376.
[18] P.L. Hammer, P. Hansen, B. Simeone, Roof duality, complementation and persistency in quadratic 0-1 optimization, Math. Programming 28 (1984) 121–155.
[19] P.L. Hammer, S. Rudeanu, Boolean Methods in Operations Research and Related Areas, Springer, Berlin, 1968.
[20] P. Hansen, B. Jaumard, Algorithm for the maximum satisfiability problem, Computing 44 (1990) 279–303.
[21] F.S. Hillier, G.J. Lieberman, Introduction to Operations Research, McGraw-Hill, New York, 2001.
[22] J.N. Hooker, Input proofs and rank one cutting-planes, ORSA J. Comput 1 (1989) 137–145.
[23] J.N. Hooker, G. Fedjki, Branch-and-cut solution of inference problems in prepositional logic, Ann. Math. Artificial Intelligence 1 (1990) 123–139.
[24] J.N. Hooker, V. Vinay, Branching rules for satisfiability, J. Automated Reasoning 15 (1995) 359–383.
[25] H.H. Hoos, T. Stuzle, http://www.satlib.org, 1999.
[26] http://www.ilog.com/products/cplex.
[27] D.S. Johnson, Approximation algorithms for combinatorial problems, J. Comput. System Sci. 9 (1974) 256–278.
[28] S. Joy, J. Mitchell, B. Borchers, A branch and cut algorithm for Max-SAT and weighted Max-SAT, in: D. Du, J. Gu, P.M. Pardalos (Eds.), Satisfiability Problem: Theory and Applications, 1997, pp. 519–536.
[29] R. Karp, Reducibility among combinatorial problems, in: R. Miller, J. Thatcher (Eds.), Complexity of Computer Computations, Plenum Press, New York, 1992, pp. 85–103.
[30] C.M. Li, Anbulagan, Heuristics based on unit propagation for satisfiability problems, in: Proceedings of IJCAI-97, Nagoya, Japan, 1997, pp. 366–371.
[31] C.L. Liu, Introduction to Combinatorial Mathematics, McGraw-Hill, New York, 1968.
[32] D. McAllester, B. Selman, H. Kautz, Evidence for invariants in local search, in: Proceedings of AAAI-97, Providence, RI, 1997, pp. 321–326.
[33] D.B. Mitchell, B. Selman, H. Levesque, Hare and easy distributions of SAT problems, in: Proceedings of AAAI-93, Washington, DC, 1993, pp. 459–465.
[34] M. Moskewics, C. Madigan, Y. Zhao, L. Zhang, S. Malik, Chaff: engineering an efficient SAT solver, in: Proceedings of the Design Automation Conference, 2001.
[35] R. Niedermeier, P. Rossmanith, New upper bounds for maximum satisfiability, J. Algorithm 36 (2000) 63–88.
[36] J.D. Park, Using weighted MAX-SAT engines to solve MPE, in: Proceedings of AAAI-02, Edmonton, AB, 2002, pp. 682–687.
[37] B. Selman, Mwff: Program for generating random max k-SAT instances, Available from DIMACS.
[38] B. Selman, H. Kautz, B. Cohen, Noise strategies for local search, in: Proceedings of AAAI-94, Seattle, WA, 1994, pp. 337–343.
[39] H. Shen, H. Zhang, Study of lower bound functions for Max-2-SAT, in: Proceedings of AAAI-02, Edmonton, AB, 2002, pp. 185–190.
[40] H. Shen, H. Zhang, An empirical study of Max-2-SAT phase transitions, in: Proceedings of LICS Workshop on Phase Transitions, 2003.
[41] D.A. Spielman, S. Teng, Smoothed analysis of algorithms: why the simplex algorithm usually takes polynomial time, J. ACM 51 (1) (2004) 385–463.

[42] R.J. Wallace, Enhancing maximum satisfiability algorithms with pure literal strategies, in: 11th Canadian Conf. on AI, 1996.
[43] R.J. Wallace, E.C. Freuder, Comparative study of constraint satisfaction and Davis–Putnam algorithms for maximum satisfiability problems, in: D. Johnson, M. Trick (Eds.), Cliques, Coloring, and Satisfiability, 1996, pp. 587–615.
[44] J.P. Walser, Integer Optimization Local Search, Springer, Berlin, 1999.
[45] Z. Xing, W. Zhang, Efficient strategies for (weighted) maximum satisfiability, in: Proceedings of 10th International Conference on Principles and Practice of Constraint Programming (CP2004), 2004, pp. 690–705.
[46] H. Zhang, H. Shen, F. Manya, Exact algorithms for Max-SAT, in: Workshop on First-Order Theorem Proving (FTP-03), 2003.
[47] W. Zhang, Phase transitions and backbones of 3-SAT and maximum 3-SAT, in: Proceedings of 7th International Conference on Principles and Practice of Constraint Programming (CP2001), 2001, pp. 155–167.