# The Power of Semidefinite Programming Relaxations for MAX-SAT

Carla P. Gomes, Willem-Jan van Hoeve, and Lucian Leahu

Dpt. of Computer Science, Cornell University, Ithaca, NY 14853, USA,
{gomes,vanhoeve,lleahu}@cs.cornell.edu

**Abstract.** Recently, Linear Programming (LP)-based relaxations have been shown promising in boosting the performance of exact MAX-SAT solvers. We compare Semidefinite Programming (SDP) based relaxations with LP relaxations for MAX-2-SAT. We will show how SDP relaxations are surprisingly powerful, providing much tighter bounds than LP relaxations, across different constrainedness regions. SDP relaxations can also be computed very efficiently, thus quickly providing tight lower and upper bounds on the optimal solution. We also show the effectiveness of SDP relaxations in providing heuristic guidance for iterative variable setting, significantly more accurate than the guidance based on LP relaxations. SDP allows us to set up to around 80% of the variables without degrading the optimal solution, while setting a single variable based on the LP relaxation generally degrades the global optimal solution in the overconstrained area. Our results therefore show that SDP relaxations may further boost exact MAX-SAT solvers.

## 1 Introduction

In recent years, we have witnessed a tremendous progress in the state-of-the-art of encodings and algorithms for Boolean Satisfiability (SAT). For example, in areas such as planning and finite model-checking, we are now able to solve large SAT problems with up to a million variables and five million constraints. More generally, SAT encodings have been shown to be very powerful in several practical domains, such as electronic design automation, AI planning, and hardware and software verification. The key algorithmic improvements that have been incorporated into state-of-the-art SAT solvers have been largely based on artificial intelligence (AI) and constraint programming (CP) techniques. For example, for complete solvers, the underlying backtrack search strategy has been enhanced by a series of increasingly sophisticated techniques, such as non-chronological backtracking, fast pruning and propagation methods, nogood (or clause) learning, and more recently randomization and restarts. While we have recently seen an increasing dialogue between the artificial intelligence (AI) and constraint programming (CP) community and the Operations Research community concerning the study and design of algorithms for SAT and variants, it has been surprisingly difficult to integrate OR based relaxations into practical approaches for

SAT. For example, despite a significant amount of beautiful Linear Programming (LP) results for SAT (see e.g., [11, 18]), practical state-of-the-art solvers do not incorporate LP relaxation techniques. The main reason seems to be the fact that, in the case of SAT, the inference performed by LP is basically equivalent to the inference performed by unit propagation, which is considerably less expensive than LP.[1] Nevertheless, when it comes to MAX-SAT, the optimization counterpart of SAT in which the objective is to assign values to boolean variables maximizing the number of satisfied clauses, there seems to be a more clear role for hybrid approaches that combine AI and OR based techniques. In fact, recently Xing and Zhang [19] made an interesting contribution in the area of hybrid approaches for MAX-SAT, showing how one can use the information provided by linear programming to effectively compute lookahead lower bounds on the number of clauses unsatisfiable. Joy et al. ([12]) have also shown how LP-based relaxations can be effective for MAX-2-SAT.

Another area that has received considerable attention in combinatorial optimization is *Semidefinite Programming* (SDP). In a semidefinite programming formulation a linear function of a symmetric matrix is optimized, subject to linear equality constraints and the constraint that the matrix be positive semidefinite. Semidefinite programming is a special case of convex programming and to some extent is similar to linear programming. In particular, the simplex, ellipsoid, and interior point methods developed for LP can be generalized to solve SDP programs. Furthermore, the rich set of LP results in dual theory, a powerful tool for sensitivity analysis and for computing bounds on the objective function have also been generalized to SDP [1, 7]. Moreover, SDP has gained considerable importance in the context of combinatorial optimization since it has been shown that it leads to tighter relaxations than those based on LP for several combinatorial problems. For example, SDP has been shown to provide very good approximations for several combinatorial problems, in particular for the stable set problem [9], for the maximum cut problem and for MAX-SAT [8]. Approximation algorithms are procedures that provide a feasible solution in polynomial time (see e.g. [10]). A key aspect that characterizes approximation algorithms is the fact that they provide some guarantee on the quality of the solution. The quality of an approximation algorithm is the maximum "distance" between its solutions and the optimal solutions, evaluated over all the possible instances of the problem. In a seminal paper, Goemans and Williamson [8] used SDP to obtain improved approximations for the Max-Cut and the MAX-2-SAT problem. In this work they present a randomized approximation algorithm for MAX-2-SAT that produces solutions of expected value at least .87856 times the optimal value. Subsequently, Feige and Goemans [17] extended this work, developing an .931-approximation algorithm for MAX-2-SAT. In our experiments we apply the "classical" SDP relaxation of Goemans and Williamson [8].

In this paper we study the quality of SDP based relaxations for MAX-SAT. In particular, we are interested in comparing the power of SDP relaxations against

[1] Unit propagation recursively sets the literals corresponding to unit clauses to true, eliminating all the clauses in which the literal appear, until a fix-point is reached.

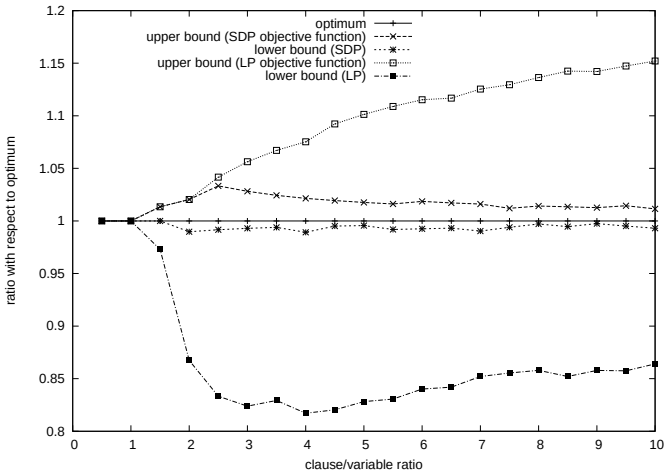



**Fig. 1.** Lower and upper bounds based on LP and SDP relaxations

LP based relaxations, since LP relaxations have been shown to be useful in speeding up practical solvers [19]. In the work reported in this paper we address the following research questions: (1) *How do the LP and SDP solutions compare as upper bounds on the optimal solution?* This is a critical question since we can use the relaxations as admissible heuristics to prune the search space. As we will see, in the overconstrained area, the upper bound based on the SDP relaxation is considerably tighter than the one provided by the LP relaxation: the upper bound provided by LP for MAX-SAT instances is always equal to the total number of clauses, therefore not informative in the overconstrained area; the upper bound given by the SDP relaxation is surprisingly close to the optimal solution (within less than 3% of the real optimal value, when varying the ratio of clauses to variables up to 10) (2) *How do the assignments based on the LP and SDP relaxations compare as lower bounds on the optimal solution?* Once again we see that the SDP relaxation outperforms the LP relaxation considerably, especially in the overconstrained area. In fact while the SDP lower bound is always within 1% of the optimal solution, the lower bound provided by the LP relaxation can be as far as 18% from optimal (see figure 1). We note that the LP solver runs faster than the SDP solver. Nevertheless, the runtimes for the SDP solver are very good, a little over 1 second per instance, on average, for an 80 variable problem, independently of the number of clauses. We also compared the quality of the solutions obtained from the SDP relaxation as a lower bound on the optimal solution against Walksat, one of the best performing local search methods for MAX-SAT. We gave Walksat about 5 minutes per instance (note that the SDP solver takes less than 2 seconds per instance). Interestingly, as the instances become more and more overconstrained, the SDP solutions become better than those provided by Walksat. Furthermore, because Walksat is a local

search solver, it does not provide an upper bound on the optimal solution, a key aspect of the SDP relaxation. (3) *To what extent the relaxations provide a global perspective of the search space and therefore to what extent they can be used as heuristics to guide a complete solver?* In order to address this issue we performed the following experiment: set the X highest values suggested by the LP/SDP relaxation; check if the optimal value of the resulting instance is still the same as the original optimal value. Once again the SDP relaxation clearly outperforms the LP relaxation. In fact, in the overconstrained area, the setting of a single value dictated by the LP relaxation generally results in a value for the optimal solution lower than the original value. The SDP relaxation on the other hand is much more robust: We can set up to an average of 84% of variables based on the SDP suggestions, without changing the value of the optimal solution. This result suggests that the SDP relaxation can be a very valuable heuristic for setting variable values in a backtrack search strategy.

## 2 Preliminaries

The Boolean satisfiability problem (SAT) is a decision problem at the core of complexity theory, artificial intelligence, logic and hardware design and verification. We consider the problem in conjunctive normal form (CNF). A formula $F$ in CNF is a conjunction of clauses, where each clause is a disjunction of literals. Each literal is a logical variable ($x$) or its negation ($\bar{x}$). The SAT problem is to determine whether there exists a variable assignment that makes the formula true (i.e., each clause is true). $k$-SAT represents the satisfiable problem where the clauses are constrained to have the length equal to $k$.

MAX-SAT is the optimization version of SAT. Given a formula we want to maximize the number of simultaneously satisfied clauses. Given an algorithm for MAX-SAT we can solve SAT, but not vice-versa, therefore MAX-SAT is more complex than SAT. The distinction becomes obvious when considering the case when the clauses are restricted to two literals per clause (2-SAT): 2-SAT is solvable in linear time, while MAX-2-SAT is NP-hard [6].

Given the importance of the problem, the complexity of SAT has received much attention. Previous results show easy-hard-easy patterns in terms of the problem hardness, as a function of the clause/variable ratio ($C/V$). Furthermore, phase-transition phenomena have been reported with respect to satisfiability (i.e., a sudden change from many satisfiable instances to none). For instance, for 2-SAT this phase transition has been proven to occur when $C/V$ is 1 [4].

Recently there have been promising results when using LP for MAX-SAT, after several unsuccessful efforts to apply integer LP to MAX-SAT (e.g.,[11, 12]). Xing and Zhang developed MaxSolver ([19]), an efficient exact algorithm for (weighted) MAX-SAT. Their solver uses a DPLL-based branch and bound algorithm and it successfully uses a lookahead LP lower bound. This lower bound is only applied to the nodes that have unit clauses, to avoid fractional values equal to 1/2 (in the case of MAX-2-SAT). MaxSolver also incorporates existing and

novel unit propagation rules, a binary-clause first rule and a dynamic-weighting variable ordering rule.

SDP relaxations have also been deployed for the MAX-2-SAT problem. Following the randomized polynomial time algorithm of Goemans and Williamson [8] which had an approximation ratio of 0.87856, there has been a series of theoretical results improving this approximation ratio. The most recent improvement is a 0.940-approximation algorithm [13]. A thorough survey of SDP-based approximation algorithms for the MAX-SAT problem is presented in [2]. See [5] for interesting results on the SDP for the so-called 2+p-SAT problem.

## 3 LP and SDP Formulations for MAX-SAT

### 3.1 LP Formulation

We consider the following ILP formulation for the MAX-SAT problem from [14]. With each clause $C_j$ we associate a variable $z_j \in \{0,1\}$. 1 corresponds to the clause being satisfied and 0 to the clause not being satisfied. For each variable $x_i$ we associate a corresponding variable $y_i$ in the ILP. $y_i$ can take the values 0 and 1, corresponding to $x_i$ being false or true, respectively. Let $C_j^+$ be the set of indices of positive literals that appear in clause $C_j$, and $C_j^-$ be the set of indices of negative literals (i.e., complemented variables) that appear in clause $C_j$. The problem can be formally stated as follows:

$$\begin{aligned} \max & \sum_{j=1}^{m} z_j \\ \text{subject to } & \sum_{i \in C_j^+} y_i + \sum_{i \in C_j^-} (1 - y_i) \geq z_j, \ \forall j \\ & y_i, z_j \in \{0,1\}, \qquad \forall i, j. \end{aligned}$$

The formulation ensures that a clause is true only if at least one of the variables that appear in the clause has the value 1. Since, we want to maximize $\sum_{j=1}^{m} z_j$ and $z_j$ can be set to 1 only when clause $C_j$ is satisfied, it follows that the objective function counts the number of satisfied clauses. By relaxing the integrality constraint, we obtain an LP relaxation for the MAX-SAT problem. This ILP formulation is equivalent to the ILP used in [19] to compute the lower bound and to the ILP solved at each node by the MAX-SAT branch and cut algorithm in [12].

It is interesting to note that there exists a trivial way to satisfy all the clauses: setting each variable $y_i$ to 0.5. Using this assignment, the sum of literals for each clause is exactly 1, hence the clause can be satisfied and the objective function is equal to the number of clauses. The value 0.5 is not at all informative, lying half way between 0 and 1, it gives no information whether the corresponding Boolean variable should be set to true or false. As the problem becomes more constrained (i.e., the number of clauses increases) the corresponding 2-SAT problem is very likely to be unsatisfiable, hence any variable assignment different than 0.5 would lead to a less than optimal objective value. Naturally, the LP solver finds the highest possible objective value (i.e., the number of clauses) when setting all variables to 0.5.

### 3.2 Semidefinite Programming

In this section we briefly introduce semidefinite programming. A general introduction to semidefinite programming applied to combinatorial optimization is given in, e.g., [7]. Semidefinite programming makes use of positive semidefinite matrices of variables. A matrix $X \in \mathbb{R}^{n \times n}$ is said to be positive semidefinite (denoted by $X \succeq 0$) when $y^{\mathsf{T}} X y \geq 0$ for all vectors $y \in \mathbb{R}^n$. Semidefinite programs have the form

$$\begin{aligned} &\max \operatorname{tr}(WX) \\ &\text{s.t. } \operatorname{tr}(A_j X) \leq b_j \ (j = 1, \ldots, m) \\ &\quad X \succeq 0. \end{aligned} \tag{1}$$

Here $\operatorname{tr}(X)$ denotes the trace of $X$, which is the sum of its diagonal elements, i.e. $\operatorname{tr}(X) = \sum_{i=1}^{n} X_{ii}$. The matrix $X$, the cost matrix $W \in \mathbb{R}^{n \times n}$ and the constraint matrices $A_j \in \mathbb{R}^{n \times n}$ are supposed to be symmetric. The $m$ reals $b_j$ and the $m$ matrices $A_j$ define $m$ constraints.

We can view semidefinite programming as an extension of linear programming. In particular, when the matrices $W$ and $A_j$ $(j = 1, \ldots, m)$ are all diagonal matrices[2], the resulting semidefinite program is equal to a linear program, where the matrix $X$ is replaced by a non-negative vector of variables $x \in \mathbb{R}^n$. In particular, then a semidefinite programming constraint $\operatorname{tr}(A_j X) \leq b_j$ corresponds to a linear programming constraint $a_j^{\mathsf{T}} x \leq b_j$, where $a_j$ represents the diagonal of $A_j$.

Theoretically, semidefinite programs have been proved to be polynomially solvable to any fixed precision using the so-called ellipsoid method (see for instance [9]). In practice, nowadays fast 'interior point' methods are being used for this purpose (see [1] for an overview).

### 3.3 Semidefinite Relaxation for MAX-2-SAT

We applied the semidefinite relaxation of MAX-2-SAT proposed by Goemans and Williamson [8]. The relaxation follows from a quadratic programming formulation of MAX-2-SAT. We first introduce this integer quadratic program.

Let the MAX-2-SAT problem consist of boolean variables $x_1, x_2, \ldots, x_n$ and a set of clauses $C$ on these variables. To each variable $x_i$ $(i = 1, \ldots, n)$, we associate a variable $y_i \in \{-1, 1\}$. Moreover, we introduce a variable $y_0 \in \{-1, 1\}$. We define $x_i$ to be true if and only if $y_i = y_0$, and false otherwise.

Next, we express the truth value of a boolean formula in terms of its variables. Given a formula $c$, we define its *value*, denoted by $v(c)$, to be 1 if the formula is true, and 0 otherwise. Hence,

$$v(x_i) = \frac{1 + y_0 y_i}{2}$$

gives the value of a boolean variable $x_i$ as defined above. Similarly,

$$v(\overline{x}_i) = 1 - v(x_i) = \frac{1 - y_0 y_i}{2}.$$

[2] A diagonal matrix is a matrix whose non-diagonal entries are zero.

Hence, the value of the formula $x_i \vee x_j$ can be expressed as

$$v(x_i \vee x_j) = 1 - v(\overline{x}_i \wedge \overline{x}_j) = 1 - v(\overline{x}_i)v(\overline{x}_j) = 1 - \frac{1 - y_0 y_i}{2}\frac{1 - y_0 y_j}{2}$$
$$= \frac{1 + y_0 y_i}{4} + \frac{1 + y_0 y_j}{4} + \frac{1 - y_i y_j}{4}.$$

The value of other clauses can be expressed similarly. If a variable $x_i$ is negated in a clause, then we replace $y_i$ by $-y_i$ in the above expression.

Now we are ready to state the integer quadratic program for MAX-2-SAT:

$$\begin{array}{ll} \max & \sum_{c \in C} v(c) \\ \text{s.t.} & y_i \in \{-1, 1\}\ \forall i \in \{0, 1, \ldots, n\}. \end{array} \tag{2}$$

It is convenient to rewrite this program as follows. We introduce an $(n+1)\times(n+1)$ matrix $Y$, such that entry $Y_{ij}$ represents $y_i y_j$ (we index the rows and columns of Y from 0 to $n$). Then program (2) can be rewritten as

$$\begin{array}{ll} \max & \text{tr}(WY) \\ \text{s.t.} & Y_{ij} \in \{-1, 1\}\ \forall i, j \in \{0, 1, \ldots, n\}, i \neq j, \end{array} \tag{3}$$

where $W$ is an $(n + 1) \times (n + 1)$ matrix representing the coefficients in the objective function of (2). For example, if the coefficient of $y_i y_j$ is $w_{ij}$, then $W_{ij} = W_{ji} = \frac{1}{2} w_{ij}$.

The final step consist in relaxing the conditions $Y_{ij} \in \{-1, 1\}$ by demanding that $Y$ should be positive semidefinite and $Y_{ii} = 1\ \forall i \in \{0, 1, \ldots, n\}$. Hence, the semidefinite relaxation of MAX-2-SAT is given by the following program

$$\begin{array}{lll} \max & \text{tr}(WY) & \\ \text{s.t.} & Y_{ii} = 1 & \forall i \in \{0, 1, \ldots, n\}, \\ & Y \succeq 0. & \end{array} \tag{4}$$

Program (4) provides an upper bound on the solution to MAX-2-SAT problems. Furthermore, the values $Y_{0i}$, representing $y_0 y_i$, correspond to the original boolean variables $x_i$ $(i = 1, \ldots, n)$. Namely, if $Y_{0i}$ is close to 1, variable $x_i$ is "close to true". Similarly, if $Y_{0i}$ is close to $-1$, variable $x_i$ is "close to false".

*Example 1.* Consider the MAX-2-SAT problem on the variables $x_1$ and $x_2$, with one clause $x_1 \vee \overline{x}_2$. The semidefinite relaxation is

$$\begin{array}{ll} \max & \frac{3}{4} + \frac{1}{4} Y_{01} - \frac{1}{4} Y_{02} + \frac{1}{4} Y_{12} \\ s.t. & Y_{ii} = 1\ (i = 0, 1, 2) \\ & Y \succeq 0. \end{array}$$

An optimal solution is

$$Y = \begin{bmatrix} 1.0 & 0.5 & -0.5 \\ 0.5 & 1.0 & 0.5 \\ -0.5 & 0.5 & 1.0 \end{bmatrix}$$

with objective value 1.125, which is larger than 1, the number of clauses.

The suggestion made by this relaxation is $Y_{01} = 0.5$ and $Y_{01} = -0.5$. This corresponds to "$x_1$ close to true" and "$x_2$ close to false". Indeed, this leads to an optimal solution for the MAX-2-SAT problem.

Example 1 shows that the solution to the semidefinite relaxation may overestimate the actual solution value. In this particular case it is even higher than the number of clauses. Moreover, the fractional solution values are quite far from integrality in this example. In practice however, we will see that the semidefinite relaxation provides surprisingly tight bounds and near-integral values for the variables.

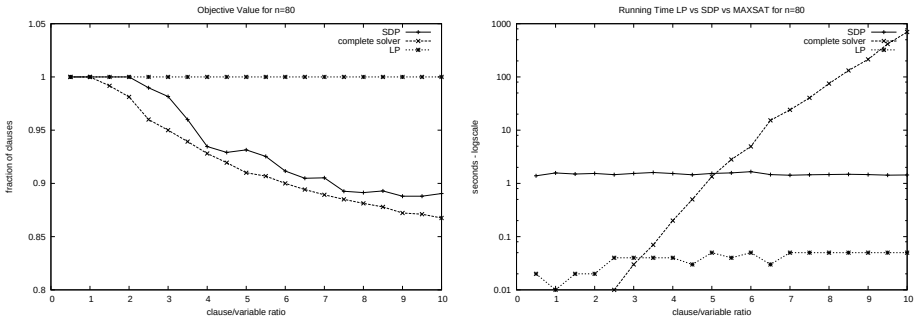
An interesting aspect of program (4) is that the problem instance is entirely encapsulated in the objective function. Hence, the solution process is likely to be independent of the number of clauses, because the model size remains constant for a given number of variables. This is an important property when such relaxations need to be applied in practice.

## 4 Experimental Setup and Results

We have used random MAX-2-SAT instances generated by Selman's MWFF package [15]. For our experiments, we have solved the LP relaxation using ILOG CPLEX libraries. For the semidefinite relaxation we have used the solver CSDP, version 5.0 [3]. To compute the optimum value for the MAX-2-SAT instances, we used MaxSolver, the complete solver from [19].

### 4.1 Quality and Fractionality of Relaxations

We begin by examining the objective value of the LP and SDP relaxations across different constrainedness regions of the problem. We examine MAX-2-SAT instances for 80 variables, varying the $C/V$ ratio from 0.5 to 10. We also solve the instances with a complete solver [19]. (The LP and SDP relaxation can be computed for much higher numbers of variables. However the need for an exact solution limits us to around 80 variables.) The left plot in Figure 2 depicts the median objective value returned by the LP and SDP relaxation versus the median value returned by the MAX-2-SAT solver (bottom curve), as a function of the $C/V$ ratio. Unless otherwise noted, each data point in all the plots corresponds to the median of 100 instances. Since both the LP and the SDP solve a relaxed version of the problem, the objective value is overestimated, and therefore the relaxations provide upper bounds on the optimal solution. We have shown in section 3.1 that we can always find an assignment which makes the objective value of the LP relaxation equal to the number of clauses, hence in the plot the fraction of satisfiable clauses is always 1. While the LP relaxation provides no information about the true solution to the problem, the SDP relaxation follows closely the behavior of the curve corresponding to the maximum fraction of sat-




(a) (b)

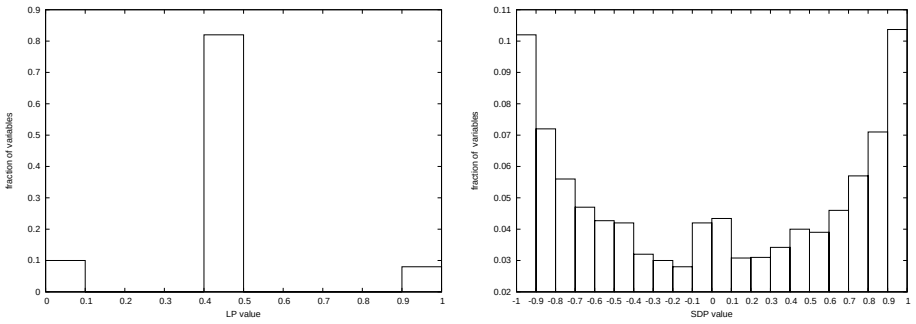
**Fig. 2.** (a) Objective value of the LP and SDP relaxations vs. optimum solution for different $C/V$ values. (b) Runtime of the LP and SDP relaxations vs. the MAX-2-SAT solver in [19].

isfiable instances.[3] Hence, the SDP relaxation is able to adapt to the difficulty of the instances and provides a meaningful upper bound on the maximum number of satisfiable clauses (especially as $C/V$ increases).

The plot on the right depicts the runtime in seconds of the two relaxations versus the MAX-2-SAT solver. The point of this plot is to observe whether the relaxations are affected by the $C/V$ ratio. We first note that the runtime of the two relaxations is hardly affected by the constrainedness of the problem, while the runtime of the MAX-2-SAT solver grows exponentially in the $C/V$ ratio (the plot's $y$ axis is a logarithmic scale). Naturally, the complete solver requires much more time in the over-constrained region, as it has to prove optimality of the found solution. The LP relaxation is computationally the least expensive across the board, except for the under-constrained region, where the problem is easy and the complete solver is able to quickly examine the search space.

In order to understand what enables the SDP relaxation to be more informed than the LP relaxation, we continue by studying the fractionality of the values returned by the two relaxations. Figure 3 plots the distribution of these values, in intervals of length 0.1 from 0 to 1 for the LP relaxation and from $-1$ to 1 for the SDP relaxation averaged over all instances ($C/V$ ratio varying from 0.5 to 10). The ends of the two intervals correspond to the Boolean values false and true, respectively. The closeness of a value to one of the ends can be interpreted as the "confidence" of the relaxation that the corresponding Boolean variable should be set to true/false. We observe that the LP relaxation only sets variables to three values: 0, 1 and 0.5. 0 and 1 correspond to false and true, respectively, however 0.5 gives us no information as to what should be assigned to the corresponding Boolean value. In contrast, the majority of values returned by the SDP relaxation

[3] Note that SDP can provide an objective value greater than the number of clauses; in those cases, for obvious reasons, we consider the number of clauses in the formula as the upper bound on the optimal value.




(a) (b)

**Fig. 3.** Distribution of the values returned by the (a) LP and (b) SDP relaxation (averaged over instances with C/V ranging from 0.5 to 10).

are concentrated towards the ends of the interval $[-1, 1]$, thus suggesting more informed guidance about the way the variables should to be set.

To further describe the fractionality of the relaxations we examine them across different constrainedness regions. Figure 4 a) plots the fraction of variables that are equal to 0.5 after solving the LP relaxation. In the under-constrained region (i.e., low clause/variable ratio) the fraction is 0, then as we pass the $C/V = 1$ point, the percentage goes up, and it approaches 1 (i.e., all variables) as the problem becomes over-constrained.

Similarly, figure 4a) plots the fraction of SDP variables whose values lie in the neighborhood of 0. We represent the fraction of variables equal to 0 and also those variables whose absolute value lies in the interval $(0, 0.1]$. In the under-constrained region the fraction of variables set to 0 is very high (close to 0.8). As we add more clauses this fraction sharply decreases and stabilizes at 0. These variables correspond to boolean variables that do not appear in the formula, hence the high fraction of such variables in the under-constrained region.

Figure 4b) plots the fraction of high confidence variables. For LP (variables having the value 0 or 1) we see a significant change around $C/V = 1$ and then the fraction decreases all the way to 0. (in fact this curve is the complement of the curve representing the variables set to 0.5 by LP). For SDP we plot the fraction of variables that are greater than 0.7 in absolute value. This fraction is low in the under-constrained region (as most of the variables are set to 0 as explained above) and it goes up to roughly half the variables as the problem becomes more constrained.

The plots demonstrate that the two relaxations examined behave very differently across the constrainedness regions. As the problem gets harder (i.e., more constrained) the LP relaxation "defaults" to an uninformative assignment (all variables are assigned 0.5). In contrast, the SDP relaxation provides a good upper bound on the optimal solution (see Figure 2 and 1), with runtimes below 2 seconds per run. Furthermore, the SDP values assigned to the Boolean variables

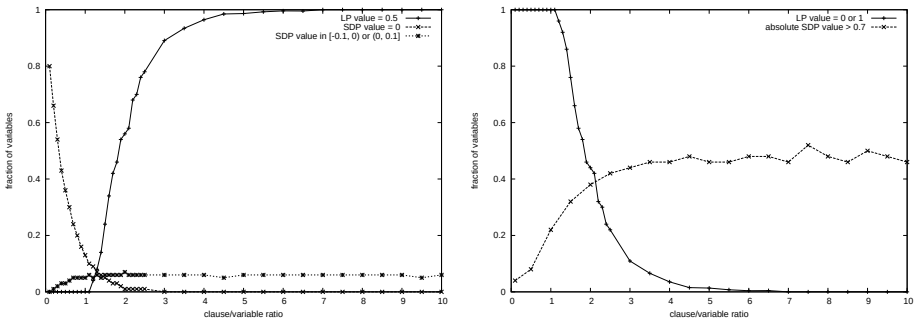



**Fig. 4.** a) Fraction of variables having the value a) 0.5 computed by LP and 0 or smaller than 0.1 in absolute value computed by SDP and b) 0 or 1 computed by the LP relaxation and above 0.7 in absolute value by the SDP relaxation.

are less fractional than those delivered by LP (Figure 3) and therefore can be used more effectively as heuristics, as we will see in the next section.

### 4.2 SDP and LP as a Backtrack Search Heuristic

Related to the fractionality of the SDP relaxation is the question of whether the SDP relaxation provides a good global perspective of the search space and therefore could be used as a heuristic by MAX-SAT solvers. To test the heuristic power of SDP we used the following method:

**Algorithm 1 SDP-based Heuristic**
Input: a MAX-2-SAT instance and $x$ (the number of variables to be set using the SDP relaxation).

Step 1: solve the SDP relaxation.

Step 2: set $x$ variables to the value suggested by the SDP relaxation[4].

Step 3: given the (partial) assignment of variables from step 2, compute the MAX-2-SAT for the original instance s.t. the maximizing assignment extends this partial assignment.

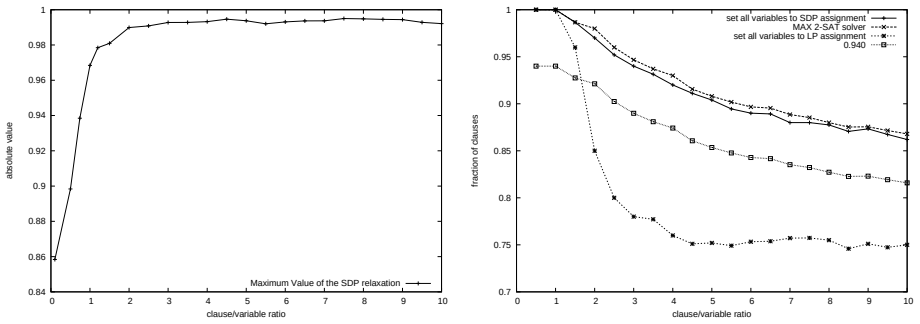
Output: the maximum number of sat clauses.

To put into perspective the heuristic power of the SDP relaxation, we compare it to that of the LP relaxation. The following method was used:

**Algorithm 2 LP-based Heuristic**
Input: a MAX-2-SAT instance and $x$ (the number of variables to be set using the LP relaxation).

Step 1: solve the LP relaxation.

[4] We consider variables in decreasing order of their absolute value.




(a) (b)

**Fig. 5.** a) The maximum value returned by the SDP relaxation and b) the heuristic power of SDP vs LP.

Step 2: set $x$ variables to the value suggested by the LP relaxation[5].

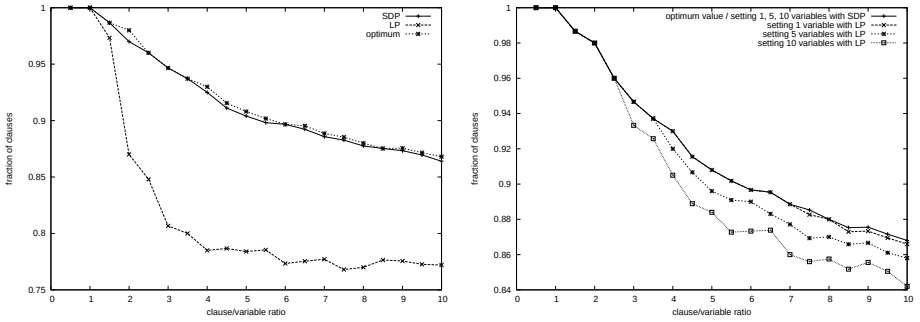
Step 3: given the (partial) assignment of variables from step 2, compute the MAX-2-SAT for the original instance s.t. the maximizing assignment extends this partial assignment.

Output: the maximum number of sat clauses[6].

Figure 5 a) plots the maximum absolute variable value returned by the SDP relaxation as we increase the constrainedness of the problem. Given the fact that this value is very close to 1 and the high density of variables having high "confidence" level, the following experiments are justified. In the first experiment conducted, we set all the variables ($x = n$) to the value suggested by the relaxations and then we report the number of satisfied clauses. Figure 5 b) shows the results for the two relaxations versus the MAX-SAT value. Each data point corresponds to the median of 100 instances. The SDP performance is quite impressive, as it stays very close to the optimum across the board. When using the LP relaxation the performance degrades significantly as the problem becomes more constrained. The SDP's heuristic power does not seem to be affected by the constrainedness of the problem. We have also included in figure 5 b) the curve corresponding to the best theoretical guarantee for an SDP based approximation algorithm (i.e., 0.940-approximation [13]). The results in figure 5, show that on random instances an algorithm using our SDP heuristic comes on average within 0.99 of the optimum. Thus, for most cases, when an estimate

[5] First consider the variables that were set to 0 or 1 by the LP relaxation and set the corresponding Boolean variables to false or true, respectively. If we exhaust all such variables, we consider variables that are 0.5 and randomly set the corresponding variable to true or false. Note that LP only assigns 0, 1, and 0.5 values to variables. Since the procedure is randomized we perform it many times for one instance

[6] Because we perform the variable setting several times for every input instance, we return the median.




(a) (b)

**Fig. 6.** Change in the number of satisfied clauses as we set a) 84% of the variables and b) 1, 5 and 10 variables using the LP and SDP.

of the MAX-2-SAT solution is needed, simply setting the values suggested by the SDP relaxation should be a good heuristic.

We also varied the number of variables set ($x$) between 0 and $n$ and studied the effect on the maximum number of satisfiable clauses. We discovered that the median maximum number of satisfiable clauses when using SDP remains the same as the optimum, when we set up to 84% of the variables (i.e., 42 variables for n=50). It is at this point that we observe a slight change in performance – see figure 6 a).

Once again, we compared SDP with LP and observed that when using LP the performance starts degrading even after setting just 1 variable and it continues to drop as we increase $x$ (figure 6 b)). In contrast, SDP makes no mistakes when setting 1 and 2 variables and we have found just one instance (in over 2000 random instances) for $x = 3$, where the maximum number of satisfied clauses decreased by 1. For $x = 10$, the number of instances where the SDP suggestion is sub-optimal is four. These results show that the SDP relaxation is very informed and by following the SDP suggestion we remain very close to optimal performance.

In section 4.1 we showed that the objective function of the SDP relaxation provides a good upper bound for MAX-2-SAT. We test the potential of the SDP relaxation to provide a lower bound, by using the SDP relaxation to set all variables and we compare it to Walksat [16]. We ran Walksat with a cutoff of $10^8$ flips. The average Walksat runtime was approximately five minutes, while the SDP relaxation was computed in approximately 1 second. The results are presented in figure 7. The plot on the left side depicts the upper bound, lower bound, optimum value and Walksat value for the $C/V$ ratio ranging from 0.5 to 10. We note that the curves are very close to each other, but Walksat still provides a better lower bound. In the plot on the right side, as we increase the $C/V$ ratio up to 100, we see that SDP outperforms Walksat, namely it provides

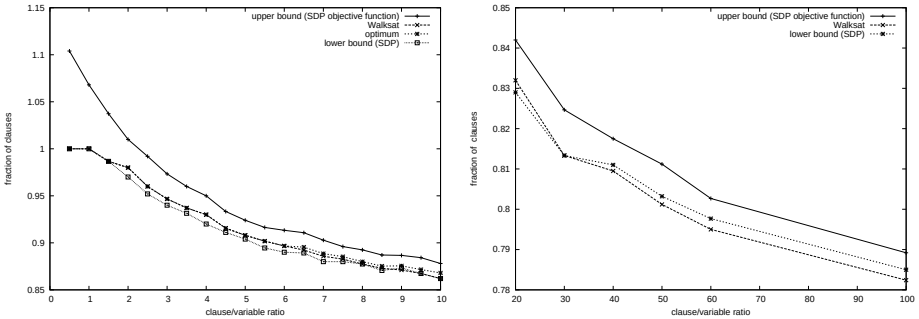



**Fig. 7.** SDP vs Walksat as a lower bound.

a better bound, using considerably less time (recall from section 4.1 that the runtime for the SDP relaxation does not depend on the $C/V$ ratio).

Given these very promising results for the guiding power of the SDP relaxation, we believe that the performance of MAX-SAT solvers could be further improved by incorporating the information provided by the SDP relaxation, for instances in the over-constrained region, where the time required to compute the SDP relaxation is considerably smaller than the time needed by the current MAX-SAT solvers (see figure 2 b)).

## 5 Conclusions and Future Work

In this paper we have shown how SDP relaxations are surprisingly powerful, providing much tighter bounds than LP relaxations, across different constrainedness regions. SDP relaxations can be computed very efficiently, thus quickly providing tight lower and upper bounds on the optimal solution. We have also shown that heuristic guidance based on the SDP relaxation for iterative variable setting is significantly more accurate than the guidance based on the LP relaxation. SDP allows us to set up to around 84% of the variables without degrading the optimal solution, while setting a single variable based on the LP relaxation generally degrades the global optimal solution in the overconstrained area. We also compared SDP against Walksat: Interestingly, as the instances become more and more constrained, the lower bound provided by the SDP relaxation outperforms Walksat, both in terms of quality and time (2 seconds for SDP versus 5 minutes for Walksat). Furthermore, Walksat has the limitation of not providing upper bounds. In our experiments, the SDP upper bound is always less than 3% above the optimal solution. Our results therefore show that SDP relaxations may further boost exact MAX-SAT solvers.

## References


1. F. Alizadeh. Interior point methods in semidefinite programming with applications to combinatorial optimization. *SIAM Journal on Optimization*, 5(1):13–51, 1995.
2. M.F. Anjos. Semidefinite optimization approaches for satisfiability and maximum-satisfiability problems. *Journal on Satisfiability, Boolean Modeling and Computation*, 1:1–47, 2005.
3. B. Borchers. A C Library for Semidefinite Programming. *Optimization Methods and Software*, 11(1):613–623, 1999.
4. V. Chvátal and B. Reed. Mike gets some (the odds are on his side). In *33th Annual Symposium of Foundations of Computer Science*, pages 620–627, 1992.
5. E. de Klerk and H. van Maaren. On semidefinite programming relaxation of 2+p-sat. *Annals of Math. and Artificial Intelligence*, 37, 2003.
6. M.R. Garey and D.S. Johnson. *Computers and Intractability.* Freeman, 1979.
7. M. Goemans and F. Rendl. Combinatorial Optimization. In H. Wolkowicz, R. Saigal, and L. Vandenberghe, editors, *Handbook of Semidefinite Programming*, pages 343–360. Kluwer, 2000.
8. M.X. Goemans and D.P. Williamson. Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming. *Journal of the ACM*, 42(6):1115–1145, 1995.
9. M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization.* John Wiley & Sons, 1988.
10. D.S. Hochbaum, editor. *Approximation algorithms for NP-Hard problems.* PWS Publishing Company, 1997.
11. J.N. Hooker and C. Fedjiki. Branch-and-cut solution of inference problems in propositional logic. *Annals of Math. and Artificial Intelligence*, 1, 1990.
12. S. Joy, J. Mitchell, and B. Borchers. A branch and cut algorithm for MAX-SAT and weighted MAX-SAT. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 35:519–536, 1997.
13. M. Lewin, D. Livnat, and U. Zwick. Improved rounding techniques for the MAX 2-SAT and MAX DI-CUT problems. In *IPCO*, pages 67–82, 2002.
14. R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.
15. B. Selman. Mwff - a program for generating random MAX k-SAT instances, 1993.
16. B. Selman, H. Kautz, and B. Cohen. Local search strategies for satisfiability testing. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 26:521–532, 1996.
17. U. Feige and M. Goemans. Approximating the value of two prover proof systems, with applications to max2sat and max dicut. In *Proceedings of the 3rd Israel Symposium on Theory of Computing and Systems*, 1995.
18. J. Warners. *Nonlinear approaches to satisfiability problems.* PhD thesis, Technische Universiteit Eindhoven, 1999.
19. Z. Xing and W. Zhang. Maxsolver: An efficient exact algorithm for (weighted) maximum satisfiability. *Artificial Intelligence*, 164(1–2):47–80, 2005.