Low-Rank Semidefinite Programming for the MAX2SAT Problem

Po-Wei Wang
Machine Learning Department
Carnegie Mellon University
Pittsburgh, PA 15213
poweiw@cs.cmu.edu

J. Zico Kolter
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213, and
Bosch Center for Artificial Intelligence
Pittsburgh, PA 15222
zkolter@cs.cmu.edu

Abstract

This paper proposes a new algorithm for solving MAX2SAT problems based on combining search methods with semidefinite programming approaches. Semidefinite programming techniques are well-known as a theoretical tool for approximating maximum satisfiability problems, but their application has traditionally been very limited by their speed and randomized nature. Our approach overcomes this difficult by using a recent approach to low-rank semidefinite programming, specialized to work in an incremental fashion suitable for use in an exact search algorithm. The method can be used both within complete or incomplete solver, and we demonstrate on a variety of problems from recent competitions. Our experiments show that the approach is faster (sometimes by orders of magnitude) than existing state-of-the-art complete and incomplete solvers, representing a substantial advance in search methods specialized for MAX2SAT problems.

1 Introduction

The maximum satisfiability (MAXSAT) task — the optimization-based version of satisfiability, where the goal is to find a maximal set of clauses to be satisfied — is a foundational problem in combinatorial optimization. These problems can be written as the optimization task

$$\underset{v_1,\ldots,v_n\in\{0,1\}}{\text{maximize}} \sum_{j=1}^{m} \bigvee_{i\in S_j^+} v_i \bigvee_{i\in S_j^-} \neg v_i, \tag{1}$$

where $v_1, \ldots, v_n$ denote the binary optimization variables, and S_j^+/S_j^- denotes the set of variables and negated variables respectively in the jth clause. MAXSAT problems have been used to solve probabilistic inference (Park, 2002), planning (Zhang and Bacchus, 2012), and clustering (Berg and Järvisalo, 2017). And the recent MAXSAT contests (Argelich et al., 2016) have aimed at evaluating a wide number of different solution methods on a wide variety of both real and synthetic problems.

In this paper, we present an approach based upon low-rank semidefinite programming (SDP) and a simple branch-and-bound strategy for solving the simplest case of the MAXSAT problem: the MAX2SAT problem, where each clause contains only two variables. While this is naturally a very limited special case of the MAXSAT problem, unlike the SAT setting (where 2-SAT can be solved in polynomial time), the MAX2SAT problem is NP-COMPLETE and forms a non-trivial starting point for more general MAXSAT problems. Although SDP relaxations of the MAX2SAT problem have been known for some time (Goemans and Williamson, 1994, 1995; Gomes, van Hoeve, and Leahu, 2006), they have not generally been viewed as practical strategies for solving MAXSAT problems (of any type), owing to the high computational cost of solving SDPs via standard methods (e.g., interior point methods, whose runtime grows cubically in the number of variables). Indeed, to the best of our knowledge, no entry in the history of the MAXSAT competition (Argelich et al., 2016) has ever used an approach based upon semidefinite programming. Furthermore, our approach also applies to more general MAXSAT problems, though the practical efficiency is much less for settings with a large number of variables per clause; but later in the paper, we will also highlight MAX3SAT results, for example, where the method is also still competitive.

In this work, we show that a recent method for low-rank semidefinite programming (the Mixing method (Wang, Chang, and Kolter, 2017)) combined with branch-and-bound strategy, warm-starts, and simple pruning based upon a dual bound, can achieve state-of-the-art performance for MAX2SAT problems. Specifically, for the MAXSAT 2016 competition problems[1], our single proposed approach dominates the best solvers (selecting the best solver for each individual problem instance for all the 2016 problems) in *both* the incomplete and complete tracks of the competition. While naturally these results need to be taken with a grain of salt, as the MAXSAT competition solvers of course did *not* specialize for the MAX2SAT case, it is notable our single solver, using ultimately a very simple search procedure, can outperform highly specialized and tuned solvers so heavily in this domain across both competition tracks. Thus, while a great deal of work remains to turn the approach into a general MAXSAT solver, the work here strongly suggests that semidefinite programming techniques can play a sizable role in the solution to MAXSAT problems going forward.

[1] The 2016 MAXSAT competition was the last year of the competition to features MAX2SAT instances, as the "random" problem track was removed in subsequent years.

2 Preliminaries and related work

In this section we highlight a number of current approaches for solving MAXSAT problems, and review relevant recent work on semidefinite approximations and fast semidefinite programming methods. We will specifically highlight the recent Mixing method (Wang, Chang, and Kolter, 2017), which forms the basis of our inner solution method.

2.1 Discrete MAXSAT solvers

The vast majority of modern methods for the MAXSAT problem are based upon discrete search method. As our proposed approach in this work (which based upon continuous optimization) differs quite substantially from these approaches, we refer the reader to a recent survey (Morgado et al., 2013) for much more detailed descriptions about the current state of the art. However, broadly speaking, there have been two main classes for these discrete solvers: 1) those based upon bounding the solution via SAT method (Marques-Sila and Planes, 2011; Koshimura et al., 2012; Ansótegui, Bonet, and Levy, 2013; Fu and Malik, 2006; Le Berre and Parrain, 2010; Eén and Sorensson, 2006) which in turn exploit the heuristic developed by the SAT community such as those in the MiniSAT solver; these solvers typically are *complete* in that they will both produce a satisfying assignment with some number of clauses satisfied *and* a verification that this is the optimal solution to the problem. And 2) those based upon local search (Luo et al., 2015, 2017), which maintain and locally adjust a solution to satisfy an increasing number of clauses; these solvers typically are *incomplete* in that they may quickly find an assignment, but often cannot prove whether or not it is optimal. There are also aggregation-based solvers such as Open-WBO (Martins, Manquinho, and Lynce, 2014), which integrate several solvers for different instances.

2.2 Approximation algorithms for MAXSAT

In addition to exact solvers based upon combinatorial search, several approaches have also considered *approximation* algorithms for the MAXSAT problem, both from theoretical and practical standpoints. Generally speaking, these focus on obtaining a high approximation ratio α, such that

$$\text{Approximated objective} \geq \alpha \cdot \text{Optimal objective}. \quad (2)$$

Specifically, Goemans and Williamson (1994) proposed an LP approximation algorithm for MAXSAT with $\alpha = 0.75$. They latter proposed also an SDP approximation (Goemans and Williamson, 1995) for MAX2SAT with $\alpha = 0.878$, which will form the basis for the SDP problem we solve here, and which we describe in more detail below. Feige and Goemans (1995) used a different and more refined SDP to raise the approximation ratio to $\alpha = 0.931$, and Karloff and Zwick (1997) extend the result to deal with MAX3SAT with $\alpha = 0.875$. Interested readers can refer to Biere, Heule, and van Maaren (2009) for a comprehensive introduction.

Although several authors have noted the effectiveness of the SDP relaxation for the MAX2SAT problem (Lotgering, 2012; Gomes, van Hoeve, and Leahu, 2006) in terms of the approximation quality, no previous works we are aware of have succeeded at turning this into a practical algorithm. Several efforts were made: Anjos (2004) investigated the effectiveness of different SDPs for the MAXSAT problems, but didn't consider their integration into an exact solver. van Maaren, van Norden, and Heule (2008) investigated the effectiveness of sum-of-squares based SDP solvers and found them not practical. And Rendl, Rinaldi, and Wiegele (2010); Krislock, Malick, and Roupin (2017) demonstrated an SDP-based branch-and-bound framework for the related MAX-CUT and binary quadratic problem (using interior point and bundle solvers) and created the Biq Church tool; however, this method was only feasible for small dense instances.

2.3 Low-rank semidefinite programming

As mentioned, the approach we ultimately use is one based upon semidefinite programming. A general SDP can be written as the optimization problem

$$\begin{aligned} \underset{X \in \mathbb{R}^{n \times n}}{\text{minimize}} \;& \langle C, X \rangle \\ \text{subject to} \;& \langle A_i, X \rangle = b_i, \quad i = 1, \dots m, \\ & X \succeq 0 \end{aligned} \quad (3)$$

where $A_i \in \mathbb{R}^{n \times n}, b_i \in \mathbb{R}, i = 1, \dots, n$ and $C \in \mathbb{R}^{n \times n}$ are problem data, and X is the optimization variable. Most general purpose semidefinite programs typically use interior point methods (Nocedal and Wright, 2006) to solve problems to very high precision, but the methods scale very poorly: interior point solvers operate in the space of $X \in \mathbb{R}^{n \times n}$, and the solution time is cubic in the number of variables, i.e., $O(n^6)$.

Owing to this fact, a number of approaches based upon *low-rank factorization* have become popular for solving large-scale semi-definite programs. Specifically, a now-classic result (Pataki, 1998) guarantees that for a problem with m constraints, there exists a rank-$\sqrt{2m}$ optimal solution. Motivated by this fact, in a seminal work, Burer and Monteiro (2003) proposed to solve the semidefinite program directly in terms of the low-rank factorization $X = V^T V$ for $V \in \mathbb{R}^{k \times n}$, where one can omit the semidefinite constraint (because it is implied by the factorization), and where k could be chosen for instance to satisfy $k \geq \sqrt{2m}$; this leads to many fewer variables in the problem of interest. They specifically used an augmented Lagrangian method to solve the constrained optimization problem, and noticed that despite the fact that the resulting problem is of course non-convex, in practice this would virtually always find the *optimal* solution to the original SDP. Although this remained an empirical rather than theoretical property for many years, very recently Boumal, Voroninski, and Bandeira (2016) has shown that there are *no spurious local optima* in the augmented Lagrangian form for these problems, and thus such methods are guaranteed to find the optimal solution.

2.4 The Mixing method

Despite the appeal of the low-rank solution approach, the method is still relatively slow for even medium-sized SDPs; because augmented Lagrangian methods require a carefully-tuned gradient descent procedure to optimize the augmented

Lagrangian and adjust the Lagrangian penalty parameter. For a specific class of semidefinite program, however, Wang, Chang, and Kolter (2017) developed a specific form of block coordinate descent, called the Mixing method, which substantially improves upon the generic low-rank approach both practically and theoretically. Specifically, the Mixing method aims to solve the diagonally-constrained SDP

$$\underset{X \succeq 0 \in \mathbb{R}^{n \times n}}{\text{minimize}} \ \langle C, X \rangle, \ \text{ s.t. } \ X_{ii} = 1, \ i = 1, \ldots, n \tag{4}$$

i.e., an SDP where there are only n equality constraints, constraining each diagonal entry of X to be 1. This is referred to as the MAXCUT SDP, since it corresponds to the SDP relaxation of the maximum cut problem, where C is the adjacency matrix of a graph. In factorized form $X = V^T V$, this corresponds to the non-convex problem

$$\underset{V \in \mathbb{R}^{k \times n}}{\text{minimize}} \ \langle C, V^T V \rangle, \ \text{ s.t. } \ \|v_i\|_2 = 1, \ i = 1, \ldots, n, \tag{5}$$

where $v_i \in \mathbb{R}^k$ denotes the ith column of V. The approach is then quite simple: if we hold all but one v_i fixed, there is a simple closed-form solution for the remaining v_i given by

$$v_i := \text{normalize}\left(-\sum_{j \neq i} c_{ij} v_j \right). \tag{6}$$

The Mixing method simply repeats this iteration until convergence; unlike the augmented Lagrangian approach, the method does not require any step size or tuned penalty parameter. In practice, it is also an order of magnitude or more faster than any other state-of-the-art solver for such problems. And as shown by Wang, Chang, and Kolter (2017), as long as k is chosen such that $k > \sqrt{2n}$, the method will still converge to the *global* optimum of the original SDP. This fast method for low-rank semidefinite programming will be the basis for our fast SDP-based MAX2SAT solver.

3 MAXSAT and the semidefinite relaxation

In this section we first derive a general SDP relaxation for the MAXSAT problem (although in the subsequent section we will focus on the MAX2SAT problem, we note that the formulation here is general and applies to *any* MAXSAT problem, though it is substantially looser for settings with more than two variables per clause). We then derive a version of the Mixing method specifically for this SDP; because the SDP is virtually identical to that for the MAXCUT problem, the algorithm is mathematically the same as the original Mixing method, with just a few additional elements introduced to make it computationally efficient for the specific problem structure of the MAXSAT.

3.1 The MAXSAT SDP

For the purposes of this section, we will define the general MAXSAT problem slightly different than in the introduction, as the optimization problem

$$\underset{v \in \{-1,1\}^n}{\text{maximize}} \ \sum_{j=1}^{m} \bigvee_{i=1}^{n} \mathbf{1}\{s_{ji} v_i > 0\}, \tag{7}$$

where $v_i \in \{-1, 1\}$ are the binary optimization variables (i.e., making these $+1/-1$ instead of 0/1 as we did in the introduction); and where $s_{ji} = \pm 1$ for variable in the clause (and taking each value depending on whether the variable is negated or not in that clause), and $s_{ji} = 0$ otherwise. For conversion to SDP form it is slightly more convenient to formulate this in minimization, or *unsatisfiability*, form

$$\underset{V \in \{-1,1\}^n}{\text{minimize}} \ \sum_{j=1}^{m} \bigwedge_{i=1}^{n} \mathbf{1}\{s_{ji} v_i < 0\}, \tag{8}$$

where $\bigwedge$ is the logical and symbol. At this point, we now seek a continuous *lower-bound* on this objective, which we refer to simply as the *loss*, and which takes the value $+1$ if an only if all variables within the clause are false (-1) and is zero or less otherwise, i.e.,

$$\text{loss}(v; s_j) \leq \texttt{unsat}(v, s_j), \ \forall v_i \in \{-1, 1\}^n. \tag{9}$$

In order to translate the problem specifically to an SDP, we specifically search for a *quadratic* loss; taking the *greatest* such lower bound that satisfies the conditions above, we arrive at the loss function

$$\text{loss}(v; s_j) = \frac{\left(\sum_{i=0}^{n} v_i s_{ji}\right)^2 - (n_j - 1)^2}{4 n_j}, \tag{10}$$

where n_j is the number of variables in clause j (i.e, 2 in all cases for MAX2SAT), and where we define $v_0 = 1$ and $s_{j0} = -1$ (we introduce these variables to make the subsequent SDP relaxation easier to write, with purely quadratic terms in the objective). This loss is best illustrated graphically, as shown in Figure 1 for the case of $n_j = 2$. For the case of any n_j, it's easy to verify that this quantity is equal to $+1$ if no clauses are satisfied, 0 if exactly one or exactly n_j clauses are satisfied, and is strictly less than 0 otherwise; these are exactly the conditions that the loss was required to meet.

We can now relax this binary problem into its vector form by relaxing the variable $v_i \in \{-1, +1\}$ to $v_i \in \mathbb{R}^k$ with $\|v_i\|_2 = 1$. In this vector form, the loss above becomes simply

$$\text{loss}(v; s_j) = \frac{\|V s_j\|^2 - (n_j - 1)^2}{4 n_j} \tag{11}$$

which is equivalent to the SDP

$$\underset{V \in \mathbb{R}^{k \times n}}{\text{minimize}} \ \langle S^T D^{-1} S, V^T V \rangle, \ \text{ s.t. } \ \|v_i\|_2 = 1, \ i = 1, \ldots, n, \tag{12}$$

where the matrices are all as defined above and with $D_{jj} = 4 n_j$. This is precisely the form of the SDP solved by the Mixing method, with the only difference being the special form of the $C = S^T D^{-1} S$ matrix, which requires some slight optimizations to produce an efficient method, which we discuss next. We also highlight that for the case of MAX2SAT, this is precisely the same form as the SDP relaxation of Goemans and Williamson (1995), but generalizes to longer clauses as well.

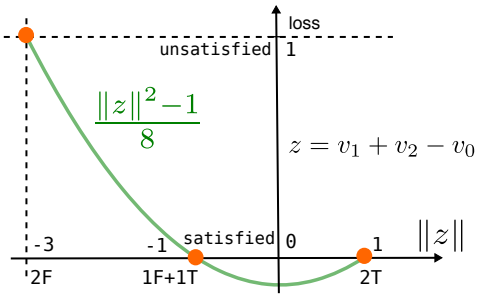

Figure 1: Convex lower bound of the unsatisfiability loss.

Initialize all v_i randomly on a unit sphere;
Let $z_j = \sum_{i=0}^{n} s_{ji} v_i$ for $j = 1, \ldots, m$;
while *not yet converged* **do**
  for $i = 1, \ldots, n$ **do**
    foreach $s_{ji} \neq 0$ **do** $z_j := z_j - s_{ji} v_i$;
    $v_i :=$ normalize $\left(- \sum_{j=1}^{m} \frac{s_{ji}}{4n_j} z_j \right)$;
    foreach $s_{ji} \neq 0$ **do** $z_j := z_j + s_{ji} v_i$;
  end
end

Algorithm 1: The Mixing method for MAXSAT relaxation

3.2 The Mixing method for the MAXSAT SDP

Recall from the previous section that the Mixing method for solving the SDP above reduces to the iterations

$$v_i := \text{normalize} \left(- \sum_{k \neq i} (S^T D^{-1} S)_{ik} v_k \right). \tag{13}$$

To compute this term efficiently without requiring precomputation of the entire $S^T D^{-1} S$ matrix ahead of time (which is actually a *dense* matrix owing to the fact that the "0th" variable v_0 participates in ever clause), note that we can cache the vectors $z_j \in \mathbb{R}^k$, $j = 1, \ldots, m$, with

$$z_j = \sum_{i=0}^{n} s_{ji} v_i. \tag{14}$$

Then the update equation can be written as

$$v_i := \text{normalize} \left(- \sum_{j=1}^{m} \frac{s_{ji}}{4n_j} z_j \right). \tag{15}$$

if we first remove all the terms in each z_j that depends on v_i. The final form of the Mixing method for this MAXSAT SDP is given in Algorithm 1. Notice that updating the variable v_i only requires time $O(k \cdot \texttt{nnz})$ times, where $\texttt{nnz}$ is the number of clauses that contain the ith variable. Furthermore, although this would be a large number for the v_0 term, we can avoid updating v_0 entirely, as it plays the role of an arbitrary "truth direction" that need not be updated from its original setting.

Initialize a priority queue $Q = \{\text{initial problem}\}$;
while Q *is not empty* **do**
  New SDP root P = Q.pop();
  Solve $f^* := \texttt{SDP}(P)$ with the Mixing method;
  if $\lceil f^* - \epsilon \rceil \geq \texttt{best}$ **then** continue;
  Update `best` and resolution orders by randomized rounding on $V := \arg \texttt{SDP}(P)$;
  foreach *subproblems of* P **do**
    Initialize `primal` and `dual` objective values;
    if $\texttt{primal} \leq \texttt{best}$ **then** Expand;
    else if $\lceil \texttt{dual} \rceil \geq \texttt{best}$ **then** Prune;
    else Push the subproblem into the Q;
  end
end

Algorithm 2: The MIXSAT algorithm

4 The MIXSAT algorithm

In this section, we combine the previous SDP approach within a simple branch and bound framework to derive a exact and complete solver for MAX2SAT instances.[2] Although the SDP relaxation and fast solution method plays the largest role in our solver, a number of other elements are crucial as well, such as warm starting based upon previous solutions in the tree, rounding to find feasible solutions at intermediate stages, and bounding the SDP optimal value by a particular set of primal and dual feasible points. We also introduce data structures such as a watched stack (similar to those used by SAT solvers) to efficiently make inferences about those clauses that have been fulfilled or falsified. Despite some additional complexity, we highlight that the ultimate algorithm here is still relatively simple compared to many state-of-the-art MAXSAT sovlers: just a branch and bound strategy with well-established optimizations adapted to this particular SDP-based heuristic. The fact that the approach can nonetheless outperform existing solvers *both* for complete and incomplete versions highlights the potential power of this SDP relaxation, not just from the theoretical side (which has been well-studied by past work), but from the practical side as well.

The basic method, which we refer to as MIXSAT (for Mixing applied to MAXSAT), is shown in Algorithm 2. The basic approach proceeds like a generic branch and bound search, with a priority queue storing problem instances (by using a priority queue ordered by heuristic values versus a stack, we can recover either a best first or depth first search, which we use in the incomplete and complete versions of the solver respectively). We pop these problems off the queue, solve the SDP via the Mixing method, and then round the result to potentially obtain a new candidate solution (the SDP solution also leads to a subsequent ordering of variables to split on). For each split, we set the relevant set of variables be either true or false, and we initialize the primal and dual ob-

[2] Since the aforementioned bound works for MAXSAT problems of any size, the methods here could be in theory applied to any MAXSAT problem. We will show that the methods generalize to MAX3SAT instances later in the paper, though theoretically the bound would be looser for longer clauses.

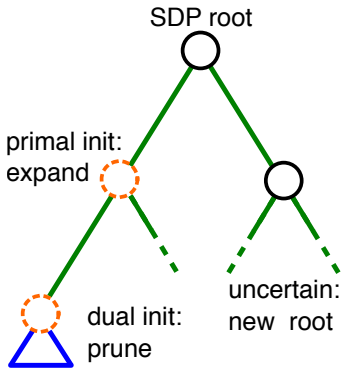

Figure 2: Illustration of primal and dual initialization.

jectives of each new subproblem, as described below, then push these on to the queue and continue until the queue is empty (for the complete solver). There are indeed several details of the process, which we describe next throughout this section.

4.1 Pruning and rounding via SDP

Pruning. Because our SDP approximation is a lower bound of the exact minimum unsatisfiability problem, it can serve as a lower-bound in the branch-and-bound framework. Specifically, as is standard in branch and bound, if the objective value of a solved SDP value is higher than the current best known candidate solution, we can prune the entire subtree below that problem. Otherwise, we expand the subtree and push the subproblems on to the queue.

Rounding. The SDP solutions also serve as a good method for obtaining candidate solutions via rounding, which can usually very quickly provide a good best known value, even in early stages of the algorithm. The randomized rounding method works by repetitively drawing random vectors r uniformly from the sphere, and assigning binary $v_i = \text{sign}(v_0^T r \cdot v_i^T r)$ as a candidate binary solution. That is, if a variable v_i is at the same side with the truth direction v_0, we assign it to be true $(+1)$ and otherwise false (-1). Finally, if the rounded solution ever obtains the same integer objective value as the best lower bound, we know we have found an optimal solution.

4.2 Bounding SDP values by initializations

Even with the pruning and rounding strategies above, it typically requires a large number of node expansions to solve even most MAX2SAT problem. As one data point, for instance for the MAX2SAT problems with 120 variables, it typically requires searching through about 10k nodes in order to provide a verified solution to the problem. However, we reduce the number of expanded nodes (and hence the number of SDP solves) by reusing the solution of an SDP to quickly find both primal and dual feasible initialization to its subproblems. Indeed, sometimes we don't even need to solve the SDP in order to expand or prune a subproblem.

Recall the vector program equivalent to our diagonal constrained SDP relaxation

$$\underset{V\in\mathbb{R}^{k\times n}}{\text{minimize}}\ f(V) := \langle C, V^T V\rangle, \quad \text{s.t. } \|v_i\|^2 = 1. \tag{16}$$

By the SDP duality, we have the following dual problem

$$\underset{\lambda\in\mathbb{R}^n}{\text{maximize}}\ D(\lambda) := -1^T\lambda, \quad \text{s.t. } C + D_\lambda \succeq 0. \tag{17}$$

Any *feasible* primal and dual solution will provide an upper and lower bound respectively of the optimal SDP objective value f^*. Furthermore, we know that the best known discrete solution, `BEST`, is always larger than the optimal discrete solution `UNSAT`. That is,

$$f(V) \geq f^* \geq D(\lambda) \quad \text{and} \quad \texttt{BEST} \geq \texttt{UNSAT} \geq \lceil f^* \rceil.$$

Thus, for any feasible primal solution V and any feasible dual solution λ, we can prune or expand the subproblem when the following condition happens, as illustrated by Figure 2.

- $f(V) \leq \texttt{BEST} \Rightarrow$ Expand (not prunable by SDP)
- $\lceil D(\lambda) \rceil \geq \texttt{BEST} \Rightarrow$ Prune (not contain optimal solution)

Upper bound by primal initialization. The primal initialization is very simple: we simply copy all unassigned variables from the root solution into its subproblems, which will still provide a feasible solution. The effective length of the remaining clauses do change when this occurs, which can be updated efficiently by maintaining each z_j properly after such updates.

Lower bound by dual initialization. The dual initialization is more complex than the primal initialization because it involves the semidefinite $C + D_\lambda \succeq 0$ constraint. The key here is that variable assignment can be viewed equivalently as moving the necessary coefficients from s_{ji} to s_{j0}. For example, assigning v_1 as false can be written as

$$z_j = 1\cdot v_1 + 1\cdot v_2 - 1\cdot v_0 \xrightarrow{v_1 := F} z_j = 0\cdot v_1 + 1\cdot v_2 - 2\cdot v_0.$$

Thus, if we look at the loss function, the coefficients c_{ij} for unassigned i and j actually remains the same in the new cost matrix $\hat{C}$. Thus, the difference in cost matrices C and $\hat{C}$ after variable assignment will only reflects in the first column and row

$$C - \hat{C} = \begin{pmatrix} 0 & \delta^T \\ \delta & 0 \end{pmatrix} \tag{18}$$

If we add a proper vector $\xi = \begin{pmatrix} \|\delta\|_1 \\ |\delta| \end{pmatrix}$, we can see that

$$C + D_{\lambda^*+\xi} = (C + D_{\lambda^*}) + \begin{pmatrix} \|\delta\|_1 & \delta^T \\ \delta & D_{|\delta|} \end{pmatrix} \succeq 0, \tag{19}$$

where λ^* is the optimal solution from the root. That is, $\lambda^* + \xi$ is a feasible solution for the new dual problem, which provides our lower bound as desired.

We also note that given an *optimal* solution of the SDP (as computed by Mixing), the optimal dual variable is simply given by

$$\lambda_i = \|V c_i\|_2. \tag{20}$$

Thus, when we solve the SDP optimally via the Mixing method, we also recover the optimal dual solution which serves as the starting point for future bounds.

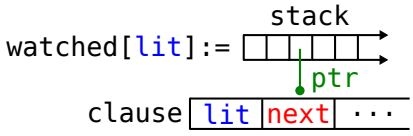

Figure 3: Illustration the watched stack.

4.3 Data structures and implementation details

Finally, there are a number of data structures and/or implementation details that are important for an efficient implementation of the branch and bound. For the full implementation and more technical details, please see the source code in https://github.com/locuslab/mixsat .

Variable ordering. We have found that the dual variable λ in (20) is also very useful in determining the resolution order (the order for splitting on variables to create the new subproblems). Thus, every time we solve an SDP, we will reorder the remaining variables according to the descending order of λ (that is, variable with larger λ_i will be resolved first).

Inference via watched stack. In order to efficiently find the set of all clauses that contain a certain variable (including after variable reordering) and efficiently inferring which clauses have been fulfilled/falsified, we use a data structure similar to the watched literal technique in SAT solvers (Moskewicz et al., 2001). Specifically, to accomplish this we used a *watched stack* data structure. For each literal, we maintain a stack of pointers, pointing to the next literal (in pivot order) in a clause. The pointer is only pushed to the stack if it's the last pointer being able to falsify the clause. This way, assigning/freeing variables only touches those clauses that haven't been satisfied, and evaluating a set of variable assignment takes exactly $O(\texttt{nnz})$ time. The watched stack is illustrated in Figure 3.

Estimating SDP objective and balancing pruning/rounding. Because the Mixing method is iterative, it is usually desirable to only solve to some given precision, and not solve the SDPs exactly. Since the Mixing method attains linear convergence (Wang, Chang, and Kolter, 2017), we estimate the converging constant by function decrease and estimate the distance to the optimal objective values. Furthermore, we need to balance between the time spent in solving SDPs and the randomized roundings (because we rounds multiple times). We have empirically found that taking rounding time proportion to the square root of the remaining variables works best.

DFS vs BFS. We have described our branch-and-bound solver in an abstract manner in Algorithm 2, where the priority queue could prioritize by any ordering. When implementing the complete solver, where the full search tree needs to be searched anyway, we let the queue Q be an stack; that is, we evaluate the nodes in the deep-first search (DFS) order.

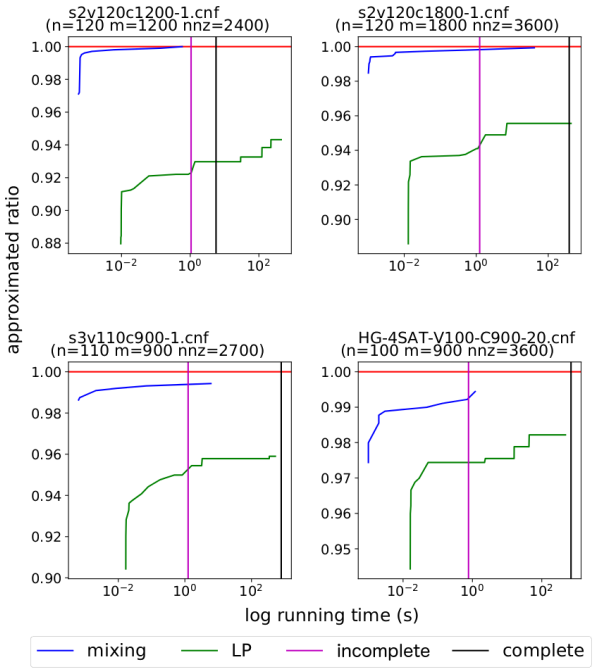

Figure 4: The approximation ratio versus running time. We demonstrated that the Mixing method not only outputs solutions faster than LPs, but also admits higher approximation ratio.

For the incomplete version, we let Q be the priority queue ordered by the clipped loss ($\sum_j \max(0, loss(V, s_j))$); that is, we explore the nodes in a best-first search (BFS) order. However, the primal and dual initialization requires the DFS order to be efficiently implemented. Thus, we run one depth-limited DFS for each of the BFS node evaluated even in the incomplete version. Specifically, we limit the depth to be less than 8 to limit the number of nodes (subproblems) pushed to the heap at each root.

5 Experimental results

In this section we present the experimental results highlighting the performance of the MIXSAT method on the relevant problems from the MAXSAT 2016 competition (later years have eliminated the "random" track of the competition, and thus don't have a sufficient number of MAX2SAT problems). Specifically, we test our solvers on all 228 the MAX2SAT instances (s2v instances) from the unweighted random categories. As a comparison point, for each problem we selected the *best* performing entry for both the complete and incomplete tracks in the MAXSAT 2016 contest; since different solvers often performed best on different problems, this sets a very high bar for comparison, though with the large caveat mentioned earlier that of course the other solvers were attempting to optimize performance over all tasks, not just MAX2SAT problems. Nonetheless, given the level of specializing in existing approaches, we feel that the comparison provides strong evidence that our SDP can achieve compelling performance in this domain. Further,

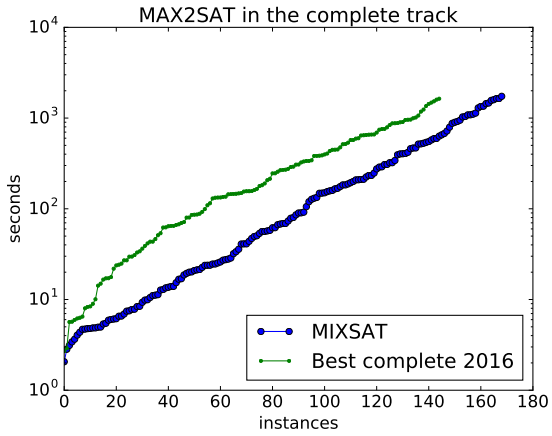

Figure 5: Experimental result of MAX2SAT in the complete track. We demonstrate that the MIXSAT algorithm consistently outperforms the best complete solvers in the MAXSAT 2016 competition in every MAX2SAT instances. Specifically, we solved 169/228 instances in 30 mins in avg 273 secs, while the best solvers solved only 145/228 instances in avg 341 secs.

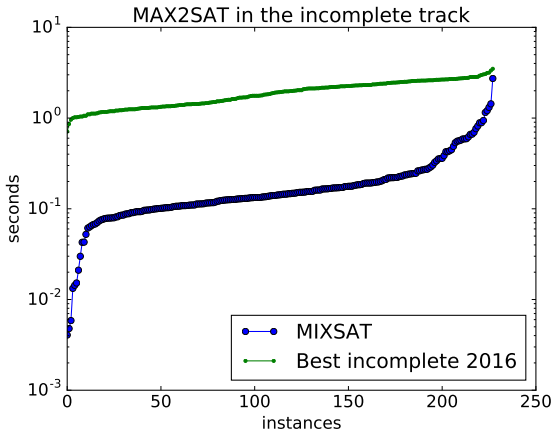

Figure 6: Experimental result of MAX2SAT in the incomplete track. We demonstrate that the MIXSAT algorithm outperform the best incomplete solvers in the MAXSAT 2016 competition in every MAX2SAT instances. The MIXSAT algorithm used 0.22 sec in average to obtain the optimal solution, while the best solvers use 1.92 sec in average. That is, we are approximately 8.72 times faster than the best solvers.

to test the performance on harder problems, we evaluated our solver on the 107 crafted MAXCUT instances in the 2016/2018 competitions, comparing it to the best solvers in 2016/2018. We also tested our solver on all the random MAX3SAT instances in 2016 to see how it generalized to different clause length. For fair comparison, we replicate the experimental setup of the 2016 environment, with exactly the same CPU (Intel Xeon E5-2620) in single core mode and a 3.5 GB memory limit.

The approximation ratio of MAXSAT SDPs. As a starting point, we first highlight the power of the MAXSAT Mixing solver combined just with randomized rounding (i.e., just looking at a single relaxation, without any branch and bound). These results are shown in Figure 4. The starting points of the blue and green curves indicate the time that the Mixing / LP solver (by Gurobi) output the solution, and the following curves indicate the highest approximation ratio generated by the LP/SDP randomized rounding procedures. We can see that the Mixing method actually output solution faster than the LP solver, and has much higher approximation ratio than LPs. Indeed, sometimes the simple randomized rounding procedure can output the optimal solution faster than the best complete and incomplete solvers (vertical black and purple lines).

Complete solvers. Next, we evaluate our solver on the above mentioned 228 MAX2SAT instances using the complete track rules, which only allows to output verified optimal solution and has a 30 minutes time limit. The result is presented in Figure 5, showing the number of solved problems versus time for MIXSAT shown against the best MAXSAT 2016 solver results. We solved **169** out of the 228 MAX2SAT instances in the random categories for the complete solvers in an average of 273 sec, while the best solvers solved only **145** instances in an average of 341 seconds (note however that the average times are not truly comparable here as we solve substantially more problems).

Incomplete solvers. We test our solvers on the 228 MAX2SAT instances using the incomplete track rules, which allows the solver to output solutions before verification and has a 5 minutes time limit. The result is presented in Figure 6. We recovered the optimal solution of all the 228 MAX2SAT instances in the random categories for the incomplete solvers in an average of 0.22 sec, while the best solvers solved in averagely 1.92 sec. That is, we are **8.72x faster** than the best solvers in average. To be specific, we found that the 2016 best solvers we compared with are usually one of CCLS (Luo et al., 2015), CnC-LS Luo et al. (2017), SC2016 (Wagner), and Swcca-ms (Cai and Su, 2013).

Crafted MAXCUT and random MAX3SAT instances. Finally, we evaluate our solver against the 107 crafted MAXCUT instances in the 2016/2018 completions, as shown in Figure 7. The result demonstrates our solver also performed well in harder instances: we recovered all optimal solution in avg 0.63 sec, which is **3x faster** comparing to the best solvers in 2016/2018 (avg 1.92s). Out of curiosity, we also tested our solver on the 144 random MAX3SAT instances in the 2016 competition. The result, shown in Figure 8, suggests that our solver is still competitive even if it is not designed for MAX3SAT. It outperformed the best solvers in 2016 on 114/144 of the instances, though didn't recover the optimal solution for 2 instances.

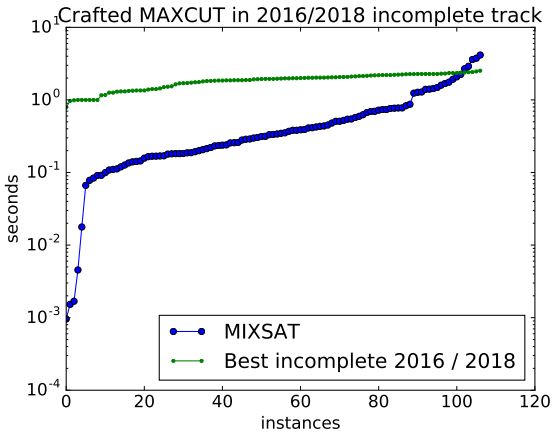

Figure 7: Experimental result of the crafted MAXCUT instances in the 2016/2018 incomplete track. The MIXSAT algorithm used 0.62 sec in average to obtain the optimal solution, while the best solvers in 2016/2018 took 1.84 sec in average. That is, we are still 3x faster in the crafted tracks.

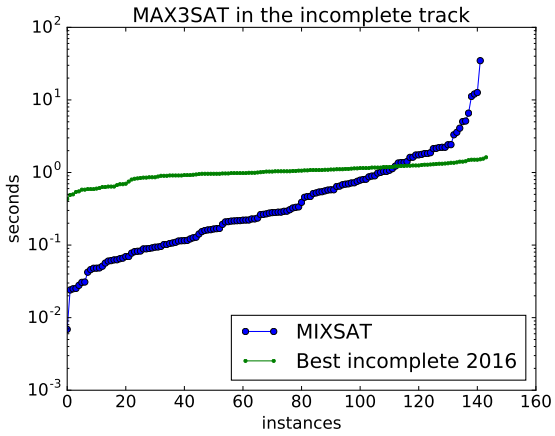

Figure 8: Experimental result of MAX3SAT in the incomplete random track. Even though our solver is not designed for MAX3SAT, it still outperformed the 2016 best solvers in 114/144 instances and recover all the optimal solutions except for 2 instances.

6 Conclusion

This paper has highlighted that an approach, based upon low-rank semidefinite programming and a simple branch-and-bound strategy, is extremely competitive for solving the MAX2SAT problem, compared with some state-of-the-art solvers in previous contest on MAXSAT solving. Further, preliminary experiments demonstrate the performance also generalizes to the MAX3SAT problem. Although the MAX2SAT/MAX3SAT domain is obviously a limited setting, the fact that an approach based upon SDP relaxations can perform well here offers substantial evidence that with modern methods for SDP solving, such relaxations can finally go beyond merely theoretically interesting, but become practical tools in the toolset of combinatorial optimization. These results suggest that they could be a powerful tool (one of many) to integrate into current solvers. More fundamentally, we hope that the work serves as a starting point for more continuous approaches integrated into combinatorial search, ultimately providing a powerful new avenue for some fundamental problems in AI.

References

Anjos, M. F. 2004. On semidefinite programming relaxations for the satisfiability problem. *Mathematical Methods of Operations Research* 60(3):349–367.

Ansótegui, C.; Bonet, M. L.; and Levy, J. 2013. Sat-based maxsat algorithms. *Artificial Intelligence* 196:77–105.

Argelich, J.; Li, C. M.; Manya, F.; and Planes, J. 2016. Max-sat-2016 eleventh max-sat evaluation. http://maxsat.ia.udl.cat/.

Berg, J., and Järvisalo, M. 2017. Cost-optimal constrained correlation clustering via weighted partial maximum satisfiability. *Artificial Intelligence* 244:110–142.

Biere, A.; Heule, M.; and van Maaren, H. 2009. *Handbook of satisfiability*, volume 185. IOS press.

Boumal, N.; Voroninski, V.; and Bandeira, A. 2016. The non-convex burer-monteiro approach works on smooth semidefinite programs. In *Advances in Neural Information Processing Systems*, 2757–2765.

Burer, S., and Monteiro, R. D. 2003. A nonlinear programming algorithm for solving semidefinite programs via low-rank factorization. *Mathematical Programming* 95(2):329–357.

Cai, S., and Su, K. 2013. Local search for boolean satisfiability with configuration checking and subscore. *Artificial Intelligence* 204:75–98.

Eén, N., and Sorensson, N. 2006. Translating pseudo-boolean constraints into sat. *Journal on Satisfiability, Boolean Modeling and Computation* 2:1–26.

Feige, U., and Goemans, M. 1995. Approximating the value of two power proof systems, with applications to max 2sat and max dicut. In *Theory of Computing and Systems, 1995. Proceedings., Third Israel Symposium on the*, 182–189. IEEE.

Fu, Z., and Malik, S. 2006. On solving the partial max-sat problem. In *International Conference on Theory and Applications of Satisfiability Testing*, 252–265. Springer.

Goemans, M. X., and Williamson, D. P. 1994. New 34-approximation algorithms for the maximum satisfiability problem. *SIAM Journal on Discrete Mathematics* 7(4):656–666.

Goemans, M. X., and Williamson, D. P. 1995. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM (JACM)* 42(6):1115–1145.

Gomes, C. P.; van Hoeve, W.-J.; and Leahu, L. 2006. The power of semidefinite programming relaxations for max-sat. In *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*, 104–118. Springer.

Karloff, H., and Zwick, U. 1997. A 7/8-approximation algorithm for max 3sat? In *Foundations of Computer Science, 1997. Proceedings., 38th Annual Symposium on*, 406–415. IEEE.

Koshimura, M.; Zhang, T.; Fujita, H.; and Hasegawa, R. 2012. Qmaxsat: A partial max-sat solver. *Journal on Satisfiability, Boolean Modeling and Computation* 8:95–100.

Krislock, N.; Malick, J.; and Roupin, F. 2017. Biqcrunch: a semidefinite branch-and-bound method for solving binary quadratic problems. *ACM Transactions on Mathematical Software (TOMS)* 43(4):32.

Le Berre, D., and Parrain, A. 2010. The sat4j library, release 2.2, system description. *Journal on Satisfiability, Boolean Modeling and Computation* 7:59–64.

Lotgering, T. 2012. Sdp-based max-2-sat decomposition.

Luo, C.; Cai, S.; Wu, W.; Jie, Z.; and Su, K. 2015. Ccls: an efficient local search algorithm for weighted maximum satisfiability. *IEEE Transactions on Computers* 64(7):1830–1843.

Luo, C.; Cai, S.; Su, K.; and Huang, W. 2017. Ccehc: An efficient local search algorithm for weighted partial maximum satisfiability. *Artificial Intelligence* 243:26–44.

Marques-Sila, J., and Planes, J. 2011. Algorithms for maximum satisfiability using unsatisfiable cores. In *Advanced Techniques in Logic Synthesis, Optimizations and Applications*. Springer. 171–182.

Martins, R.; Manquinho, V.; and Lynce, I. 2014. Open-wbo: A modular maxsat solver. In *International Conference on Theory and Applications of Satisfiability Testing*, 438–445. Springer.

Morgado, A.; Heras, F.; Liffiton, M.; Planes, J.; and Marques-Silva, J. 2013. Iterative and core-guided maxsat solving: A survey and assessment. *Constraints* 18(4):478–534.

Moskewicz, M. W.; Madigan, C. F.; Zhao, Y.; Zhang, L.; and Malik, S. 2001. Chaff: Engineering an efficient sat solver. In *Proceedings of the 38th annual Design Automation Conference*, 530–535. ACM.

Nocedal, J., and Wright, S. 2006. *Numerical optimization*. Springer, second edition.

Park, J. D. 2002. Using weighted max-sat engines to solve mpe. In *Eighteenth national conference on Artificial intelligence*, 682–687. American Association for Artificial Intelligence.

Pataki, G. 1998. On the rank of extreme matrices in semidefinite programs and the multiplicity of optimal eigenvalues. *Mathematics of operations research* 23(2):339–358.

Rendl, F.; Rinaldi, G.; and Wiegele, A. 2010. Solving max-cut to optimality by intersecting semidefinite and polyhedral relaxations. *Mathematical Programming* 121(2):307.

van Maaren, H.; van Norden, L.; and Heule, M. J. 2008. Sums of squares based approximation algorithms for max-sat. *Discrete Applied Mathematics* 156(10):1754–1779.

Wagner, M. Maxsat solver sc2016 (internal technical report).

Wang, P.-W.; Chang, W.-C.; and Kolter, J. Z. 2017. The mixing method: low-rank coordinate descent for semidefinite programming with diagonal constraints. *arXiv preprint arXiv:1706.00476*.

Zhang, L., and Bacchus, F. 2012. Maxsat heuristics for cost optimal planning. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*.