MSc thesis

Master's Programme in Computer Science

Incomplete MaxSAT Solving by Linear Programming Relaxation and Rounding

Esa Kemppainen

June 8, 2020

Faculty of Science

University of Helsinki

Supervisor(s)

Assoc. Prof. Matti Järvisalo, Dr. Jeremias Berg

Examiner(s)

Assoc. Prof. Matti Järvisalo, Dr. Jeremias Berg

Contact information

P. O. Box 68 (Pietari Kalmin katu 5)
00014 University of Helsinki,Finland

Email address: info@cs.helsinki.fi
URL: http://www.cs.helsinki.fi/

HELSINGIN YLIOPISTO – HELSINGFORS UNIVERSITET – UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Koulutusohjelma — Utbildningsprogram — Study programme
Faculty of Science	Master's Programme in Computer Science

Tekijä — Författare — Author

Esa Kemppainen

Työn nimi — Arbetets titel — Title

Incomplete MaxSAT Solving by Linear Programming Relaxation and Rounding

Ohjaajat — Handledare — Supervisors

Assoc. Prof. Matti Järvisalo, Dr. Jeremias Berg

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
MSc thesis	June 8, 2020	57 pages

Tiivistelmä — Referat — Abstract

NP-hard optimization problems can be found in various real-world settings such as scheduling, planning and data analysis. Coming up with algorithms that can efficiently solve these problems can save various rescources. Instead of developing problem domain specific algorithms we can encode a problem instance as an instance of maximum satisfiability (MaxSAT), which is an optimization extension of Boolean satisfiability (SAT). We can then solve instances resulting from this encoding using MaxSAT specific algorithms. This way we can solve instances in various different problem domains by focusing on developing algorithms to solve MaxSAT instances.

Computing an optimal solution and proving optimality of the found solution can be time-consuming in real-world settings. Finding an optimal solution for problems in these settings is often not feasible. Instead we are only interested in finding a good quality solution fast. Incomplete solvers trade guaranteed optimality for better scalability.

In this thesis, we study an incomplete solution approach for solving MaxSAT based on linear programming relaxation and rounding. Linear programming (LP) relaxation and rounding has been used for obtaining approximation algorithms on various NP-hard optimization problems. As such we are interested in investigating the effectiveness of this approach on MaxSAT. We describe multiple rounding heuristics that are empirically evaluated on random, crafted and industrial MaxSAT instances from yearly MaxSAT Evaluations. We compare rounding approaches against each other and to state-of-the-art incomplete solvers SATLike and Loandra. The LP relaxation based rounding approaches are not competitive in general against either SATLike or Loandra However, for some problem domains our approach manages to be competitive against SATLike and Loandra.

ACM Computing Classification System (CCS)
Mathematics of computing → Discrete mathematics → Combinatorics

Avainsanat — Nyckelord — Keywords

maximum satisfiability, linear programming, combinatorial optimization

Säilytyspaikka — Förvaringsställe — Where deposited

Helsinki University Library

Muita tietoja — övriga uppgifter — Additional information

Algorithms study track

Contents

1 Introduction

In real-world settings we are often confronted with the task of finding the best possible solution to a problem with a time limit. Such problems are called optimization problems. Oftentimes interesting optimization problems are NP-hard. If the solution space for an optimization problem is discrete we call it a combinatorial optimization problem [88]. NP-hard combinatorial optimization problems arise for example in planning [63], scheduling [75, 17], bioinformatics [46] and data analysis [14]. Algorithms that compute low cost solutions to these problems can save money, time and various kinds of various kinds of other resources. Therefore, there is lot of interest in developing algorithms that can be used to solve NP-hard optimization problems efficiently. In this thesis we develop an incomplete solution approach based on the so-called declarative paradigm for solving combinatorial optimization problems.

Solution approaches for combinatorial optimization problems can be divided into complete and incomplete approaches. Given enough resources, the complete approaches find an optimal solution to a problem and prove its optimality These can be further divided into problem-specific exact algorithms [12] and exact declarative methods [91]. Problem-specific exact algorithms are algorithms developed for finding an optimal solution for instances in a specific problem domain. Exact declarative methods offer a way to solve instances from different problem domains. This is done by encoding a problem instance into a constraint language for which there exists an efficient solver. A declarative solver is an implementation of an algorithm designed for solving instances of constraint optimization problems for specific constraint languages. Examples of commonly used constraint optimization problems are maximum satisfiability (MaxSAT) [70] and integer linear programming (ILP) [27, 25] both of which we detail in this thesis. Solutions for the instances resulting from encoding can be then decoded back to solutions for the original problem instance.

MaxSAT is an optimization extension of the well-known NP-complete problem known as Boolean satisfiability, or SAT [24]. The underlying constraint language used by MaxSAT is conjunctive normal form (CNF) formulas. Once a problem instance is encoded as a CNF formula, we can use a MaxSAT solver to obtain a solution for the encoding. Many different approaches have been proposed for MaxSAT solving [66, 64, 35, 5, 78, 92, 30, 13].

The most important one for this work is to encode MaxSAT as integer linear programming (ILP) [6]. ILP is also an exact declarative method used to solve combinatorial optimization problems. ILP uses linear inequalities as the underlying constraint language.

In contrast to complete solution approaches, incomplete solution approaches are designed to give the best found solution within a given time limit without guaranteeing the optimality. Unless P = NP, no complete approach is not going to be effective on all instances of an NP-hard optimization problem. Furthermore, in many real-world applications, we are more interested in finding a good quality solution efficiently than finding an optimal solution. As such there is an interest in developing incomplete approaches since they scale better with real-world applications by sacrificing the guarantee of finding an optimal solution. Incomplete approaches can be further roughly divided into two categories; approximation algorithms [103, 47, 102] and local search algorithms [50]. A local search algorithm finds a solution and tries to improve the already found solution by searching the neighboring search space. Approximation algorithms obtain solutions for NP-hard problems efficiently and the obtained solution is guaranteed to be a certain factor away from an optimal solution.

One commonly used approximation algorithm for ILP is based on using linear programming (LP) relaxation [89, 53] and rounding. Similarly to ILP, LP is a declarative method based on linear inequalities. The main difference between these two is that compared to ILP, LP does not have integrality constraints over variables which makes LP polynomial-time solvable [61]. The LP relaxation on an ILP instance creates a related LP instance with same linear inequality constraints but with the integrality constraint removed. Solving this related LP instance and rounding the solution to have integer values can create an approximation algorithm for the original ILP instance. Using LP relaxations and rounding is a well-studied approach on obtaining a solution for NP-hard optimization problems [23, 19, 2]. However, using LP relaxations and rounding to solve MaxSAT instances encoded as ILP has not gained as much attention, even if similar relaxation approaches using semidefinite programming (SDP) [40] and SDP relaxation have shown promising results on a special case of MaxSAT known as MAX-2-SAT [42] where each clause is restricted to have exactly two variables. In the referred work it was shown that good quality lower and upper bounds for optimal solutions could be found within seconds by using SDP relaxations. LP relaxations have also been shown to be effective on MAX-2-SAT in [57] and have been used to improve a complete solver by obtaining lower bounds for MAX-2-SAT and MAX-3-SAT instances [105]. In contrast to these works, we explore using an LP re-

laxation approach with rounding for the general MaxSAT problem where clauses have no restrictions on number of variables. Furthermore, we empirically evaluate the approaches using crafted and industrial benchmark instances in addition to random instances.

More specifically we investigate the quality of solutions obtained by using LP relaxations and rounding to solve MaxSAT instances. We describe multiple different rounding heuristics and empirically evaluate them by running them on MaxSAT instances used in MaxSAT Evaluations [10, 9]. We show that there are differences between rounding approaches. We also show that our approach manages to obtain good solutions on some problem domains for *non-partial* MaxSAT instances when compared to state-of-the-art solvers.

This thesis is organized as follows. In Chapter 2 we go over the definitions of MaxSAT and give an overview of approaches used to solve MaxSAT instances. In Chapter 3 we go over the definitions of ILP and LP, discuss what LP relaxation is and give an overview of central approaches to solve ILP and LP instances. In Chapter 4 we describe how to encode MaxSAT problems into ILP instances and further into LP, discuss different rounding heuristics considered in this thesis and discuss how these approaches were implemented. In Chapter 5 we present results from our empirical evaluation by comparing the considered rounding heuristics against each other and against state-of-the-art incomplete MaxSAT solvers. Chapter 6 concludes this thesis and discusses possible future work.

2 Maximum Satisfiability

As our goal is to develop an approach for solving maximum satisfiability (MaxSAT) we start by defining concepts related to MaxSAT. Firstly in Section 2.1 we define what the Boolean satsifiability problem is. In Section 2.2 we define the MaxSAT problem. Lastly in Section 2.3 we review algorithmic approaches that are used to solve MaxSAT instances.

2.1 Boolean Satisfiability

Boolean satisfiability [34], or SAT for short, is the problem of determining whether a given propositional formula is satisfiable or not. This is a well-known NP-complete problem [24]. Propositional formulas consist of atomic propositions and logical operators. A propositional formulas in conjunctive normal formal, or CNF for short, are constructed as follows.

- A literal l is either a Boolean variable x or its negation $\neg x$.
- A clause C of length m is is a disjunction $l_1 \vee \ldots \vee l_m$ of literals.
- A CNF formula F with k clauses is a conjunction $C_1 \wedge \ldots \wedge C_k$.

Any propositional logic formula can be transformed into CNF without loss of generality. This can be done so that the size of the resulting formula is linear in the size of the original formula in a standard way using the so-called Tseitin encoding [101]. A truth assignment τ sets each literal to either true or false. More formally, given a set of Boolean variables V, a truth assignment τ is a mapping $\tau : V \to \{0, 1\}$, where zero corresponds to false and one to true. For a CNF formula to evaluate to true at least one literal in each clause needs to be satisfied by a truth assignment. More precisely:

- A literal l is satisfied by τ, if l is a Boolean variable x and $\tau(x) = 1$ or if l is negated variable $\neg x$ and $\tau(x) = 0$.
- A clause $C = l_1 \vee \ldots \vee l_n$ is satisfied by τ, if $\tau(l_k) = 1$ for some $1, \ldots, n$.
- A propositional formula F, is satisfied by τ, if all clauses $C_1 \wedge \ldots \wedge C_k$ are satisfied.

Since a CNF formula consists of clauses that are connected by conjunctions, we can view a CNF formula $F = C_1 \wedge \ldots \wedge C_k$ as a set of clauses $F = \{C_1, \ldots, C_k\}$.

A SAT solver [76] is a program that, given a CNF formula F as an input, decides if the formula F have a satisfying truth assignment. Advances in SAT solvers has made them competitive in solving computationally hard problems [54, 90]. Instead of having to come up with specific algorithm for some decision problem we can encode the problem to a CNF formula and then use a SAT solver to solve the SAT instance. If a solution for the SAT instance exists, a SAT solver can return the solution for the instance which can be mapped back to a solution for the original problem instance.

Example 1 *The k-coloring is a problem, where given a graph $G = (V, E)$ and an integer k, the goal is to decide if each of the vertices can be colored so that no adjacent vertices have the same color assigned using k colors. This problem can be encoded into SAT as follows.*

A variable $x_{v,i}$ corresponds to a vertex v that has been assigned a color i. For each vertex and for each i, where $1 \leq i \leq k$, we create the following clauses.

$$\bigvee_{i=1}^{k} x_{v,i}$$

To enforce that there is only one color assigned to each vertex $v \in V$ we create the following constraints.

$$\bigwedge_{i=1}^{k-1} \bigwedge_{j=i+1}^{k} (\neg x_{v,i} \vee \neg x_{v,j})$$

To enforce that, for each edge $(v, u) \in E$, vertices v and u cannot have the same color assigned we create the following constraints.

$$\bigwedge_{i=1}^{k} (\neg x_{v,i} \vee \neg x_{u,i})$$

If the propositional formula consisting of these clauses is satisfiable, then there is an assigning of colors that colors the given graph G. If the propositional formula is unsatisfiable, then there is no assigning of colors that colors the given graph G.

2.2 MaxSAT

Maximum satisfiablity, or MaxSAT [70], is an optimization extension of the Boolean satisfiability problem. Whereas the goal in the Boolean satisfiability problem is to decide if a given Boolean formula is satisfiable, in MaxSAT the goal is to find a truth assignment that maximizes the number of satisfied clauses in a given Boolean formula. For some intuition let us consider the following example.

Example 2 *Let F be the CNF formula*

$$(x) \wedge (x \vee y) \wedge (\neg x \vee y) \wedge (x \vee \neg y) \wedge (\neg x \vee \neg y).$$

This formula is unsatisfiable. However, different truth assignments satisfy different numbers of clauses. If for example we set x to false then we can satisfy at most three clauses. On the other hand, if we set x to true, then we can satisfy four clauses as $x = 1$ satisfies clauses $(x), (x \vee y)$ and $(x \vee \neg y)$, and the remaining clauses $(\neg x \vee y)$ and $(\neg x \vee \neg y)$ can be satisfied by either setting y to true or to false.

The formula F in Example 2 is a *non-partial* MaxSAT instance. This means that there are no requirements on which clauses must be satisfied. However, in many practical problems we usually have some constraints that must be satisfied. In other words, when encoding a problem to MaxSAT we would want to force certain clauses to be true in all solutions. A *partial* MaxSAT instance may also contain clauses that must be satisfied. These types of clauses are called hard clauses. Clauses that do not have to be satisfied are called soft clauses.

Definition 1 *Let F be a partial MaxSAT instance $F = \{F_h, F_s\}$, where F_h is a set of hard clauses and F_s is a set of soft clauses. A solution for F is a truth assignment that satisfies all hard clauses of F.*

Non-partial can be seen as a special case where the set of hard clauses is empty. Hence, by the definition, any truth assignment for a *non-partial* MaxSAT instance is a solution.

Example 3 *Let F be a partial MaxSAT instance $F = \{F_h, F_s\}$, where*

$$F_h = \{(x \vee y), (x \vee z), (x \vee \neg y), (\neg x \vee \neg y)\} \text{ and}$$
$$F_s = \{(x), (y), (z)\}.$$

Here a truth assignment τ that sets x, y and z to true would not be a solution as it would not satisfy the hard clause $(\neg x \vee \neg y)$. On the other hand, a truth assignment τ' that sets x and z to true and y to not true would be a solution.

Furthermore, weights can be associated with soft clauses of MaxSAT instances. These types of instances are called weighted MaxSAT instances. Hard clauses have no assigned weights as they need to be satisfied by a truth assignment for it to be a solution. If all the weights of a MaxSAT instance are one, we call the instance an unweighted MaxSAT instance. The cost of a solution is defined as follows.

Definition 2 *Given a weighted MaxSAT instance F and a truth assignment τ, the cost of τ is the sum of all weights of the soft clauses of the F not satisfied by τ. More formally* $cost(F, \tau) = \sum_{C \in F_s} w(C) \cdot (1 - \tau(C))$.

Definition 3 *Let F be a MaxSAT formula. An optimal solution for F is a solution that has the smallest cost of all solutions.*

Now we can define the MaxSAT problem as finding a solution for a MaxSAT instance F that has the smallest cost over all solutions for F. With these definitions let us consider the following example.

Example 4 *Let F be a MaxSAT formula $F = \{F_h, F_s\}$, where*

$$F_h = \{(x \vee y), (x \vee z), (x \vee \neg y), (\neg x \vee \neg y)\} \text{ and}$$
$$F_s = \{(x, 3), (y, 2), (z, 5)\}.$$

Here a truth assignment τ that sets x to true and y and z to false has $cost(F, \tau) = 7$. While this is a solution for F it is not optimal. An optimal solution τ'sets x and z to true and y to false and has $cost(F, \tau') = 2$.

To give an example on how to encode a problem in MaxSAT let us consider the graph coloring problem already discussed in the previous section.

Example 5 *In the optimization variant of the problem presented in Example 1, the goal is to find a minimum number of colors to color a given graph $G = (V, E)$ such that no adjacent vertices have the same coloring.*

Let $G = (V, E)$ be a graph. Since there cannot be more colors used to color a graph than there are vertices we can use number of vertices as an upper bound on the number of colors we consider. A variable x_c corresponds to a color c and setting $x_c = 1$ means that the color c is used. Since the goal in MaxSAT is to satisfy as many clauses as possible, the instance contains following constraints as singleton soft clauses.

$$\bigwedge_{c=1}^{k} \neg x_c$$

Here each variable is negated since we want to minimize the number of colors used or in other words, to maximize the number of colors not used. A variable $x_{v,c}$ corresponds to having the color c assigned to the vertex v. To enforce that each vertex $v \in V$ has exactly one color assigned, the instance contains the following constraints as hard clauses:

$$(\bigvee_{c=1}^{k} x_{v,c}) \text{ and}$$
$$\bigwedge_{i=1}^{k-1} \bigwedge_{j=i+1}^{k} (\neg x_{v,c_i} \vee \neg x_{v,c_j}).$$

The first constraint forces that each vertex has at least one color assigned and the second constraint forces that for each vertex at most one color is assigned. (These kinds of cardinality constraints are very common in MaxSAT encodings and there are multiple different ways to encode them as CNF [94, 11].) To enforce that no adjacent vertices have the same color assigned, the instance contains the following constraints as hard clauses:

$$\bigwedge_{(v,u)\in E} (\neg x_{v,c} \vee \neg x_{u,c}).$$

To force a literal to be true that corresponds to a color being used, the instance contains the following constraints as hard clauses for each vertex $v \in V$:

$$\bigwedge_{c=1}^{k} (\neg x_{v,c} \vee x_c).$$

Now if a vertex v is assigned a color c, these clauses will enforce $x_c = 1$, falsifying the corresponding soft clause. An optimal solution to this MaxSAT encoding will correspond to a coloring of graph G using minimal number of colors.

2.3 MaxSAT Solving

Solution approaches to MaxSAT and more generally to combinatorial optimzation can be divided into two different categories: complete and incomplete. Complete solvers will, given enough time, will find the best possible solution and prove that it is an optimal solution. Incomplete solvers on the other hand try to find the best solution they can find in the amount of time given but will not guarantee the optimality of the found solution. In this section we will give an overview of approaches in both of these categories for solving MaxSAT instance.

2.3.1 Complete Algorithms

Approaches for MaxSAT that are SAT-based make use of SAT solvers as a subroutines and have been shown to be an effective approach to solving MaxSAT instances [4, 10, 9]. Complete MaxSAT solution approaches that are SAT-based can be split into three distinct algorithmic approaches: the linear search approach [66, 64], the core-guided approach [35, 5, 78, 3, 82, 84] and the implicit hitting set approach [92, 29].

Linear search approach finds an optimal solution by using the currently best found solution as an upper bound. Linear search approach is an upper bounding method, i.e., it starts by finding a solution and then it keeps querying SAT solver for a solution that has a better cost than the currently best found solution. First the algorithm finds any truth assignment that is a solution for a MaxSAT formula F. Once the algorithm has found such a truth assignment, a constraint which will be satisfied only if a lower cost solution is found is added creating a modified formula F'. Then a SAT solver is called on this modified formula F'. The algorithm keeps repeating this step until the solver returns that the formula is unsatisfiable, meaning that the previously found solution is an optimal solution for F. This approach is used for example by the QMaxSAT [64] solver and has been shown to be effective on some problem domains.

Where as the linear-search approach is an upper bounding method, the core-guided approach is a lower bounding method. To help understand how this approach works let us first define what a core is. A core of a MaxSAT formula F is a subset $F'_s \subset F_s$ of soft clauses such that $F'_s \cup F_h$ is not satisfiable. A core can be obtained using a SAT solver [32, 13]. A core-guided approach finds a core for a given MaxSAT formula F, then relaxes the clauses in the core by creating a new formula F' in which one of the relaxed clauses can be falsified and adding new cardinality constraints over the relaxed clauses. Then a SAT solver is called on this modified formula F'. The algorithm keeps iterating over this until the modified formula F' is satisfied. The underlying idea behind this approach is that any solution for F has to falsify at least one clause from each obtained core.

The implicit hitting set approach also extracts cores iteratively. In contrast to the core-guided approach, the implicit hitting set approach does not add cardinality constraints to the formula. Instead a so-called hitting set over the accumulated cores is computed. This is done by using integer linear programming (ILP) [93, 85, 27, 25]. Then the SAT solver is given the MaxSAT instance F', with this hitting set removed from the set of soft clauses. This is done until SAT solver returns a satisfying truth assignment, which will be an optimal solution to the MaxSAT instance [29]. This approach of not adding new constraints at each iteration aims to improve the time spent in the SAT solver step in comparison to the core-guided approach as the size of the formula given to a SAT solver does not increase at each iteration. In contrast, implicit hitting set solvers might extract more cores increasing the time spent on computing minimal hitting sets.

In addition to these SAT-based approaches the branch and bound approach has also been proposed for solving MaxSAT [48, 71, 73, 72, 18]. Branch and bound is a widely used algorithm design paradigm for solving discrete and combinatorial optimization problems [45]. While for MaxSAT this is not competitive approach for large instances, for smaller instances the branch-and-bound approach works rather well.

2.3.2 Incomplete Algorithms

The other central category of MaxSAT solvers are incomplete algorithms. Where as complete algorithms guarantee optimality of found solutions, incomplete algorithms cannot give guarantees of optimality for found solutions. The motivation for these incomplete algorithms comes from the improved scalability. Complete algorithms, while guaranteeing optimality, can be slow on some instances due to the algorithm having to find an optimal

solution and prove that the solution found is actually an optimal solution. Removing this requirement can improve the scalability of an algorithm. Especially for real-world applications, we might be much more interested in finding a good quality solution fast rather than finding the actual optimal solution. This does not necessarily mean that the incomplete algorithms would be unable to find an optimal solution. The solution given by an incomplete solver can still be optimal.

While we distinguish between complete and incomplete algorithms, complete algorithms are often used as a basis for incomplete MaxSAT solvers. These types of algorithms that are complete but can report solutions even when interrupted are called anytime algorithms. For example, the linear-search approach, while being a complete approach, is used by many incomplete solvers [80, 77, 56, 30, 13]. Since a linear-search algorithm finds intermediate solutions and then improves on the found solution, a solver using a linear-search algorithm can report the already found solutions that are not optimal even when interrupted. Core-guided and the implicit hitting set approaches by themselves produce only optimal solution as they are lower bounding approaches. However, depending on the implementation of these approaches, they can be used as anytime algorithms [7].

Local search

A local search approach for MaxSAT [22, 98, 21, 20, 69] first picks randomly a truth assignment for a MaxSAT instance F. Then it selects an unsatisfied clause and flips the truth assignment of a variable found in the clause. An issue that arises with this approach is the existence of hard clauses that must be satisfied. To combat this, solvers implementing a local search algorithm employ ways to favor satisfying hard clauses over soft clauses. For example, a solver called SATLike [69] uses a weighting scheme called Weighting-PMS which adds weights to hard clauses. Whenever SATLike cannot flip truth assignments in a way that decreases the cost it will update the weights. For unsatisfied clauses weights are increased with a certain probability and for satisfied clauses the weights are decreased with a certain probability. For hard clauses the weight increment is higher than for the soft clauses. This is to help highlight hard clauses from the soft clauses.

Linear search approximation

Linear search approximation approach have a few different implementations. For example some algorithms use enumeration of minimal correction subsets [80, 77]. A minimal

correction subset is a minimal subset of clauses for which removing them would make the formula satisfiable. Such a subset corresponds to a solution for the MaxSAT instance since all the clauses not in this subset can be satisfied by a truth assignment. This approach has shown to be promising on giving good quality solutions to MaxSAT instance compared to other incomplete solvers [77].

Open-WBO-INC [56] is another notable incomplete solver. It is based on a linear-search approach. Open-WBO-INC does uses different approximation techniques aiming at converging to a good solution faster. One such technique that Open-WBO-INC uses is that it clusters the clauses of a formula F into k different weights. Another approach is subproblem minimization. Instead of solving the weighted instance this approach solves sequence of unweighted instances. This approach aims to speeds up the process of finding a good quality solution.

LinSBPS [30] is another linear-search based solver. LinSBPS uses so the called varying resolution approach. This approach simplifies the original problem by dividing the weights of the clauses by a large number. This is done due to the fact that instances where the weights are huge, the memory requirements might increase which slows down the solving process. Decreasing the weights will improve the solving times. Once the simplified solution is solved optimally the algorithm uses smaller number to divide the weights and solves this new simplified instance again. This is done until the original problem is solved. To further boost the performance LinSBPS also implements solution-based phase saving [7].

Core-boosted linear search

One of the best incomplete solvers currently is Loandra [10]. Loandra uses a search strategy called core-boosted linear search approach which is a combination of the core-guided and linear-search approaches to solve MaxSAT instances [13]. This approach is split into two phases. The first phase uses a core-guided algorithm for trying to find an optimal solution. If the first phase finds an optimal solution the algorithm terminates. If it does not find an optimal solution the algorithm moves to second phase where it gives the found solution and working instance of the given formula to a linear-search algorithm. This linear-search phase is run until time limit or finding an optimal solution.

From the results of recent MaxSAT Evaluations the best incomplete solvers are currently Loandra, SATLike and LinSBPS for unweighted MaxSAT instances and TT-Open-WBO-

Inc [83], Loandra and Open-WBO-Inc(inc-bmo-satlike) for weighted MaxSAT instances [10].

2.3.3 MaxSAT With ILP

The approach studied in this thesis makes use of linear programming (LP) [60, 31, 100, 59, 26, 65] based on integer linear programming (ILP) formulation of MaxSAT [6, 41, 70, 74], which can also be directly used to solve MaxSAT [6, 28]. Integer linear programming is used by some SAT-based algorithms such as the implicit hitting set algorithms to compute the hitting sets. Integer linear programming can also be used directly to solve MaxSAT as well. This can be done by encoding MaxSAT to an integer linear program and then using an integer linear programming solver to solve the ILP encoding of MaxSAT formula [41]. We will detail this approach in the following chapters.

3 Linear Programming

Recall that our goal is to study the use of linear programming relaxation and rounding for incomplete MaxSAT solving. In this chapter we first present the definitions for linear programming in Section 3.1. Then in Section 3.2 we present the definitions for integer linear programming. In Section 3.3 we present the concept of linear programming relaxation and rounding. Lastly in Sections 3.4 and 3.5 we discuss about algorithmic approaches used to solve both integer linear programming and linear programming.

3.1 Linear Programming Definitions

Linear programming, or LP for short, [60, 31, 100, 59, 26, 65] is a mathematical modeling and optimization paradigm that aims to minimize a linear objective function consisting of n variables $x_1, \ldots, x_n$ while being subject to linear inequality constraints.

Definition 4 *Linear programming is an optimization problem over n variables $x_1, \ldots, x_n$, where the goal is to maximize linear objective function $f(\bar{x}) = c_1x_1 + \ldots + c_nx_n$ with coefficients $c_1, \ldots, c_n$, subject to linear inequality constraints $a_{i1}x_1 + a_{in}x_n \leq b_i$ for $1 \leq i \leq m$.*

An LP can be expressed in the following form.

$$
\begin{array}{lll}
\text{Minimize} & \sum_{j=1}^{n} c_j \cdot x_j & \\
\text{subject to} & \sum_{j=1}^{n} a_{ij} \cdot x_j \circ b_i, & i = 1, \ldots, m, \\
& x_j \in \mathbb{R}, & \\
\text{where} & \circ \in \{<, \leq, >, \geq, =\}. & (3.1)
\end{array}
$$

Definition 5 *A solution to an LP is an assignment of variables that satisfy all the linear constraints of the LP.*

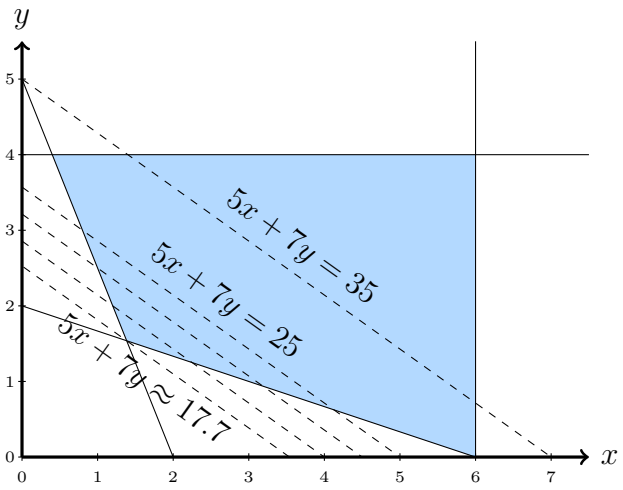


Figure 3.1: Feasible region for constraints $y \leq 4,\ x \leq 6,\ 5x + 2y \geq 10$, and $x + 3y \geq 6$.

Definition 6 *The feasible region of an LP is the set of all points that satisfies its constraints. In other words, it is the set of all solutions for an LP.*

Example 6 *In Figure 3.1 the shaded area represents the feasible region for a linear program with linear inequality constraints constraints* $y \leq 4,\ x \leq 6,\ 5x + 2y \geq 10$ *and* $x + 3y \geq 6$.

Definition 7 *A solution for an LP that minimizes the objective function value over all solutions of the LP is an optimal solution.*

If the feasible region for an LP is empty, then there are no assignments for variables that satisfy the linear constraints of the LP. In this case we say that the linear program is infeasible. To give more intuition on linear programming let us consider the following LP instance.

Example 7 *Consider the following LP.*

$$
\begin{array}{ll}
\textit{Minimize} & 5x + 7y \\
\textit{subject to} & 5x + 2y \geq 10 \\
& x + 3y \geq 6 \\
& x \leq 6 \\
& y \leq 4 \\
& x, y \geq 0.
\end{array}
$$

Now in the Figure 3.1 a dashed line represents a contour line, i.e., all points that have the same objective function value. Once we have decreased the value so that the line intersects with the last extreme point, i.e., a corner of the feasible region, we have found an optimal solution. For this linear program the optimal solution sets $x = 18/13$ *and* $y = 20/13$ *resulting in an objective function value of* $230/13 \approx 17.7$

Finding a solution to linear program can be done in polynomial time [61]. Encoding NP-hard problems compactly into LP instances cannot be done unless P = NP.

3.2 Integer Linear Programming Definitions

Linear programming is a special case of integer linear programming, or ILP [93, 85, 27, 25] for short. Similarly to a linear program, an integer linear program is an optimization problem, where the goal is to minimize a linear objective function consisting of n variables $x_1, \ldots, x_n$ while being subject to linear inequality constraints. The difference is that the problem also has integrality constraints meaning that some of the variables are forced to be integers. More formally any ILP can be expressed in the following form.

$$
\begin{array}{lll}
\text{Minimize:} & \sum_{j=1}^{n} c_j \cdot x_j & \\
\text{subject to} & \sum_{j=1}^{n} a_{ij} \cdot x_j \circ b_i, & i = 1, \ldots, m, \\
& x_j \in \mathbb{Z} & \\
\text{where} & \circ \in \{<, \leq, >, \geq, =\}. &
\end{array}
$$

As the integrality constraints in the ILP instances considered in this thesis are constrained to be binary values zero and one, we can consider special case for the ILP known as 0-1 ILP or binary ILP.

Finding a solution for this binary ILP problem is known to be NP-complete [81] as SAT can be reduced to it.

To give an example on how to model problems in ILP, let us consider the graph coloring problem and the vertex cover problem.

Example 8 *Consider again the graph coloring problem described in Example 5. Let* k *be the number of vertices. The problem can be modeled as an linear program as follows.*

$$
\begin{array}{lll}
\textit{Minimize} & \sum_{i=1}^{k} c_i & \textit{for each color } 1 \leq i \leq k \ , \\
\textit{subject to} & \sum_{i=1}^{k} x_{v,i} > 0, & \textit{for each vertex } v \in V \ , \\
& x_{v,i} + x_{u,i} \leq 1, & \textit{for each edge } (v,u) \in E \textit{ and } 1 \leq i \leq k \ , \\
& x_{v,i} - c_i \leq 0, & \textit{for each vertex } v \in V \textit{ and } 1 \leq i \leq k \ , \\
& x_{v,i}, c_i \in \{0,1\}, & \textit{for all } v \in V \textit{ and } 1 \leq i \leq k.
\end{array}
$$

The objective function corresponds to the sum of all used colors. The first constraint corresponds to forcing a single color assignment on each vertex. The second constraint corresponds to forcing adjacent vertices to not have the same assigned color. The third constraint forces assinging c_i to one if the corresponding color has been assigned to some vertex. An optimal solution to this integral linear program corresponds to an optimal solution for graph coloring problem.

Example 9 *In the vertex cover problem, we are given an undirected graph $G = (V, E)$, for which we must find a minimum set of vertices, such that each edge of the graph is incident to at least one of the vertices in the vertex cover. This problem can be formulated as an ILP as follows.*

$$
\begin{array}{lll}
\textit{Minimize} & \sum_{v \in V} x_v, & \textit{for all } v \in V \\
\textit{subject to} & x_u + x_v \geq 1, & \textit{for all edges } (u,v) \in E \\
& x_v \in \{0,1\}, & \textit{for all } v \in V
\end{array}
$$

Here $x_v = 1$ corresponds to a vertex being picked for the vertex cover. With this formulation we can find an optimal solution to an instance of the minimum vertex cover problem.

3.3 LP Relaxation and Approximations

While ILP is NP-hard, removing the integrality constraints creates a related LP problem that can be solved in polynomial time. This is called an LP relaxation [89, 53, 52, 67, 68] of the ILP. Removing the integrality constraint means that we relax the constraint $x \in \mathbb{Z}$

to a constraint $x \in \mathbb{R}$. For a 0-1 ILP we relax the binary constraint $x \in \{0, 1\}$ to a linear constraint $0 \leq x \leq 1$. Solution for the LP relaxation can be used as a lower bound on an optimal solution for the original ILP instance.

Example 10 *Consider the following ILP.*

$$\begin{aligned} &\textit{Minimize} && 5x + 7y \\ &\textit{subject to} && 5x + 2y \geq 10, \\ & && x + 3y \geq 6, \\ & && x \leq 6, \\ & && y \leq 4, \\ & && y, x \in \mathbb{N}. \end{aligned}$$

The LP relaxation of this ILP is obtained by relaxing the last constraint $y, x \in \mathbb{N}$ to $y, x \geq 0$. This results in the following LP.

$$\begin{aligned} &\textit{Minimize} && 5x + 7y, \\ &\textit{subject to} && 5x + 2y \geq 10, \\ & && x + 3y \geq 6, \\ & && x \leq 6, \\ & && y \leq 4, \\ & && y, x \geq 0. \end{aligned}$$

In Figure 3.2 we can see the feasible points for the ILP instance and feasible region for the LP relaxation is the shaded area.

Solutions for the LP relaxation can be solutions for the original ILP instance but they do not have to be. The solution might assign non-integer values for variables. For example, if the goal was to maximize the sum $x + y$ and the only linear constraint is $x + y < 2$, an optimal solution assigns x or y a fractional value. However, if solving the relaxed ILP instance assigns only integer values to the variables then it is also a solution for the original ILP problem. In the former case, where the solution to LP relaxation is not a

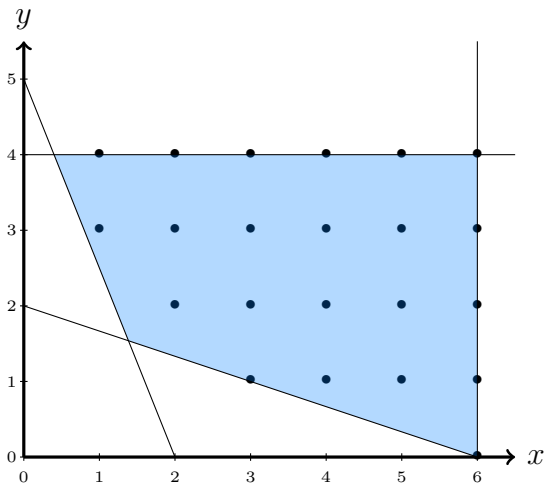


Figure 3.2: Feasible points for an ILP instance and feasible region for its LP relaxation.

solution original ILP instance, in some cases we can still use the solution for LP relaxation instance to form a solution for the ILP instance. This can be done with rounding, i.e., by rounding the non-integer values in the solution for LP relaxation to be integers. This then gives us an approximation of the optimal solution for the ILP instance. This can create an approximation algorithm for the original problem.

Definition 8 *A c-approximation algorithm gives a solution that is a factor of c away from the optimal solution for minimization problems. More formally, given an optimal solution OPT, a c-approximation algorithm results in a solution S for which $S \leq c \cdot OPT$.*

Rounding approaches are problem-specific. This means that while certain rounding procedure can result in solutions for one problem, for other problems it might not. To give intuition on how LP relaxation and rounding works, let us consider the previously mentioned vertex cover example.

Example 11 *The LP relaxation of the ILP formulation of the vertex cover problem in Example 9 results in the following linear program.*

$$
\begin{array}{lll}
\textit{Minimize} & \sum_{v \in V} c_v \cdot x_v & \\
\textit{subject to} & x_u + x_v \geq 1 & \textit{for all } \{u, v\} \in E, \\
& 0 \leq x_v \leq 1 & \textit{for all } v \in V.
\end{array}
$$

Now with this relaxation, the variables can be assigned to values between zero and one. To round the given solution to the relaxed instance we can apply a very simple rounding strategy of using $1/2$ *as a threshold for rounding. All variables with value below this threshold will be rounded to zero and variables with value equal or above this threshold will be rounded to one. Now since each edge* $(v, u) \in E$ *has a corresponding constraint* $x_v + x_u \geq 1$*, both of the vertices must have a value of at least* $1/2$*, meaning the rounding will set at least one of the variables to one. This corresponds to picking at least one vertex for each edge in the vertex cover problem. From this it follows that all solutions obtained from this rounding are solutions for the original vertex cover problem. In the worst case where every variable has been assigned a value of* $1/2$ *this rounding will result in a solution that has two times the number of vertices in vertex cover compared to an optimal solution. Therefore this rounding strategy gives us a 2-approximation algorithm for the vertex cover problem [102].*

With some graphs this approach can result in an optimal solution but not always. Consider the two following examples.

Example 12 *Let* $G = (V, E)$ *be a graph, where* $V = \{a, b, c, d, e\}$ *and* $E = \{(a, b), (a, c), (a, e), (b, c), (b, e), (e, d)\}$*. Figure 3.3 illustrates this graph. With this graph, the LP relaxation gives each vertex the following values* $a = 9/10,\ b = 3/5,\ c = 2/5,\ d = 9/10$ *and* $e = 2/5$*. Now if we use the previously mentioned rounding that uses threshold of* $1/2$ *we obtain an assignment, where* $a = 1,\ b = 1,\ c = 0,\ d = 1$ *and* $e = 0$ *giving us a solution with the objective function value of three. This is also an optimal solution to the problem instance in question.*

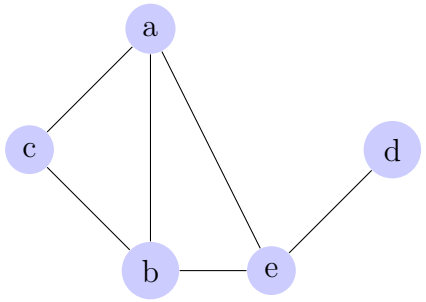


Figure 3.3: Undirected graph G

As we can see this simple rounding scheme gave us an optimal solution for a specific vertex cover instance. Now let us consider an instance where the LP relaxation and rounding does not result in an optimal solution.

Example 13 *Let $G' = (V, E)$ be a graph, where $V = \{a, b, c, d, e\}$ and there is an edge between each node (graph is a clique). Figure 3.4 illustrates this graph. Now for each vertex the LP relaxation assigns for each vertex $v \in V$ a value of $1/2$. Now if we use the same rounding strategy as before, we obtain a solution that that assigns $a = b = c = d = e = 1$ which has the objective function value of five. However, an optimal solution to this problem has four vertices in vertex cover, meaning that the solution given by the LP relaxation rounding is not optimal.*

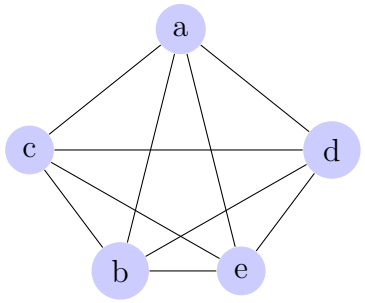


Figure 3.4: Undirected graph G'

From Examples 12 and 13 we can see that the LP relaxation rounding can result in an optimal solution to the problem in question but cannot guarantee it. Depending on the problem there are other rounding strategies that can result in better quality solutions. For example we could use randomized rounding where the rounding is decided by probability of the assigned value, i.e., a variable x_i is rounded to one with the probability of x_i and to zero with the probability of $1 - x_i$. However, depending on the problem we are trying to solve, a rounding method like this could lead to solutions that do not satisfy the linear constraints of the problem. In the vertex cover problem, for example, nothing guarantees that for each edge $(v, u) \in E$ either x_v or x_u would be assigned a value of one which corresponds to neither of them being guaranteed to be in the vertex cover.

3.4 ILP Solving

Central approaches for solving ILP problems include branch and bound [1, 99], the cutting plane method [43, 44] and the branch and cut method [87, 16]. All of these methods use the LP relaxation as a part of the solving process.

Branch and bound solves ILP problems by first solving the LP relaxation. Then the algorithm checks if all variables are integers. If so an optimal solution is found. Otherwise

the algorithm branches, i.e., bounds the non-integer variables with lower bound on the other branch and upper bound on the other branch. For example if $x = 7/3$ then branching would be done by adding new bound $x \leq 2$ on the other branch and bound $x \geq 3$ to the other branch. This is then repeated until an optimal solution is found or the ILP is found to be infeasible.

The general idea of the cutting plane method is that the LP relaxation for the ILP instance is first solved. Once we have the solution for LP instance we check if the variables are integer. If yes then we found an optimal solution and the algorithm terminates. If not we add a new constraint to the LP relaxation that essentially cuts the section of the feasible region where an optimal solution for LP was found but does not have the solution for the ILP instance.

Branch and cut can be considered a combination of branch and bound and cutting plane methods. Similarly to branch and bound and cutting planes method, branch and cut first solves the LP relaxation of the ILP instance. The algorithm then proceeds similarly to branch and bound but for each node cutting planes are searched. This helps the algorithm to have more tighter bounds on the non-integer values than regular branch and bound approach.

3.5 LP Solving

Two notable types of algorithms for solving LP instances are simplex [26, 58, 33] and interior point [59, 79, 86] methods. Depending on the problem being solved one method might work better than the other but for the most part both of these methods are competitive with each other [51].

The simplex method

The simplex algorithm (or method) is a classic optimization method for solving linear programs. The simplex method uses the fact that an optimal solution is found in one of the extreme points of the feasible region formed by the linear constraints of the LP. The algorithm starts from one extreme point and then checks if it is an optimal solution. If not it moves along the edge of the feasible region to a next extreme point and again checks the optimality of the found solution. Basically the simplex algorithm traverses along the edges of the feasible region of a linear program finding increasingly better solutions until

it finds an optimal solution. This algorithm can take exponential time in the worst case [62]. However, in most real-life applications the simplex method runs in polynomial time [95].

Interior-point method

Another central method to solve linear program is the interior-point method . Where as the simplex algorithm traversed along the edges of the feasible region the interior point method traverses through the interior of feasible region. At each iteration the algorithm checks if it has converged to one of the feasible regions extreme points. If so the algorithm stops and an optimal solution is found. Otherwise the algorithm computes a search direction, moves and again checks for convergence. The algorithm keeps doing this until it finds an optimal solution. This method has been shown to run in polynomial time [59]. Linear programs were shown to be polynomial time solvable before with ellipsoid method [61] but was too slow for practical use. Interior-point method had better worst case complexity and created more interest in developing better algorithms for solving LP problems [96, 104].

4 LP Relaxations for Incomplete MaxSAT Solving

We explore using LP relaxations and rounding to solve MaxSAT instances. As mentioned in prior chapters ILP can be used to solve MaxSAT instances. In this chapter we first go over in Section 4.1 how MaxSAT instances are encoded as ILP. In Section 4.2 we go over the LP relaxation of the ILP and present rounding procedures which we will empirically evaluate. Finally in Section 4.3 we discuss how these rounding procedures were implemented.

4.1 Encoding MaxSAT to ILP

To solve MaxSAT via ILP we use a standard encoding of MaxSAT to ILP presented in the literature [6, 41, 70, 74]. Given a MaxSAT formula F we encode it to an ILP as follows. For a clause $C_i \in F$, let C_i^+ be the set of indices of the positive literals appearing in a clause C_i and C_i^- the set of indices of the negative literals appearing in a clause C_i.

Definition 9 *Given a partial MaxSAT instance $F = (F_h, F_s, w)$, let ILP(F) be the following ILP.*

$$
\begin{aligned}
&\textit{Minimize} && \sum_{C_i \in F_s} w(C_i) \cdot b_i, \ \textit{where } w(C_i) \textit{ is the weight of clause } C_i \\
&\textit{subject to} && \sum_{j \in C_i^+} x_j + \sum_{j \in C_i^-} (1 - x_j) > 0, \ \textit{for each } C_i \in F_h \\
& && \sum_{j \in C_i^+} x_j + \sum_{j \in C_i^-} (1 - x_j) + b_i > 0, \ \textit{for each } C_i \in F_s \\
& && x, y, b_i \in \{0, 1\}.
\end{aligned}
$$

In this formulation, for each soft clause $C_i \in F$ we introduce a new binary relaxation variable b_i. The objective function takes a sum over all the weights and the corresponding relaxation variables as coefficient of the formula F. This means that for every assignment,

where $b_i = 1$, the cost of the objective function increases by the associated weight. For each clause $C \in F$ a linear inequality constraint is formed by taking the sum $\sum_{j \in C_i^+} x_j + \sum_{j \in C_i^-} (1 - x_j)$ over all the literals in a clause. In order for this sum to correspond to a satisfying truth assignment, the sum needs to be greater than zero. For the soft clauses a corresponding relaxation variable b_i is added to the sum. Now a solution for this ILP must assign either $x_j = 1$ for $j \in C_i^+$, $x_j = 0$ for $j \in C_i^-$ in each clause $C \in F$ or $b_i = 1$ for each soft clause $C \in F_s$. Therefore, any solution obtained for ILP(F) will be a solution for the MaxSAT instance F. Furthermore, any optimal solution found for the ILP(F) will be an optimal solution for F. Let us consider the following example.

Example 14 *Consider the MaxSAT instance* $F = \{(x \vee y, 3), (\neg x \vee y, 4), (x \vee \neg y, 2), (\neg x \vee \neg y, 10)\}$. *Now ILP($F$) is*

$$
\begin{aligned}
&\textit{Minimize} && 3 \cdot b_1 + 4 \cdot b_2 + 2 \cdot b_3 + 10 \cdot b_4 \\
&\textit{subject to} && x + y + b_1 > 0, \\
& && -x + y + b_2 > -1, \\
& && x - y + b_3 > -1, \\
& && -x - y + b_4 > -2, \\
& && x, y, b_i \in \{0, 1\}.
\end{aligned}
$$

An optimal solution for this integer linear program assigns $x = 0, y = 1$ *and* $b_3 = 1$ *resulting in the cost of two.*

Solving MaxSAT instances encoded this way to ILP using ILP solvers has been shown to be competitive in some problem domains [6, 28].

4.2 Rounding LP Relaxations for MaxSAT

Using LP relaxations and rounding to obtain a solution for NP-hard optimization problems is well studied [23, 19, 2, 97, 49, 15]. As discussed in the previous chapter, ILP has been found to be a rather efficient approach for some MaxSAT instances [6, 28]. Removing the integrality constraints from the ILP formulation and rounding the solution for the LP relaxation speeds up solving MaxSAT instances at the cost of losing (guaranteed)

optimality of the found solution. To show that rounding a solution for LP relaxation does not guarantee optimality, let us consider the following example.

Example 15 *Let F be the same MaxSAT instance as in Example 14. Now the LP relaxation of ILP(F) is as follows.*

$$
\begin{aligned}
&\textit{Minimize} && 3 \cdot b_1 + 4 \cdot b_2 + 2 \cdot b_3 + 10 \cdot b_4 \\
&\textit{subject to} && x + y + b_1 > 0, \\
& && -x + y + b_2 > -1, \\
& && x - y + b_3 > -1, \\
& && -x - y + b_4 > -2, \\
& && x, y, b_i \in [0, 1].
\end{aligned}
$$

An optimal solution for this LP assigns $x = y = 1/2$ and $b_i = 0$ for $1 \leq i \leq 4$. Rounding this solution to integers with a rounding strategy that sets $x = 1$ if $1/2 \leq x$ and $x = 0$ otherwise, we obtain a solution that assigns $x = y = 1$. The solution obtained using this rounding satisfies the first three constraints. The constraint $-x - y + b_4 > -2$ is not satisfied meaning that we must assign $b_4 = 1$. Assigning $b_4 = 1$ incurs a cost of 10. This is not an optimal solution since a solution that assigns $x = 0$, $y = 1$ and $b_3 = 1$ has a cost of two.

If we only consider *non-partial* MaxSAT instances and only apply rounding to non-relaxation variables, we can always assign values to relaxation variables in a way that guarantees the obtained assignment to be a solution. This is due to the fact that there are no clauses that must be satisfied in order for a truth assignment to be a solution. Rounding should hence be only applied on variables corresponding literals of the MaxSAT instance.

Rounding relaxation variables can create too tight bounds creating assignments that are not solutions for the ILP, which is why in this work we do not round relaxation variables. While relaxation variables could be rounded in theory, it requires more care and is left as future work. Consider the following example.

Example 16 *Let F be an unweighted non-partial MaxSAT instance. Assume that F has a clause $C = (x \vee y)$. ILP(F) contains the linear constraint $x + y + b_C > 0$. Assume that*

solving the LP relaxation of the ILP(F) we would obtain an assignment $x = y = b_C = 5/12$. This would still be a solution for the LP relaxation as the sum $x+y+b_C = 3 \cdot 5/12 > 1 > 0$. The rounding procedure that would round all variables below $1/2$ to zero would result in a solution that would not satisfy the linear constraint $x + y + b_C > 0$. Therefore, the assignment resulting from this rounding would not be a solution for the ILP.

The rounding methods considered in this thesis can be split into two categories: threshold rounding and iterative rounding. Threshold rounding methods solve the LP relaxation once and then apply rounding to all non-relaxation variables. Iterative rounding methods solve the LP relaxation, round at least one non-relaxation variable to either zero or one, update LP relaxation by fixing said variable to what it was rounded, i.e., add constraint $x_i = 1$ or $x_i = 0$ to the LP relaxation, and solve the updated LP relaxation. This is iteratively done until all non-relaxation variables have been fixed. The most obvious difference between these two types of rounding methods is that the threshold rounding methods are faster than the iterative rounding methods. This is due to threshold methods having to solve the LP relaxation only once. While iterative methods are slower, at each iteration a new solution obtained for the updated LP relaxation can guide the approach toward solutions of better quality.

4.2.1 Threshold Rounding

The first set of rounding methods we consider are threshold rounding methods. Algorithm 1 outlines a general threshold rounding algorithm. Given a MaxSAT instance F the algorithm first obtains the LP relaxation of ILP(F) on line 1. Then on line 3 it calls a LP solver to solve the LP relaxation and saves the obtained solution into *sol*. Finally all non-relaxation variables are rounded by a rounding procedure on lines 4-6. In this thesis

Algorithm 1: General threshold rounding

input: MaxSAT instance F

1 L' = LP relaxation of $ILP(F)$
2 *literals* = a set of all variables in F
3 sol = **solve**(L')
4 **foreach** $x \in$ *literals* **do**
5 | rounding procedure
6 **end**

we consider three threshold rounding schemes which we will refer to as SIMPLE, RANDOM and CF.

Algorithm 2 outlines SIMPLE, the first rounding approach we consider. This algorithm rounds a variable x to one if $v \geq 1/2$ and to zero otherwise, where v is the value assigned to x in the solution for the LP relaxation. More specifically, it first creates a variable v on line 1 that gets a value that was assigned for the variable x in solution *sol* for the LP relaxation. Then on line 2 it checks whether the value v is greater than or equal to $1/2$. If it is greater or equal, x is rounded to one and to zero otherwise.

Algorithm 2: Simple rounding (SIMPLE)

input: Variable x and solution *sol*

1 v = value assigned to x by solution *sol*
2 **if** $v \geq 1/2$ **then** $x = 1$
3 **else** $x = 0$

Algorithm 3 outlines the second threshold rounding approach RANDOM. This algorithm rounds a variable to one with the probability of v, where v is the value assigned to a variable x in the LP solution *sol* and to zero with the probability of $1 - v$. On line 2 the algorithm computes for each variable x a random number r between zero and one. If the number r is less than the value assigned to variable x in the *sol*, we set the variable to one and to zero otherwise. For example, if we have a solution to the LP relaxation where $x = 2/3$ and $y = 1/3$, x would be more likely set to one and y to zero. However, there is a possibility that both of these variables would be rounded to the opposite of what they were closest to, i.e., if assigned value was closer to one it would be rounded to zero and vice versa. Rounding variables this way has been shown to result in a $(1 - \frac{1}{\mathrm{e}})-$approximation algorithm [41].

Algorithm 3: Random rounding (RANDOM)

input: Variable x and solution *sol*

1 v = value assigned to x by solution *sol*
2 r = random number between 0 and 1
3 **if** $r \leq v$ **then** $x = 1$
4 **else** $x = 0$

Algorithm 4 outlines the final considered threshold rounding method CF. CF takes the previously mentioned RANDOM algorithm and combines it with a (1/2)-approximation

Algorithm 4: Coin-flip rounding (CF)

input: Variable x and solution *sol*

```
1 r1 = random number between 0 and 1
2 if 1/2 ≤ r then
3 |   RANDOM(x, sol)
4 else
5 |   r2 = random number between 0 and 1
6 |   if 1/2 ≤ r2 then x = 1
7 |   else  x = 0
8 end
```

algorithm [106, 55], where each variable is set to one with probability of 1/2 and to zero otherwise. For each variable x the algorithm first picks a random number $r1$ between zero and one at line 1. If the value of $r1$ is greater or equal to 1/2, then at line 5 CF calls the rounding procedure RANDOM. Otherwise, the algorithm picks a new random number *r2* between zero and one. If the value of *r2* is greater or equal to 1/2, then the algorithm assigns $x = 1$ and $x = 0$ otherwise. This rounding approach has been shown to be a 3/4-approximation algorithm [41].

4.2.2 Iterative Rounding

The other set of rounding approaches we focus on are the so-called iterative rounding methods. Given a MaxSAT instance F, algorithms that are iterative work by solving the LP relaxation of the $ILP(F)$, then fixing at least one variable to one or zero and then solving the instance again with the fixed variables. Compared to threshold methods,

Algorithm 5: General iterative rounding

input: MaxSAT instance F

```
1 L' = LP relaxation of ILP(F)
2 literals = a set of all literals in F
3 while length(literals) > 0 do
4 |   sol = solve(L')
5 |   rounding procedure
6 end
```

iterative methods tighten the feasible region at each iteration. This way the algorithm can be, in theory, guided towards a better quality solution after fixing a variable. On the other hand, iterative rounding can be more time-consuming as the LP relaxation must be solved multiple times before we obtain a solution. Algorithm 5 outlines the structure of the general iterative rounding algorithm. Lines 1-2 are the same as in Algorithm 1. At line 4 the algorithm starts a while loop that will terminate once the set *literals* is empty. At each iteration each rounding procedure will remove the fixed variables from this set. First step at each iteration is to solve the LP relaxation. Then the algorithm calls a rounding procedure at line 6. In this thesis we consider the following iterative rounding schemes: ITER, ITERBATCH, ITERRANDD and BORB.

Algorithm 6: (Simple) Iterative rounding (ITER)

input: Set *literals*, solution *sol* and LP relaxation L'

1 $best = x \in literals$ that maximizes $|1/2 -$ value of x in $sol\ |$
2 $v =$ value assigned to variable *best* in solution *sol*
3 **if** $v \geq 1/2$ **then** add $\mathbf{var}(best) = 1$ to L'
4 **else** add $\mathbf{var}(best) = 0$ to L'
5 remove $\mathbf{var}(best)$ from *literals*

Algorithm 6 outlines the first iterative rounding method ITER, where we round one variable to either one or zero. After the LP relaxation has been solved the algorithm finds the variable x that has been assigned value farthest away from $1/2$ in the *sol* at line 1, i.e., the variable that maximizes $|1/2 - x|$. Once such a variable is found the algorithm rounds the variable to which value it is closer to, zero or one. At line 5 the algorithm removes the now fixed variable from the *literals*.

Algorithm 7: Iterative batch rounding (ITERBATCH)

input: Set *literals*, integer k, solution *sol* and LP relaxation L'

1 $batch =$ pick k number of variables for which $|1/2 -$ value of x in $sol\ |$ is maximized
2 **foreach** $x \in batch$ **do**
3 **if** $v \geq 1/2$ **then** add $\mathbf{var}(x) = 1$ to L'
4 **else** add $\mathbf{var}(x) = 0$ to L'
5 remove $\mathbf{var}(x)$ from *literals*
6 **end**

The second iterative rounding approach we consider is ITERBATCH. Algorithm 7 outlines

this approach. Overall this algorithm works similarly to algorithm outlined in Algorithm 6. However, instead of fixing only one variable, a batch of k variables are fixed at each iteration. At line 1 IterBatch takes a k number of variables that have been assigned a value farthest away from 1/2 in the *sol*. The LP relaxation is then updated by fixing all variables in the batch to the value obtained by rounding each variable to which value they are closest to, zero or one. Then the variables in the batch are all removed from the set *literals*. The idea behind this approach is to keep the iterative approach but speed up the solving process by fixing more variables at each iteration which lowers the number of required solver calls.

Algorithm 8: Iterative randomized rounding (IterRand)

input: Set *literals*, integer k, solution *sol* and LP relaxation L'

1 *batch* = pick k number of variables for which $|1/2 -$ value of x in *sol* $|$ is maximized
2 x = uniformly at random chosen variable from *batch*
3 **if** $v \geq 1/2$ **then** add $\mathbf{var}(x) = 1$ to L'
4 **else** add $\mathbf{var}(x) = 0$ to L'
5 remove $\mathbf{var}(x)$ from *literals*

Iterative Randomized, or IterRand, rounding outlined by Algorithm 8 initially works similarly to batching as it takes a batch of variables. The difference between these two is that the IterRand at line 2 picks one variable from batch uniformly at random. The LP relaxation is then updated by fixing this randomly chosen variable to value obtained by rounding it to which value it is closer to, zero or one. Then the variable is removed from the set *literals*. Randomizing the variable selection can direct the rounding algorithm away from getting stuck in bad local optima.

Algorithm 9: Best of randomized batch rounding (BoRB)

input: Set *literals*, integer k, solution *sol* and LP relaxation L'

1 *randomBatch* = pick k number of variables uniformly at random from *literals*
2 $x = x \in$ *randomBatch* that maximizes $|1/2 -$ value of x in *sol* $|$
3 **if** $v \geq 1/2$ **then** add $\mathbf{var}(x) = 1$ to L'
4 **else** add $\mathbf{var}(x) = 0$ to L'
5 remove $\mathbf{var}(x)$ from *literals*

Best of randomized batch, or BoRB, rounding outlined in Algorithm 9 employs a more aggressive randomization. At line 1 instead of picking batch of best variables as before,

the algorithm picks uniformly at random k variables. From this randomized batch a variable that has been assigned a value that is farthest away from 1/2. The LP relaxation is then updated by fixing the variable to value obtained by rounding the variable to which value it is closer to, zero or one. Then the variable is removed from the set *literals*. Randomization in IterRand happened among the best candidates which might still leave us stuck in a bad local optima. In BoRB the randomization happens during selection of the candidate variables To balance the aggressive randomization BoRB picks a *best* candidate for rounding.

4.3 Implementation

The rounding approaches Simple, Random, CF, Iter, IterBatch, IterRand and BoRB were all implemented on top of the SCIP [38, 39] framework. SCIP is a framework for constraint integer programming that is developed at Konrad-Zuse-Zentrum für Informationstechnik Berlin. The underlying LP solver in SCIP is SoPlex [36, 37] which implements the simplex algorithm.

For each rounding approach, a given MaxSAT instance F is first transformed into the LP relaxation of $ILP(F)$ with a Python script and then passed to the rounding procedure. A simple bash script handles passing these files to rounding scripts and uses `ulimit -t <number-of-seconds>` command to make sure rounding terminates in case of timeout. We opted on implementing rounding scripts by using Python 2.7 and therefore used PySCIPOpt which is a Python interface for SCIP.

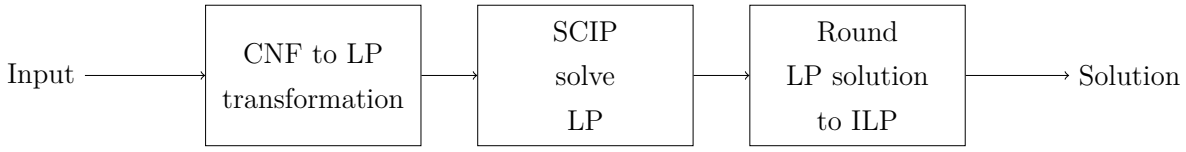


Figure 4.1: Flow of threshold script design

Scripts for Simple, Random and CF first read the problem into memory and calls SCIP to solve the LP instance. Once the instance has been solved by SCIP, a simple for loop fixes all variables using a corresponding rounding procedure. After rounding the cost of the solution is computed. The flow of these three scripts is illustrated by Figure 4.1.

Scripts for Iter, IterBatch, IterRand and BoRB were implemented by a simple while loop that terminates once all variables have been fixed. At each loop SCIP is asked

to solve the LP relaxation of the original instance with the fixed variables. After there are no variables to fix the cost of the solution is computed. Figure 4.2 illustrates the flow.

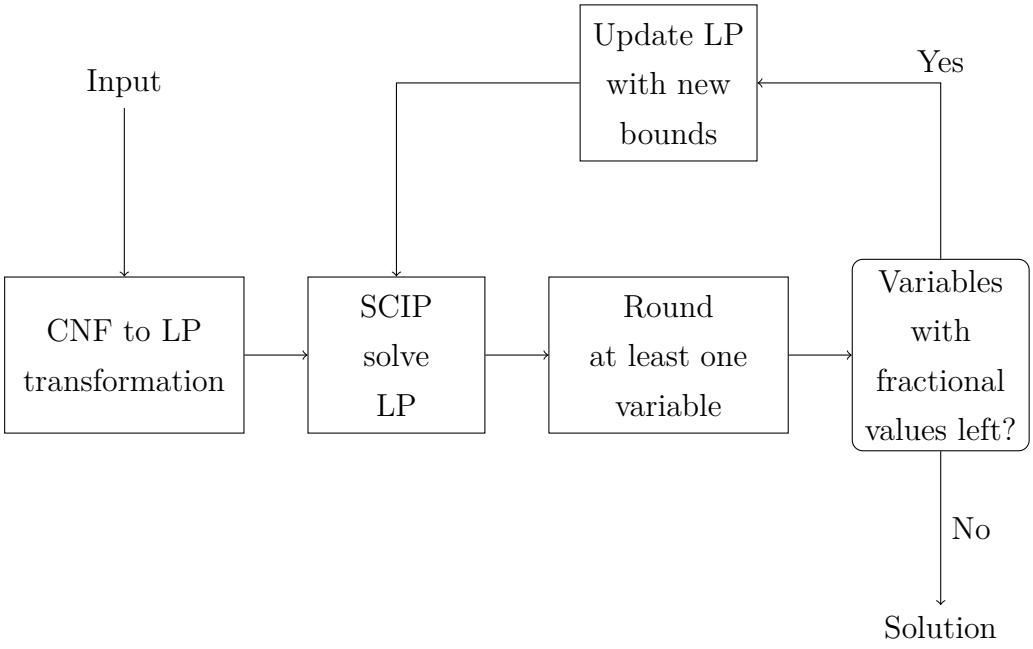


Figure 4.2: Flow of iterative script design

As a further heuristic in iterative methods we fixed all variables that were assigned to zero or to one in the solution for LP relaxation obtained by SCIP. This is done to lower the number of calls to SCIP. Otherwise many iterations would be spent on fixing only few variables to zero or one and SCIP would return same solution on the next iteration.

5 Experiments

In this chapter we empirically investigate the rounding methods described in the previous chapter. First in Section 5.1 we detail the evaluation setup. Then in Section 5.2 we present results from an empirical evaluation comparing different rounding methods using *non-partial* MaxSAT instances. In Section 5.3 we analyze rounding methods in more detail for a couple of instances from different interesting families. Next in Section 5.4 we investigate the quality of solutions obtained with the rounding approaches to state-of-the-art incomplete MaxSAT solvers. Lastly in Section 5.5 we explore using these rounding approaches to solve *partial* MaxSAT instances.

5.1 Evaluation Setup

The main focus in our experiments is on how well the considered rounding methods perform on *non-partial* MaxSAT instances. This is due to our rounding methods not being able to guarantee assignments which would satisfy hard clauses. Our benchmark set is a subset of the Master Set of MaxSAT Instances [8] which includes all instances used in recent MaxSAT Evaluations [10, 9]. The benchmark set has 618 MaxSAT instances from which 331 are unweighted and 287 are weighted instances. These benchmarks include crafted, random and industrial instances. Families included are **drmx-atmosk**, **drmx-cryptogen**, **frb**, **generalized-ising**, **maxcut** and **ramsey** for weighted instances and **maxcut**, **packup**, **ramsey** and **set-covering** for unweighted instances. All the previously mentioned rounding approaches were run on these benchmarks while enforcing a 300-second per-instance time limit.

To compare the results of each rounding method an incomplete score ranking method was used. This is a method that is used in the MaxSAT Evaluations [10, 9] for the incomplete track. For each instance a ratio of the best solution found and the solution of the solver is computed. The score of a solver s on an instance i is

$$\text{score}(s, i) = \frac{(\text{best found cost for } i) + 1}{(\text{solution by solver for instance } i) + 1}.$$

If a solver timed out it receives a score of zero for that instance. The total score for a

solver s is the average over scores for all instances:

$$\text{totalscore}(s) = \frac{1}{(\text{num of instances})} \sum_{i \in \text{instances}} \text{score}(s, i).$$

When comparing multiple solvers, a higher totalscore for a solver means that the solver obtained better quality solutions compared to other solvers in the evaluation.

5.2 Comparing Different Rounding Methods

The first experiment aims to compare all the different rounding approaches against each other. We consider an instance as solved if variables are rounded within the given time limit, i.e., for threshold methods LP relaxation must be solved once and for iterative methods must have fixed all variables within the time limit.

Table 5.1: Total score comparison with different rounding methods

Rounding method	Total score	# solved instances
Iter	0.857	553
IterRand	0.854	564
BoRB	0.848	564
Random	0.507	605
CF	0.494	605
IterBatch	0.422	597
Simple	0.316	605

As we can see from Table 5.1 iterative methods result in better quality solutions compared to threshold methods in terms of scores. However, IterBatch results in worse quality solutions than Random and CF. The simple iterative method Iter managed to score the highest with also having the lowest number of solved instances overall. Seemingly randomization of variable selection results in fewer LP solves in some instances as the iterative methods IterRand and BoRB managed to solve more instances. While from Table 5.1 we can see that total scores for iterative methods were really close to each other, from Table 5.2 we can see that Iter found the best solution in more instances than other iterative methods. This would explain why Iter scored highest in the overall comparison even while having lowest number of solved instances.

Table 5.2: Comparison of how many times a rounding method scored 1

Rounding method	# times scored 1
Iter	317
IterRand	218
BoRB	187
Random	11
CF	14
IterBatch	45
Simple	23

From Table 5.3 we can see that both IteRand and BoRB managed to obtain a better score than threshold methods in most instances against most methods and the former managed to obtain higher score against IterBatch more times than other methods. Iter seems to be held back by the number of solved instances in this comparison as many times the difference of wins is the same as difference between solved instances. Iter obtained a solution for 553 instances where as IterRand and BoRB obtained solutions for 564 instances. Furthermore, when we compare these three methods we can see in Table 5.3 that Iter scored better than IterRand in 283 instances and IterRand scored better than Iter only on 202 instances. Similarly with BoRB which scored better in 192 instances and Iter in 298 instances.

Table 5.3: Matrix representation of wins across different rounding methods. Cell represents number of instances where rounding method *M1* obtained higher score than rounding method *M2*.

M1 \ M2	Simple	Random	CF	Iter	IterBatch	IterRand	BoRB
Simple	X	103	150	52	33	42	41
Random	492	X	329	53	381	42	43
CF	447	263	X	53	383	41	41
Iter	546	546	546	X	519	**283**	**298**
IterBatch	170	215	214	58	X	45	51
IterRand	556	**557**	**558**	202	**531**	X	256
BoRB	**557**	556	**558**	192	526	213	X

Figure 5.1 gives more details for instance specific scores of Iter and IterRand. Scores for both are mostly concentrated within $0.8 - 1$ range. However, when Iter scored one

there is a wide spread of different scores given for IterRand in the range $[0.2, 1.0]$. On the other hand, when IterRand scored one, the scores for Iter are not as spread out as they are mostly within $[0.6, 1.0]$ range. Majority of the spread was caused by both weighted and unweighted instances within the **ramsey** family. This spread for scores for IterRand would further explain why Iter scored higher in the overall comparison even when it had more time outs than other iterative methods.

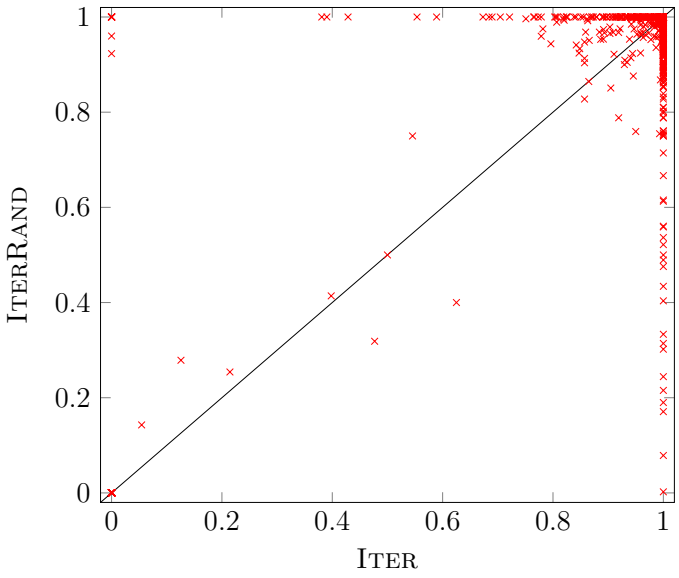


Figure 5.1: Score comparison between Iter (x-axis) and IterRand (y-axis).

Table 5.4: Total score comparison between rounding methods in different families. For each row total score was recalculated where only instances within corresponding family were considered.

Family	Simple	Random	CF	Iter	IterBatch	IterRand	BoRB
weighted	0.286	0.409	0.399	**0.822**	0.318	0.783	0.774
drmx-atmostk	0.12	0.034	0.022	**0.989**	0.137	0.878	0.863
drmx-cryptogen	0.202	0.094	0.054	**0.499**	0.006	0.498	0.497
frb	0.0	0.001	0.001	**0.999**	0.0	0.991	0.991
generalized-ising	0.961	**0.988**	0.985	0.0	0.961	0.0	0.0
maxcut	0.421	0.795	0.78	**0.982**	0.421	0.94	0.942
ramsey	0.094	0.135	0.152	**0.838**	0.432	0.79	0.744
unweighted	0.342	0.592	0.577	0.887	0.513	**0.916**	0.911
maxcut	0.425	0.752	0.753	**0.968**	0.425	0.966	0.958
packup	**1.0**	0.881	0.01	**1.0**	**1.0**	**1.0**	**1.0**
ramsey	0.122	0.171	0.252	**0.907**	0.487	0.884	0.88
set-covering	0.18	0.294	0.14	0.532	**0.887**	0.738	0.746

In Table 5.4 we can also see that the rounding approach ITER had the best total score across most instance families. The only exceptions were **generalized-ising**, where most iterative methods obtained a total score of zero making threshold rounding approach RANDOM obtain the highest total score and **set-covering**, where ITERBATCH rounding approach obtained the highest total score. Instances in **generalized-ising** have a low number of 125 variables and a high number of over 60000 clauses. These instances took over a minute to solve for the threshold methods. On the other hand, some instances of the **frb** family also have similarly high number of clauses but could be solved in seconds even by iterative methods. However, these instances have higher number of variables compared to **generalized-ising**. This would suggest that a higher clause to variable ratio makes an instance harder to solve for iterative methods. The instances that timed out were in general the ones that had high number of clauses. ITERRAND did manage to obtain highest total score for the overall unweighted category. In Table 5.1 we see that while ITER got the highest total score it solved less instances than its randomized counterparts ITERRAND and BORB. All of these 11 instances were from **set-covering** family which explains why ITER scored low when rounding methods were compared within that family.

Table 5.5: Runtime comparison of different rounding methods (in seconds).

Comparison	SIMPLE	RANDOM	CF	ITER	ITERBATCH	ITERRAND	BORB
Mean	13.181	12.710	12.746	34.684	17.150	31.195	30.688
Median	0.081	0.08	0.081	01.0465	0.079	0.878	0.857
Cumulative	8146	7854	7876	21434	10598	19278	18964
# Timeouts	13	13	13	65	21	54	54

Table 5.6: Number of instances solved within time limit for each rounding method.

Solver/Method	# solved $\leq 1s$	# solved $\leq 5s$	# solved $\leq 60s$
SIMPLE	512	547	576
RANDOM	512	548	579
CF	511	548	579
ITER	301	474	551
ITERBATCH	491	523	564
ITERRAND	319	500	551
BORB	320	503	551

From Table 5.5 we can see that iterative methods were much slower than threshold methods. The number of timeouts also increased considerably going from threshold methods to iterative. However, all rounding approaches managed to find a solution in under one second for most of the of the instances. For most instances a solution was obtained fast but a few of the instances took a long time to solve which can be seen from the fact that mean is high but median is low. This can be seen in Table 5.6. Half of the instances could be solved by any method within one second and around 500 instances could be solved within five seconds by any method. Increasing time limit to one minute does not increase the number of solved instances dramatically for any rounding method. However, there are still few instances that none of the rounding methods could solve within the given 300-second time limit.

In Figures 5.2 and 5.3 we have omitted threshold methods from the comparison. This is due to threshold methods producing poor quality solutions in general. However, a single threshold runtime Threshold is plotted as a reference on how time consuming a single LP solve can be. From Figure 5.2, which shows times for the entire benchmark set, we can see clear gap between iterative methods and threshold methods. Iterative methods outside of IterBatch can be seen to need more time to solve more instances. IterBatch being able to solve more instances faster is due to the fact that it requires fewer LP solver calls compared to the other iterative methods. IterBatch is still noticeably slower than the threshold methods.

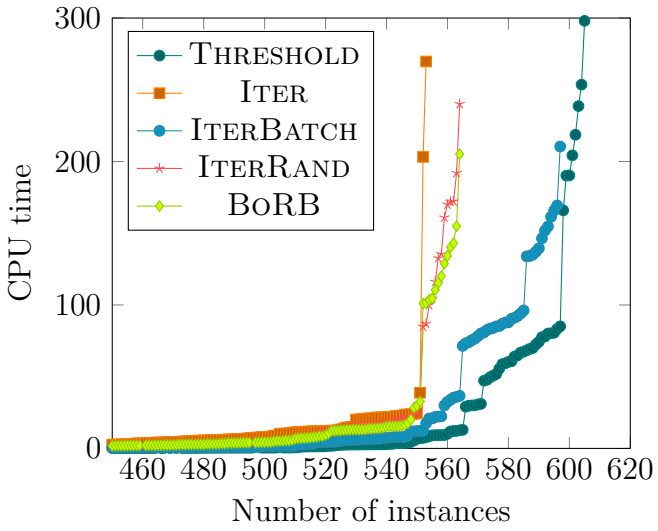
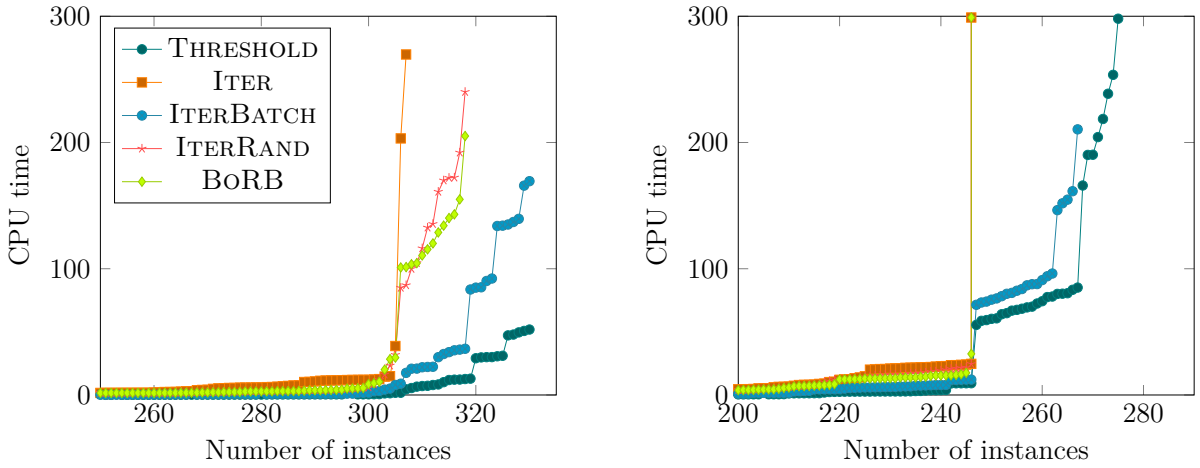


Figure 5.2: Entire benchmark set



(a) Unweighted instances

(b) Weighted instances

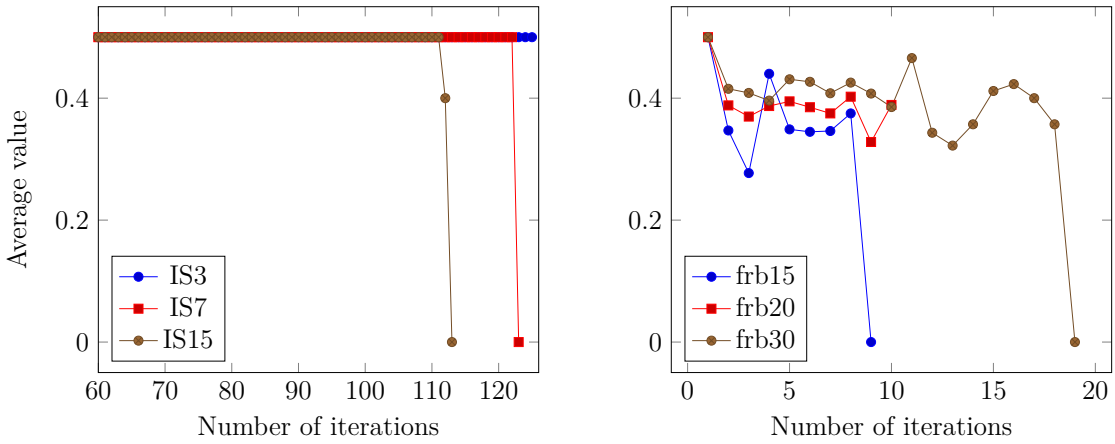
Figure 5.3: Figures where where runtimes are divided to (a) unweighted and (b) weighted instances.

From Figure 5.3 (a), which shows times for unweighted instances, we can see similar divide between methods and IterBatch rounding is more separated from threshold methods. in Figure 5.3 (b), which shows times for weighted instances, the rounding methods are rather close to each other up until around 240 number of instances. After that the iterative methods outside of IterBatch do not solve more instances even with a 300-second time limit. As we can see, solving these instances took a lot of time even for threshold methods. If even a single LP solver call can take up 200 or even 300 seconds it is clear that an iterative method solving instances multiple times would result in time outs. Weighted instances seem to be harder for the LP solver to solve compared to unweighted instances.

5.3 LP Relaxation Solution Analysis

In this section we take a more detailed look into two MaxSAT families: **generalized-ising** and **frb**. These two families were chosen as they were the most divisive families between threshold and iterative methods. Rounding approaches also were competitive with state-of-the-art solvers for these families. We are especially interested in investigating the assignments for variables in the LP relaxation solutions and how fixing variables affects the assignments in the next iteration.

We will start by considering instances from the **generalized-ising**. Iterative methods did not manage to obtain a solution for any of the instances from this family. We ran three randomly selected instances from this family without a time limit using Iter. The

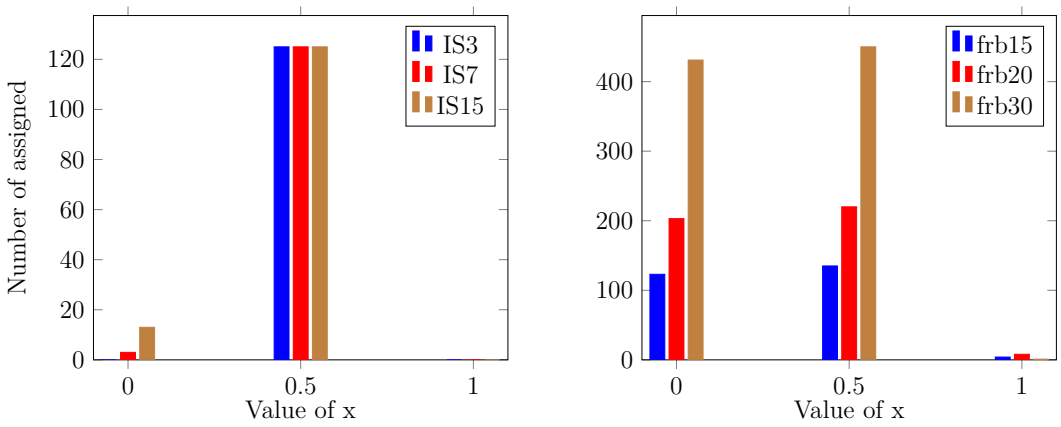


(a) **generalized-ising** (b) **frb**

Figure 5.4: Plots of averages of the obtained values for variables for each iteration for (a) instances of the **generalized-ising** family and (b) instances of the **frb** family. These are obtained using ITER.

instances chosen all had 125 variables each and 60375 clauses, where each clause had on average three variables. From the Figure 5.4 (a) we can see that fixing a variable had no noticeable effect on next obtained solution. First solution obtained for the LP relaxation assigned all variables to 1/2 and after fixing a variable, all variables were again assigned a value of 1/2. Only when a majority of the variables were fixed, two of the instances, IS7 and IS15, obtained solutions where some of the variables were assigned a value of zero. It took ITER 122 iterations before a solution for the LP relaxation assigned different values from 1/2 to variables for the IS7 instance and 111 iterations for the IS15 instance. From Figure 5.5 (a) we can see that no other values than 1/2 and zero were assigned for these instances. LP relaxations do not offer enough new information after fixing variables for the iterative methods to be viable for these instances. For these instances using a threshold methods will result in same or similar assignments. Therefore, depending on how their solution quality compares to state-of-the-art solvers, for similar instances a threshold method should be used.

Next we will consider instances from the **frb** family. Iterative methods managed to obtain much better scores than threshold methods in this family. Threshold methods scored zero or close to zero where as iterative methods scored close to one with the exception of ITERBATCH. Three instances from **frb** were chosen randomly and they were ran without a time limit using ITER. These instances were more varied than the chosen **generalized-ising** instances in terms of how many variables and clauses each instance had. The number of variables ranged from 135 to 450 and similarly the number of clauses ranged

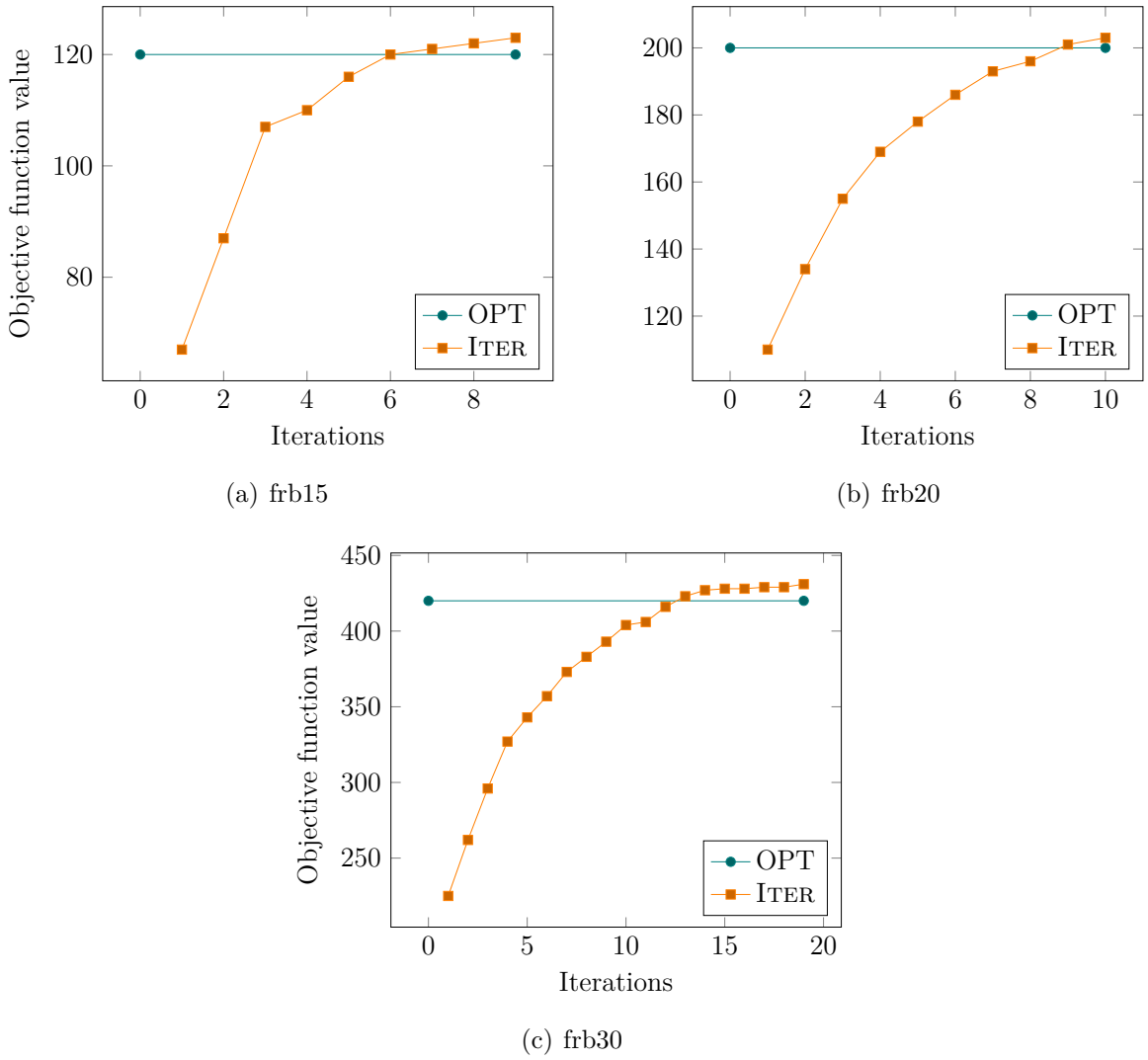


(a) **generalized-ising** (b) **frb**

Figure 5.5: Spread of assigned values for variables in the obtained solutions (fixed variables are excluded) for all iterations for (a) instances of the **generalized-ising** family and (b) instances of the **frb** family. These are obtained using ITER.

from 2631 to 18350. For each instance the average number of variables in a clause was two. From Figure 5.4 (b) we can see that fixing variables affected assignments obtained from solving LP relaxation more than in the instances of the **generalized-ising** family. Initially all the variables have been assigned a value of 1/2 but immediately on the next iteration, the solutions for the updated LP relaxations has assigned different values for variables. For these instances, the solutions for LP relaxations after fixing variables seem to help to guide the algorithm towards a solution better and require fewer iterations compared to **generalized-ising**. From Figure 5.5 (b) we can see that the variables in these instances were only assigned values zero, one and 1/2. This means that a solution for LP relaxation, outside of value 1/2, assigned mostly a value of zero to all variables after fixing some variables. Since many variables were assigned to zero after a single fixed variable it is clear that threshold methods would not obtain a good quality solution as all the variables were assigned a value of 1/2 initially. For example using SIMPLE for these instances would assign all variables a value of one which results in the following objective function values: 339456 for frb15, 1245114 for frb20 and 8072900 for frb30. The optimal objective function values are 120, 200 and 420 respectively. From Figure 5.6 we can see that ITER obtained solutions that have objective function value that is very close to an optimal solution values. The exact objective function values obtained by ITER for these instances are: 123 for frb15, 203 for frb20 and 431 for frb30.

Seemingly when the clauses to variables ratio of the instance is high, as in the instances of **generalized-ising** family, the solutions obtained to LP relaxations do not have assign-



(a) frb15

(b) frb20

(c) frb30

Figure 5.6: Objective function value on each iteration compared to an objective value of an optimal solution.

ments that would help fixing more variables to integer values. Similar results have been reported for special case of MaxSAT called MAX-2-SAT, where clauses are restricted to exactly two variables [42]. This suggests that benefits of iterative methods might be lost on these types of instances as the solution obtained will be same or close to same as a solution obtained by threshold methods. When the clauses to variables ratio of the instance is lower, as in the instances of **frb**, fixing variables has greater effect on the next obtained LP relaxation solution. For these types of instances the benefits of iterative methods become more apparent.

5.4 Comparison with State-of-the-Art Solvers

To see how LP relaxation based approach for incomplete MaxSAT solving compares to current state-of-the-art incomplete MaxSAT solvers, we ran SATLike and Loandra on the benchmark set. Both of these solvers have good results in recent MaxSAT Evaluations for the incomplete track [10, 9]. For this section the total score is re-computed with respect to all considered rounding methods and newly considered solvers in Table 5.7.

Table 5.7: Total score comparison with state-of-the-art solvers (*non-partial* instances)

Solver/Method	Total score	# solved instances
SATLike	0.969	618
Loandra	0.946	618
IterRand	0.671	564
Iter	0.669	553
BoRB	0.668	564
Random	0.408	605
CF	0.395	605
IterBatch	0.324	597
Simple	0.246	605

Table 5.8: Score comparison between rounding methods and state-of-the-art solvers in different families. For each row score was recalculated where only instances within corresponding family.

Family	Simple	Random	CF	Iter	IterBatch	IterRand	BoRB	SATLike	Loandra
weighted	0.215	0.332	0.324	0.646	0.232	0.621	0.617	0.936	**0.976**
drmx-atmostk	0.035	0.007	0.004	0.716	0.043	0.664	0.649	0.567	**1.0**
drmx-cryptogen	0.002	0.001	0.0	0.499	0.006	0.497	0.497	0.952	**0.98**
frb	0.0	0.001	0.001	0.973	0.0	0.966	0.966	**1.0**	0.999
generalized-ising	0.945	0.971	0.968	0.0	0.945	0.0	0.0	**0.998**	0.99
maxcut	0.348	0.661	0.648	0.813	0.348	0.776	0.778	**1.0**	0.974
ramsey	0.07	0.081	0.064	0.386	0.16	0.365	0.347	**0.982**	0.934
unweighted	0.273	0.475	0.457	0.689	0.404	0.715	0.712	**0.997**	0.92
maxcut	0.333	0.595	0.596	0.76	0.333	0.755	0.748	**1.0**	0.942
packup	**1.0**	0.881	0.01	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**
ramsey	0.1	0.134	0.187	0.567	0.299	0.561	0.563	**0.983**	0.975
set-covering	0.162	0.265	0.124	0.499	0.778	0.677	0.685	**0.996**	0.781

As we can see in the Table 5.7 all of the LP-based methods are very far behind state-of-the-art solvers in general. state-of-the-art solvers managed to obtain a solution on all

instances and in general resulted in better quality solutions. Even the best score iterative methods obtained is 0.671 which is much lower than the worst score obtained by state-of-the-art solvers which is 0.946. From Table 5.8 we can see that on some instance families the rounding approaches appear to be competitive with SATLike and Loandra. On the **drmx-atmostk** instances SATLike scored lower than ITER, ITERRAND and BORB. On the **frb** instances the same iterative methods managed to score highly, well above 0.9. Threshold methods, while generally produce generally low quality solutions, scored highly on the **generalized-ising** instances with scores well above 0.9. Rounding methods managed to obtain a better overall score for unweighted instances than for weighted instances. However, in unweighted families all rounding methods still performed poorly outside of the **packup** instances in wich almost all rounding methods managed to score one. On the **set-covering** instances ITERBATCH rounding managed to score similarly to Loandra. This shows that LP relaxation and rounding approach can result in good quality solutions on some benchmarks even if in general the solution quality is much worse.

5.5 Results for Partial MaxSAT Instances

The final experiment aims to evaluate our rounding approach on *partial* MaxSAT instances. Especially we are interested in how many times rounding methods could find an actual solution for these instances. Recall that our rounding methods do not guarantee that the obtained assignment is a solution for a *partial* MaxSAT instance. The benchmark set we used for this experiment was a subset of weighted and unweighted MaxSAT instances that were used to test incomplete MaxSAT solvers at MaxSAT Evaluation 2019 [10]. We filtered out *non-partial* instances from this set and instances that were too time consuming to transform into LP instances. There was a total of 330 instances from which 176 were unweighted and 154 were weighted instances. Similarly to our *non-partial* MaxSAT benchmarks, this benchmark set included crafted, random and industrial instances.

As we can see from Table 5.9, using LP relaxations and rounding to solve *partial* MaxSAT instances does not result in solutions for most of the instances. Iterative methods managed to find more solutions compared to threshold methods. As threshold methods do not gain any information if linear constraint is being satisfied or not after rounding variables this is to be expected. At every iteration iterative methods can ask LP solver for a solution that in theory would guide rounding algorithm towards a solution. Similarly to *non-partial* instances *partial* instances had much more timeouts for iterative methods in comparison

to threshold methods. However, even the iterative methods did not manage to obtain a solution for majority of the instances.

Table 5.9: *Partial* benchmark results

Rounding method	# solved	timed out	infeasible
Simple	29	26	275
Random	10	26	294
CF	24	26	280
Iter	43	99	188
IterBatch	26	44	260
IterRand	47	70	213
BoRB	46	58	226

Table 5.10: Total score comparison with state-of-the-art solvers (restricted *partial* instances)

Solver/Method	total score	# solved instances
SATLike	0.909	44
Loandra	0.859	39
IterRand	0.794	43
Iter	0.789	43
BoRB	0.77	42
IterBatch	0.309	16
Simple	0.206	11
CF	0.002	13
Random	0.0	0

To compare the solution quality we ran the instances where at least one of the rounding methods found a solution and no rounding method timed out on state-of-the-art solvers. These restrictions were made as including all instances lowered the score for all rounding methods significantly. There were 44 considered instances with these restrictions. From Table 5.10 we can see that the solution quality for the iterative methods is rather competitive when compared to state-of-the-art solvers. Introducing hard clauses to MaxSAT instances using this approach requires more work to be done as clearly our implementation does not result in enough solutions for this to be a viable approach. However, the solution quality for the found solutions suggests that a better implementation using this approach could be competitive with state-of-the-art solvers.

6 Conclusions

Using LP relaxations and rounding as an approximation algorithm for solving NP-hard optimization problems is a well studied approach. However, this approach has not received much attention in MaxSAT research. This thesis focused on using LP relaxations as a way to solve MaxSAT instances and more specifically investigated the quality of the solutions obtained by this approach. We defined the MaxSAT to ILP formulation used and the multiple rounding heuristics considered. We presented experimental results of using these rounding methods on multiple MaxSAT instances that are used to evaluate incomplete MaxSAT solvers. These results showed comparison with the presented rounding heuristics and comparison against current state-of-the-art incomplete MaxSAT solvers. Our implementation of using LP relaxation and rounding on MaxSAT did not result in a competitive approach in general. Depending on the instance, solutions for LP relaxations do not seem to offer much guidance on how variables should be assigned. However, for some problem domains this approach did show promising results and as such this work could be extended.

There are a few ways to extend this work. As many instances obtained a solution very fast, these solutions could be used as further constraints on the original LP relaxation and start a new round of iterations. This way rounding algorithms could be forced to find new and better quality solutions under a longer time limit. Due to being able to obtain a solution fast this approach could also be implemented in a SAT/IP hybrid solver, where the initial assignment is obtained by LP relaxation and rounding and then SAT-based approach would improve this found solution. Other implementation related future work would be to implement a better variable selection which would take into consideration on what clauses are not being satisfied. This becomes especially apparent in *partial* MaxSAT instances were all rounding approaches resulted in infeasible solutions for most instances. Once we end up with an infeasible solution a rollback could be implemented to change fixed assignments until the solution is feasible again. Future work could also take a further look into instances where rounding approaches resulted in good quality solutions and if this knowledge could be used to improve solution quality in general.

Bibliography

[1] A.Land and A. Doig. "An Automatic Method of Solving Discrete Programming Problems". In: *Econometrica* 28.3 (1960), pp. 497–520.

[2] N. Alon and A. Srinivasan. "Improved Parallel Approximation of a Class of Integer Programming Problems". In: *Algorithmica* 17.4 (1997), pp. 449–462.

[3] M. Alviano, C. Dodaro, and F. Ricca. "A MaxSAT Algorithm Using Cardinality Constraints of Bounded Size". In: *IJCAI*. AAAI Press, 2015, pp. 2677–2683.

[4] C. Ansótegui, M. L. Bonet, and J. Levy. "SAT-based MaxSAT algorithms". In: *Artif. Intell.* 196 (2013), pp. 77–105.

[5] C. Ansótegui, M. L. Bonet, and J. Levy. "Solving (Weighted) Partial MaxSAT through Satisfiability Testing". In: *SAT*. Vol. 5584. Lecture Notes in Computer Science. Springer, 2009, pp. 427–440.

[6] C. Ansótegui and J. Gabàs. "Solving (Weighted) Partial MaxSAT with ILP". In: *CPAIOR*. Vol. 7874. Lecture Notes in Computer Science. Springer, 2013, pp. 403–409.

[7] C. Ansótegui and J. Gabàs. "WPM3: An (in)complete algorithm for weighted partial MaxSAT". In: *Artif. Intell.* 250 (2017), pp. 37–57.

[8] F. Bacchus. *Master Set of MaxSat Instances*. 2020. URL: http://www.cs.toronto.edu/maxsat-lib/maxsat-instances/master-set/ (visited on 05/01/2020).

[9] F. Bacchus, M. Järvisalo, and R. Martins. "MaxSAT Evaluation 2018: New Developments and Detailed Results". In: *J. Satisf. Boolean Model. Comput.* 11.1 (2019), pp. 99–131.

[10] F. Bacchus, M. Järvisalo, and R. Martins. *MaxSAT Evaluation 2019*. 2019. URL: https://maxsat-evaluations.github.io/2019/ (visited on 04/01/2020).

[11] O. Bailleux and Y. Boufkhad. "Efficient CNF Encoding of Boolean Cardinality Constraints". In: *CP*. Vol. 2833. Lecture Notes in Computer Science. Springer, 2003, pp. 108–122.

[12] R. Bellman. "Dynamic Programming Treatment of the Travelling Salesman Problem". In: *J. ACM* 9.1 (1962), pp. 61–63.

[13] J. Berg, E. Demirovic, and P. J. Stuckey. “Core-Boosted Linear Search for Incomplete MaxSAT”. In: *CPAIOR*. Vol. 11494. Lecture Notes in Computer Science. Springer, 2019, pp. 39–56.

[14] J. Berg and M. Järvisalo. “Optimal Correlation Clustering via MaxSAT”. In: *ICDM Workshops*. IEEE Computer Society, 2013, pp. 750–757.

[15] D. Bertsimas and R. V. Vohra. “Rounding algorithms for covering problems”. In: *Math. Program.* 80 (1998), pp. 63–89.

[16] R. E. Bixby and E. K. Lee. “Solving a Truck Dispatching Scheduling Problem Using Branch-and-Cut”. In: *Oper. Res.* 46.3 (1998), pp. 355–367.

[17] M. Bofill, M. Garcia, J. Suy, and M. Villaret. “MaxSAT-Based Scheduling of B2B Meetings”. In: *CPAIOR*. Vol. 9075. Lecture Notes in Computer Science. Springer, 2015, pp. 65–73.

[18] B. Borchers and J. Furman. “A Two-Phase Exact Algorithm for MAX-SAT and Weighted MAX-SAT Problems”. In: *J. Comb. Optim.* 2.4 (1998), pp. 299–306.

[19] J. Byrka, F. Grandoni, T. Rothvoß, and L. Sanità. “Steiner Tree Approximation via Iterative Randomized Rounding”. In: *J. ACM* 60.1 (2013), 6:1–6:33.

[20] S. Cai, C. Luo, J. Lin, and K. Su. “New local search methods for partial MaxSAT”. In: *Artif. Intell.* 240 (2016), pp. 1–18.

[21] S. Cai, C. Luo, J. Thornton, and K. Su. “Tailoring Local Search for Partial MaxSAT”. In: *AAAI*. AAAI Press, 2014, pp. 2623–2629.

[22] B. Cha, K. Iwama, Y. Kambayashi, and S. Miyazaki. “Local Search Algorithms for Partial MAXSAT”. In: *AAAI/IAAI*. AAAI Press / The MIT Press, 1997, pp. 263–268.

[23] C. Chekuri and V. Madan. “Constant Factor Approximation for Subset Feedback Set Problems via a new LP relaxation”. In: *SODA*. SIAM, 2016, pp. 808–820.

[24] S. A. Cook. “The Complexity of Theorem-Proving Procedures”. In: *STOC*. ACM, 1971, pp. 151–158.

[25] H. P. Crowder, E. L. Johnson, and M. W. Padberg. “Solving Large-Scale Zero-One Linear Programming Problems”. In: *Oper. Res.* 31.5 (1983), pp. 803–834.

[26] G. Dantzig. *Linear programming and extensions*. Princeton Univ. Press, 1965.

[27] G. B. Dantzig, D. R. Fulkerson, and S. M. Johnson. “Solution of a Large-Scale Traveling-Salesman Problem”. In: *Oper. Res.* 2.4 (1954), pp. 393–410.

[28] J. Davies and F. Bacchus. “Exploiting the Power of MIP Solvers in MAXSAT”. In: *SAT*. Vol. 7962. Lecture Notes in Computer Science. Springer, 2013, pp. 166–181.

[29] J. Davies and F. Bacchus. “Solving MAXSAT by Solving a Sequence of Simpler SAT Instances”. In: *CP*. Vol. 6876. Lecture Notes in Computer Science. Springer, 2011, pp. 225–239.

[30] E. Demirovic and P. J. Stuckey. “Techniques Inspired by Local Search for Incomplete MaxSAT and the Linear Algorithm: Varying Resolution and Solution-Guided Search”. In: *CP*. Vol. 11802. Lecture Notes in Computer Science. Springer, 2019, pp. 177–194.

[31] J. Dunagan and S. S. Vempala. “A simple polynomial-time rescaling algorithm for solving linear programs”. In: *STOC*. ACM, 2004, pp. 315–320.

[32] N. Eén and N. Sörensson. “Temporal induction by incremental SAT solving”. In: *Electron. Notes Theor. Comput. Sci.* 89.4 (2003), pp. 543–560.

[33] J. J. H. Forrest and J. A. Tomlin. “Implementing the Simplex Method for the Optimization Subroutine Library”. In: *IBM Syst. J.* 31.1 (1992), pp. 11–25.

[34] J. Franco and J. Martin. “A History of Satisfiability”. In: *Handbook of Satisfiability*. Vol. 185. Frontiers in Artificial Intelligence and Applications. IOS Press, 2009, pp. 3–74.

[35] Z. Fu and S. Malik. “On Solving the Partial MAX-SAT Problem”. In: *SAT*. Vol. 4121. Lecture Notes in Computer Science. Springer, 2006, pp. 252–265.

[36] A. M. Gleixner, D. E. Steffy, and K. Wolter. “Improving the accuracy of linear programming solvers with iterative refinement”. In: *ISSAC*. ACM, 2012, pp. 187–194.

[37] A. M. Gleixner, D. E. Steffy, and K. Wolter. “Iterative Refinement for Linear Programming”. In: *INFORMS J. Comput.* 28.3 (2016), pp. 449–464.

[38] A. Gleixner, M. Bastubbe, L. Eifler, T. Gally, G. Gamrath, R. L. Gottwald, G. Hendel, C. Hojny, T. Koch, M. E. Lübbecke, S. J. Maher, M. Miltenberger, B. Müller, M. E. Pfetsch, C. Puchert, D. Rehfeldt, F. Schlösser, C. Schubert, F. Serrano, Y. Shinano, J. M. Viernickel, M. Walter, F. Wegscheider, J. T. Witt, and J. Witzig. *The SCIP Optimization Suite 6.0*. ZIB-Report 18-26. Zuse Institute Berlin, 2018. URL: http://nbn-resolving.de/urn:nbn:de:0297-zib-69361.

[39] A. Gleixner, M. Bastubbe, L. Eifler, T. Gally, G. Gamrath, R. L. Gottwald, G. Hendel, C. Hojny, T. Koch, M. E. Lübbecke, S. J. Maher, M. Miltenberger, B. Müller, M. E. Pfetsch, C. Puchert, D. Rehfeldt, F. Schlösser, C. Schubert, F. Serrano, Y. Shinano, J. M. Viernickel, M. Walter, F. Wegscheider, J. T. Witt, and J. Witzig. *The SCIP Optimization Suite 6.0*. Technical Report. Optimization Online, 2018. URL: http://www.optimization-online.org/DB_HTML/2018/07/6692.html.

[40] M. X. Goemans and D. P. Williamson. "Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming". In: *J. ACM* 42.6 (1995), pp. 1115–1145.

[41] M. X. Goemans and D. P. Williamson. "New 3/4-Approximation Algorithms for the Maximum Satisfiability Problem". In: *SIAM J. Discrete Math.* 7.4 (1994), pp. 656–666.

[42] C. P. Gomes, W. J. van Hoeve, and L. Leahu. "The Power of Semidefinite Programming Relaxations for MAX-SAT". In: *CPAIOR*. Vol. 3990. Lecture Notes in Computer Science. Springer, 2006, pp. 104–118.

[43] R. E. Gomory. "Outline of an Algorithm for Integer Solutions to Linear Programs *and* An Algorithm for the Mixed Integer Problem". In: *50 Years of Integer Programming*. Springer, 2010, pp. 77–103.

[44] M. Grötschel, M. Jünger, and G. Reinelt. "A Cutting Plane Algorithm for the Linear Ordering Problem". In: *Oper. Res.* 32.6 (1984), pp. 1195–1220.

[45] R. Grymin and S. Jagiello. "Fast Branch and Bound Algorithm for the Travelling Salesman Problem". In: *CISIM*. Vol. 9842. Lecture Notes in Computer Science. Springer, 2016, pp. 206–217.

[46] J. Guerra and I. Lynce. "Reasoning over Biological Networks Using Maximum Satisfiability". In: *CP*. Vol. 7514. Lecture Notes in Computer Science. Springer, 2012, pp. 941–956.

[47] A. Gupta, V. Nagarajan, and R. Ravi. "Approximation Algorithms for Optimal Decision Trees and Adaptive TSP Problems". In: *Math. Oper. Res.* 42.3 (2017), pp. 876–896.

[48] F. Heras, J. Larrosa, and A. Oliveras. "MiniMaxSAT: An Efficient Weighted Max-SAT solver". In: *J. Artif. Intell. Res.* 31 (2008), pp. 1–32.

[49] D. S. Hochbaum. "Approximation Algorithms for the Set Covering and Vertex Cover Problems". In: *SIAM J. Comput.* 11.3 (1982), pp. 555–556.

[50] H. H. Hoos and T. Stützle. *Stochastic Local Search: Foundations & Applications*. Elsevier / Morgan Kaufmann, 2004.

[51] T. Illés and T. Terlaky. "Pivot versus interior point methods: Pros and cons". In: *Eur. J. Oper. Res.* 140.2 (2002), pp. 170–190.

[52] T. S. Jaakkola, D. A. Sontag, A. Globerson, and M. Meila. "Learning Bayesian Network Structure using LP Relaxations". In: *AISTATS*. Vol. 9. JMLR Proceedings. JMLR.org, 2010, pp. 358–365.

[53] K. Jain. "A Factor 2 Approximation Algorithm for the Generalized Steiner Network Problem". In: *Combinatorica* 21.1 (2001), pp. 39–60.

[54] M. Järvisalo, D. Le Berre, O. Roussel, and L. Simon. "The International SAT Solver Competitions". In: *AI Magazine* 33.1 (2012).

[55] D. S. Johnson. "Approximation Algorithms for Combinatorial Problems". In: *J. Comput. Syst. Sci.* 9.3 (1974), pp. 256–278.

[56] S. Joshi, P. Kumar, S. Rao, and R. Martins. "Open-WBO-Inc: Approximation Strategies for Incomplete Weighted MaxSAT". In: *J. Satisf. Boolean Model. Comput.* 11.1 (2019), pp. 73–97.

[57] S. Joy, J. E. Mitchell, and B. Borchers. "A branch and cut algorithm for MAX-SAT and weighted MAX-SAT". In: *Satisfiability Problem: Theory and Applications*. Vol. 35. DIMACS Series in Discrete Mathematics and Theoretical Computer Science. DIMACS/AMS, 1996, pp. 519–536.

[58] G. Kalai. "A Subexponential Randomized Simplex Algorithm (Extended Abstract)". In: *STOC*. ACM, 1992, pp. 475–482.

[59] N. Karmarkar. "A New Polynomial-Time Algorithm for Linear Programming". In: *STOC*. ACM, 1984, pp. 302–311.

[60] J. A. Kelner and D. A. Spielman. "A randomized polynomial-time simplex algorithm for linear programming". In: *STOC*. ACM, 2006, pp. 51–60.

[61] L. Khachiyan. "A Polynomial Algorithm in Linear Programming". In: *Soviet Mathematics Doklady* 20 (1979), pp. 191–194.

[62] V. Klee and G. J. Minty. "How good is the simplex algorithm". In: *Inequalities* 3.3 (1972), pp. 159–175.

[63] A. Klose and A. Drexl. "Facility location models for distribution system design". In: *Eur. J. Oper. Res.* 162.1 (2005), pp. 4–29.

[64] M. Koshimura, T. Zhang, H. Fujita, and R. Hasegawa. “QMaxSAT: A Partial Max-SAT Solver”. In: *JSAT* 8.1/2 (2012), pp. 95–100.

[65] F. Kuhn. “Local Approximation of Covering and Packing Problems”. In: *Encyclopedia of Algorithms*. Springer, 2008.

[66] D. Le Berre and A. Parrain. “The Sat4j library, release 2.2”. In: *J. Satisf. Boolean Model. Comput.* 7.2-3 (2010), pp. 59–6.

[67] L. Leahu and C. P. Gomes. “LP as a Global Search Heuristic Across Different Constrainedness Regions”. In: *CP*. Vol. 3709. Lecture Notes in Computer Science. Springer, 2005, p. 853.

[68] L. Leahu and C. P. Gomes. “Quality of LP-Based Approximations for Highly Combinatorial Problems”. In: *CP*. Vol. 3258. Lecture Notes in Computer Science. Springer, 2004, pp. 377–392.

[69] Z. Lei and S. Cai. “Solving (Weighted) Partial MaxSAT by Dynamic Local Search for SAT”. In: *IJCAI*. ijcai.org, 2018, pp. 1346–1352.

[70] C. L. Li and F. Manyà. “MaxSAT, Hard and Soft Constraints”. In: *Handbook of Satisfiability*. Vol. 185. Frontiers in Artificial Intelligence and Applications. IOS Press, 2009, pp. 613–631.

[71] C. L. Li, F. Manyà, N. O. Mohamedou, and J. Planes. “Resolution-based lower bounds in MaxSAT”. In: *Constraints An Int. J.* 15.4 (2010), pp. 456–484.

[72] C. Li, F. Manyà, and J. Planes. “Exploiting Unit Propagation to Compute Lower Bounds in Branch and Bound Max-SAT Solvers”. In: *CP*. Vol. 3709. Lecture Notes in Computer Science. Springer, 2005, pp. 403–414.

[73] C. Li and Z. Quan. “An Efficient Branch-and-Bound Algorithm Based on MaxSAT for the Maximum Clique Problem”. In: *AAAI*. AAAI Press, 2010.

[74] V. M. Manquinho, J. P. Marques-Silva, and J. Planes. “Algorithms for Weighted Boolean Optimization”. In: *SAT*. Vol. 5584. Lecture Notes in Computer Science. Springer, 2009, pp. 495–508.

[75] F. Manyà, S. Negrete, C. Roig, and J. R. Soler. “A MaxSAT-Based Approach to the Team Composition Problem in a Classroom”. In: *AAMAS Workshops (Visionary Papers)*. Vol. 10643. Lecture Notes in Computer Science. Springer, 2017, pp. 164–173.

[76] J. P. Marques-Silva, I. Lynce, and S. Malik. “Conflict-Driven Clause Learning SAT Solvers”. In: *Handbook of Satisfiability*. Vol. 185. Frontiers in Artificial Intelligence and Applications. IOS Press, 2009, pp. 131–153.

[77] J. Marques-Silva, F. Heras, M. Janota, A. Previti, and A. Belov. “On Computing Minimal Correction Subsets”. In: *IJCAI*. IJCAI/AAAI, 2013, pp. 615–622.

[78] J. Marques-Silva and J. Planes. “Algorithms for Maximum Satisfiability using Unsatisfiable Cores”. In: *DATE*. ACM, 2008, pp. 408–413.

[79] S. Mehrotra. “On the Implementation of a Primal-Dual Interior Point Method”. In: *SIAM Journal on Optimization* 2.4 (1992), pp. 575–601.

[80] C. Mencía, A. Previti, and J. Marques-Silva. “Literal-Based MCS Extraction”. In: *IJCAI*. AAAI Press, 2015, pp. 1973–1979.

[81] R. E. Miller and J. W. Thatcher, eds. *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*. The IBM Research Symposia Series. Plenum Press, New York, 1972.

[82] A. Morgado, C. Dodaro, and J. Marques-Silva. “Core-Guided MaxSAT with Soft Cardinality Constraints”. In: *CP*. Vol. 8656. Lecture Notes in Computer Science. Springer, 2014, pp. 564–573.

[83] A. Nadel. “Anytime Weighted MaxSAT with Improved Polarity Selection and Bit-Vector Optimization”. In: *FMCAD*. IEEE, 2019, pp. 193–202.

[84] N. Narodytska and F. Bacchus. “Maximum Satisfiability Using Core-Guided MaxSAT Resolution”. In: *AAAI*. AAAI Press, 2014, pp. 2717–2723.

[85] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. Wiley interscience series in discrete mathematics and optimization. Wiley, 1988.

[86] Y. E. Nesterov and A. Nemirovskii. *Interior-point polynomial algorithms in convex programming*. Vol. 13. Siam studies in applied mathematics. SIAM, 1994.

[87] M. Padberg and G. Rinaldi. “A Branch-and-Cut Algorithm for the Resolution of Large-Scale Symmetric Traveling Salesman Problems”. In: *SIAM Review* 33.1 (1991), pp. 60–100.

[88] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, 1982.

[89] P. Raghavan and C. D. Thompson. “Randomized rounding: a technique for provably good algorithms and algorithmic proofs”. In: *Combinatorica* 7.4 (1987), pp. 365–374.

[90] J. Rintanen, K. Heljanko, and I. Niemelä. “Planning as satisfiability: parallel plans and algorithms for plan search”. In: *Artif. Intell.* 170.12-13 (2006), pp. 1031–1080.

[91] F. Rossi, P. van Beek, and T. Walsh, eds. *Handbook of Constraint Programming*. Vol. 2. Foundations of Artificial Intelligence. Elsevier, 2006.

[92] P. Saikko, J. Berg, and M. Järvisalo. “LMHS: A SAT-IP Hybrid MaxSAT Solver”. In: *SAT*. Vol. 9710. Lecture Notes in Computer Science. Springer, 2016, pp. 539–546.

[93] A. Schrijver. *Theory of linear and integer programming*. Wiley-Interscience series in discrete mathematics and optimization. Wiley, 1999.

[94] C. Sinz. “Towards an Optimal CNF Encoding of Boolean Cardinality Constraints”. In: *CP*. Vol. 3709. Lecture Notes in Computer Science. Springer, 2005, pp. 827–831.

[95] D. A. Spielman and S. Teng. “Smoothed analysis of algorithms: why the simplex algorithm usually takes polynomial time”. In: *STOC*. ACM, 2001, pp. 296–305.

[96] G. Strang. “Karmarkar’s algorithm and its place in applied mathematics”. In: *he Mathematical Intelligencer* 9 (1987), pp. 4–10.

[97] Y. Takazawa, S. Mizuno, and T. Kitahara. “Approximation algorithms for the covering-type k-violation linear program”. In: *Optimization Letters* 13.7 (2019), pp. 1515–1521.

[98] J. Thornton, S. Bain, A. Sattar, and D. N. Pham. “A Two Level Local Search for MAX-SAT Problems with Hard and Soft Constraints”. In: *Australian Joint Conference on Artificial Intelligence*. Vol. 2557. Lecture Notes in Computer Science. Springer, 2002, pp. 603–614.

[99] J. A. Tomlin. “Technical Note - An Improved Branch-and-Bound Method for Integer Programming”. In: *Oper. Res.* 19.4 (1971), pp. 1070–1075.

[100] L. Trevisan, G. B. Sorkin, M. Sudan, and D. P. Williamson. “Gadgets, Approximation, and Linear Programming”. In: *SIAM J. Comput.* 29.6 (2000), pp. 2074–2097.

[101] G. S. Tseitin. “On the Complexity of Derivation in Propositional Calculus”. In: 1983.

[102] V. V. Vazirani. *Approximation algorithms*. Springer, 2001.

[103] D. P. Williamson and D. B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.

[104] M. H. Wright. "The interior-point revolution in constrained optimization". In: *High Performance Algorithms and Software in Nonlinear Optimization*. Ed. by R. De Leone, A. Murli, P. M. Pardalos, and G. Toraldo. Boston, MA: Springer US, 1998, pp. 359–381. ISBN: 978-1-4613-3279-4. DOI: 10.1007/978-1-4613-3279-4_23. URL: https://doi.org/10.1007/978-1-4613-3279-4_23.

[105] Z. Xing and W. Zhang. "MaxSolver: An efficient exact algorithm for (weighted) maximum satisfiability". In: *Artif. Intell.* 164.1-2 (2005), pp. 47–80.

[106] M. Yannakakis. "On the Approximation of Maximum Satisfiability". In: *SODA*. ACM/SIAM, 1992, pp. 1–9.