Reducing SAT to Max2SAT

Carlos Ansótegui[1] and **Jordi Levy**[2]
[1]Logic & Optimization Group (LOG), University of Lleida
[2]IIIA-CSIC
carlos.ansotegui@udl.cat, levy@iiia.csic.es

Abstract

In the literature we find reductions from 3SAT to Max2SAT. These reductions are based on the usage of a gadget, i.e., a combinatorial structure that allows translating constraints of one problem to constraints of another. Unfortunately, the generation of these gadgets lacks an intuitive or efficient method. In this paper, we provide an efficient and constructive method for Reducing SAT to Max2SAT and show empirical results of how MaxSAT solvers are more efficient than SAT solvers solving the translation of hard formulas for Resolution.

1 Introduction

The SAT problem asks to determine whether there is an assignment to the Boolean variables in a Boolean formula in Conjunctive Normal Form (CNF) that satisfies all clauses (constraints). Its optimization version, the Maximum Satisfiability (MaxSAT) problem is the task of finding an assignment to the variables of the formula such that a maximum number of clauses is satisfied.

In this paper, we review the reductions from 3SAT into Max2SAT from [Garey *et al.*, 1976] and [Trevisan *et al.*, 2000] (allowing to prove that Max2SAT is NP-Complete). These reductions are based on the usage of a *gadget*, i.e., a finite combinatorial structure that allows translating constraints of one problem to constraints of another. Obviously, we also have to ensure that the gadgets can be constructed in polynomial time.

Unfortunately, in general, the gadgets available in the literature lack an intuitive and efficient construction method. As stated in [Trevisan *et al.*, 2000]: "Despite their importance, the construction of gadgets has always been a "black art" with no general methods of construction known". The first known gadget from 3SAT to Max2SAT was provided by [Garey *et al.*, 1976] and later on in [Trevisan *et al.*, 2000] a gadget of better *quality*, actually optimal, was automatically computed.

In general, we are not just interested in any gadget but those of high quality. Roughly speaking, a (α, β)-gadget from a family of constraints $\mathcal{F}_1$ to $\mathcal{F}_2$ is a translation of every $\mathcal{F}_1$ constraint into β-many $\mathcal{F}_2$ constraints such that when the original constraint is falsified, $\alpha - 1$ many new constraints are falsified, and when the original constraint is satisfied, α new constraints are satisfied. One of the main applications of gadgets is to translate results of in/approximability from one constraint problem to another. A ρ-approximation algorithm for a family of constraint problems $\mathcal{F}$ is a polynomial algorithm that computes an assignment that satisfies a fraction ρ of the maximum number of satisfiable constraints. If we have a (α, β)-gadget from $\mathcal{F}_1$ to $\mathcal{F}_2$ and a ρ-approximation algorithm for $\mathcal{F}_2$, then we can obtain a $(1 - \alpha(1 - \rho))$-approximation algorithm for $\mathcal{F}_1$. Therefore, the smaller the parameter α is, the better the translation works. Notice that, for $\alpha = 1$, the translation is somehow *perfect*. [Trevisan *et al.*, 2000], using an optimal (smallest possible α) gadget from 3SAT into Max2SAT, and [Feige and Goemans, 1995] 0.931-approximability result for Max2SAT, were able to prove 0.801-approximability of Max3SAT. However, this approximability bound was later improved with direct techniques, and nowadays it doesn't seem easy to improve approximability or inapproximability results with gadgets.

Equipped with gadgets from 3SAT to Max2SAT, SAT can be trivially reduced to Max2SAT by first reducing SAT to 3SAT using the *folklore gadget* that splits a kSAT clause into a set of 3SAT clauses. As we will see, the concatenation of these gadgets is not optimal, even for $k = 4$.

Our first contribution in this paper is to provide a new generation method for gadgets from kSAT into Max2SAT to which we refer as the *regular gadget*. Additionally, we prove the new gadget can be refined by the application of the MaxSAT resolution rule, obtaining a gadget with the best α reported so far, for arbitrary k. We also show that the gadget seems to be optimal up to $k = 5$ by using a MIP solver to prove it.

Our second contribution is to prove the usefulness of gadgets to boost SAT solvers. This is an additional application of gadgets not reported so far. In particular, we present an approach that translates SAT into Max2SAT and applies a MaxSAT solver. We experiment with several gadgets on SAT instances encoding the Pigeon Hole principle and show the efficiency of the new approach. Finally, we sketch a base algorithm that can be used to design a new generation of SAT solvers.

2 Preliminaries

A k-ary *constraint function* is a Boolean function $f : \{0,1\}^k \rightarrow \{0,1\}$. A *constraint family* is a set $\mathcal{F}$ of con-

straint functions (with possibly distinct arities). A *constraint*, over variables $V = \{x_1, \ldots, x_n\}$ and constraint family $\mathcal{F}$, is a pair formed by a k-ary constraint function $f \in \mathcal{F}$ and a subset of k variables, noted $f(x_{i_1}, \ldots, x_{i_k})$, or $f(\overline{x})$ for simplicity. A *(weighted) constraint problem*, over variables V and constraint family $\mathcal{F}$, is a multiset of pairs (weight, constraint) over V and $\mathcal{F}$, where the weight is a rational number, noted $P = \{_{(w_1)}\, f_1(x^1_{i_1}, \ldots, x^1_{i_{k_1}}), \ldots, _{(w_m)}\, f_m(x^m_{i_1}, \ldots, x^m_{i_{k_m}})\}$. In this paper we will focus on constraint functions that may be represented as (weighted) clauses $_{(w)}\, l_1 \vee \cdots \vee l_k$, where l_i's are literals and w a rational weight. An *assignment* is a function $I : \{x_1, \ldots, x_n\} \to \{0, 1\}$. We say that an assignment I *satisfies* a constraint $f(x_{i_1}, \ldots, x_{i_k})$, if $I(f(\overline{x})) =_{\text{def}} f(I(x_{i_1}), \ldots, I(x_{i_k})) = 1$. The value of an assignment I for a constraint problem $P = \{_{(w_i)}\, f_i(\overline{x})\}_{i=1,\ldots,m}$, is the sum of the weights of the constraints that this assignment satisfies, i.e. $I(P) = \sum_{i=1}^{m} w_i\, I(f_i(\overline{x}))$. An assignment is said to be *optimal* for a constraint problem if it maximizes its value.

We refer to MaxEkSAT as the constraint family defined by the constraint functions of the form $f(x_1, \ldots, x_k) = l_1 \vee \cdots \vee l_k$, where every l_i may be either x_i or $\overline{x}_i$.

We refer to Max2SAT as the union of the constraint families MaxEiSAT, for $0 \leq i \leq k$, and to MaxSAT as the union for every $i \geq 0$.

Definition 1. *An* (α, β)-gadget *from* $\mathcal{F}_1$ *to* $\mathcal{F}_2$ *is a function that, for any constraint* $f(\overline{x})$ *over* $\mathcal{F}_1$ *returns a weighted constraint problem* $P = \{_{(w_i)}\, g_i(\overline{x}, \overline{b})\}_{i=1,\ldots,m}$ *over* $\mathcal{F}_2$ *and variables* $\{\overline{x}\} \cup \{\overline{b}\}$*, where* $\overline{b}$ *are fresh variables distinct from* $\overline{x}$*, such that* $\beta = \sum_{i=1}^{m} w_i$ *and, for any assignment* $I : \{\overline{x}\} \to \{0, 1\}$*:*

1. *If* $I(f(\overline{x})) = 1$*, for any extension of* I *to* $I' : \{\overline{x}\} \cup \{\overline{b}\} \to \{0, 1\}$*,* $I'(P) \leq \alpha$ *and there exists one of such extension with* $I'(P) = \alpha$*.*
2. *If* $I(f(\overline{x})) = 0$*, for any extension of* I *to* $I' : \{\overline{x}\} \cup \{\overline{b}\} \to \{0, 1\}$*,* $I'(P) \leq \alpha - 1$*.*

   Additionally, if there exist one of such extensions with $I'(P) = \alpha - 1$*, the gadget is said to be* strict*.*

An optimal gadget is a gadget of minimum α*.*

Lemma 1. *The concatenation of a* (α_1, β_1)*-gadget from* $\mathcal{F}_1$ *to* $\mathcal{F}_2$ *and a* (α_2, β_2)*-gadget from* $\mathcal{F}_2$ *to* $\mathcal{F}_3$ *results into a* $(\beta_1\,(\alpha_2 - 1) + \alpha_1,\ \beta_1\beta_2)$*-gadget from* $\mathcal{F}_1$ *to* $\mathcal{F}_3$*.*

Proof. The first gadget multiplies the total weight of constraints by β_1, and the second by β_2. Therefore, the composition multiplies it by $\beta = \beta_1\beta_2$.

For any assignment, if the original constraint is falsified, the optimal extension after the first gadget satisfies constraints with a weight of $\alpha_1 - 1$, and falsifies the rest $\beta_1 - (\alpha_1 - 1)$. The second gadget satisfies constraints for a weight of $\alpha_2 - 1$ of the falsified plus α_2 of the satisfied. Therefore, the composition satisfies constraints with a total weight $(\alpha_2 - 1)(\beta_1 - (\alpha_1 - 1)) + \alpha_2(\alpha_1 - 1) = \beta_1(\alpha_2 - 1) + \alpha_1 - 1$.

If the original constraint is satisfied, the optimal extension after the first gadget satisfies α_1 and falsifies the rest $\beta_1 - \alpha_1$. After the second gadget the weight of satisfied constraints is $(\alpha_2 - 1)(\beta_1 - \alpha_1) + \alpha_2\alpha_1 = \beta_1(\alpha_2 - 1) + \alpha_1$.

The difference between both situations is one, hence the composition is a gadget, and $\alpha = \beta_1(\alpha_2 - 1) + \alpha_1$. □

3 From 3SAT to Max2SAT

In order to show that the Max2SAT problem is NP-hard, it suffices to show that the 3SAT problem can be reduced to the Max2SAT problem. One way to perform such reduction is through the usage of gadgets.

Imagine we are given a 3SAT formula. The idea is to replace every clause into the 3SAT formula by a set of Max2SAT clauses polynomial on the size of the clause. The replacement strategy is precisely what we call a gadget. If any clause has fewer than 3 literals, we can simply not apply the gadget.

Lemma 2 ([Garey *et al.*, 1976])**.** *Given a 3SAT clause* $x_1 \vee x_2 \vee x_3$*, the set of Max2SAT clauses:*

$$
\begin{array}{ll}
_{(1)}\, x_1 & _{(1)}\, b_1 \\
_{(1)}\, x_2 & _{(1)}\, \overline{b}_1 \vee x_1 \\
_{(1)}\, x_3 & _{(1)}\, \overline{b}_1 \vee x_2 \\
_{(1)}\, \overline{x}_1 \vee \overline{x}_2 & _{(1)}\, \overline{b}_1 \vee x_3 \\
_{(1)}\, \overline{x}_1 \vee \overline{x}_3 & \\
_{(1)}\, \overline{x}_2 \vee \overline{x}_3 &
\end{array}
$$

Defines a $(7, 10)$*-gadget from 3SAT to Max2SAT, where* b_1 *is an auxiliary variable.*

Proof. Notice that if an assignment satisfies $x_1 \vee x_2 \vee x_3$, then exactly 7 of the 10 Max2SAT clauses can be satisfied. On the contrary, if an assignment does not satisfy $x_1 \vee x_2 \vee x_3$, then exactly 6 of the 10 Max2SAT clauses can be satisfied. □

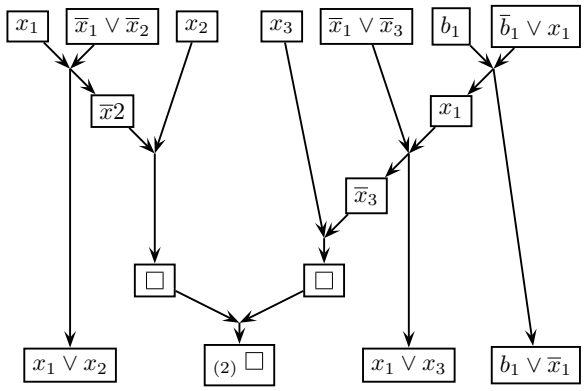
Let φ be a 3SAT formula of m_1 unary, m_2 binary and m_3 ternary clauses, and φ' the resulting Max2SAT formula after replacing every ternary clause by the Max2SAT clauses from an (α, β)-gadget from 3SAT into Max2SAT. Then, φ is satisfiable iff the maximum number of satisfied clauses in φ' is $m_1 + m_2 + \alpha\, m_3$. As concluded in [Garey *et al.*, 1976], since 3SAT reduces to Max2SAT, it follows that Max2SAT (as a decision problem) is NP-complete.

4 MaxSAT Equivalent Gadgets

We can further refine the $(7, 10)$-gadget from [Garey *et al.*, 1976] by applying the MaxSAT resolution rule on the set of Max2SAT clauses. This rule was first defined by [Larrosa and Heras, 2005; Larrosa *et al.*, 2008], and proven complete by [Bonet *et al.*, 2006; Bonet *et al.*, 2007]. In MaxSAT Resolution we proceed by replacing a set of clauses with another set of MaxSAT equivalent clauses:

$$
\frac{_{(w)}\, x \vee C_1 \quad _{(w)}\, \overline{x} \vee C_2}{\begin{array}{l} _{(w)}\, C_1 \vee C_2 \\ _{(w)}\, x \vee C_1 \vee \overline{C}_2 \\ _{(w)}\, \overline{x} \vee \overline{C}_1 \vee C_2 \end{array}}
\qquad
\frac{_{(w_1+w_2)}\, C}{\begin{array}{l} _{(w_1)}\, C \\ _{(w_2)}\, C \end{array}}
\qquad
\frac{_{(w_1)}\, C \quad _{(w_2)}\, C}{_{(w_1+w_2)}\, C}
$$

Applying the following transformations to the original [Garey *et al.*, 1976] gadget in Lemma 2, and removing the empty clause $_{(2)}\, \square$ that we obtain, we get the 3SAT to Max2SAT gadget described in Lemma 3:



Notice that in these gadget transformations we are allowed to remove empty clauses $_{(w)}\,\square$, and it has the effect of reducing the value of α in w (in the case of the [Garey *et al.*, 1976] gadget, from $\alpha = 7$ to $\alpha = 5$).

Lemma 3. *Given a 3SAT clause $x_1 \vee x_2 \vee x_3$, the set of Max2SAT clauses:*

$$\begin{array}{ll} {}_{(1)}\, x_1 \vee x_2 & {}_{(1)}\, b_1 \vee \overline{x}_1 \\ {}_{(1)}\, x_1 \vee x_3 & {}_{(1)}\, \overline{b}_1 \vee x_2 \\ {}_{(1)}\, \overline{x}_2 \vee \overline{x}_3 & {}_{(1)}\, \overline{b}_1 \vee x_3 \end{array}$$

Defines a $(5, 6)$-gadget from 3SAT to Max2SAT, where b_1 is an auxiliary variable.

Notice that the set of clauses described in Lemma 3 is unsatisfiable. Since MaxSAT resolution is complete, this means that we could derive an additional empty clause. However, in this refutation process, the rest of the clauses obtained are not binary. Therefore, the process of gadget transformation described in this section, based on MaxSAT resolution and empty-clause removal, allows us to simplify and reduce the α of a gadget, but it is not a complete method. In particular, we will see in the next sections that there are more efficient 3SAT to Max2SAT gadgets that cannot be obtained in this way.

5 Computing Gadgets Automatically

In [Trevisan *et al.*, 2000] a gadget from 3SAT to Max2SAT was computed automatically through a Mixed Integer Programming formulation.

Lemma 4 ([Trevisan *et al.*, 2000])**.** *Given a 3SAT clause $x_1 \vee x_2 \vee x_3$, the set of Max2SAT clauses:*

$$\begin{array}{ll} {}_{(1/2)}\, x_1 \vee x_3 & {}_{(1/2)}\, x_3 \vee \overline{b}_1 \\ {}_{(1/2)}\, \overline{x}_1 \vee \overline{x}_3 & {}_{(1/2)}\, \overline{x}_3 \vee b_1 \\ {}_{(1/2)}\, x_1 \vee \overline{b}_1 & {}_{(1)}\, x_2 \vee b_1 \\ {}_{(1/2)}\, \overline{x}_1 \vee b_1 & \end{array}$$

Defines a $(3.5, 4)$-gadget from 3SAT to Max2SAT optimal and strict, where b_1 is a new fresh auxiliary variable. Notice that all clauses have weight $1/2$, except the last one.

We reproduce to some extent the approach in [Trevisan *et al.*, 2000]. The main idea is to limit the search space of possible gadgets to a finite and reasonably sized one. Given the input kSAT clause $x_1 \vee \cdots \vee x_k$ and a maximum number of auxiliary variables b_j, we consider all the possible Max2SAT clauses we can build with the x_i and b_j vars. Then, we create a Mixed Integer Programming formulation that additionally includes a set of positive real variables representing the weight for each of the possible Max2SAT clauses. The idea is to let the MIP solver find an assignment to the weight variables that satisfies the definition of an (α, β)-gadget and minimizes the value of α. If the MIT solver finds weight 0 for a Max2SAT clause, then this clause is not considered in the gadget.

k	$\#b$	α	β	time
3	1	3.5	4	0.1
4	2	6	7	5
5	3	8.5	10	1200

Table 1: Gadgets for kSAT into Max2SAT automatically computed.

In Table 1, we present the results for input clauses of length k and number of auxiliary variables $\#b$. We applied on the MIP formulation the solver Gurobi v9.1. The machine has 2.1GHz and 5GB RAM. We show the α and β of the computed gadget and the time exhausted in less than 8 hours. From the results, we can conclude that we need to come up with a constructive generation method, as we do in the next Section, if we want to scale. However, these results are interesting since they prove that does exist a better gadget in terms of α with a less or equal number of auxiliary variables.

6 From kSAT to Max2SAT

A trivial way to reduce MaxSAT to Max2SAT is to reduce MaxSAT to Max3SAT, and then apply a gadget from Max3SAT into Max2SAT. We can reduce MaxSAT to Max3SAT with the following well-known gadget:

Lemma 5. *Given a kSAT clause $x_1 \vee \ldots \vee x_k$, the set of Max3SAT clauses:*

$$\begin{array}{l} {}_{(1)}\, x_1 \vee x_2 \vee b_1 \\ {}_{(1)}\, \overline{b}_1 \vee x_3 \vee b_2 \\ \cdots \\ {}_{(1)}\, \overline{b}_{k-4} \vee x_{k-2} \vee b_{k-3} \\ {}_{(1)}\, \overline{b}_{k-3} \vee x_{k-1} \vee x_k \end{array}$$

Defines a $(k-2, k-2)$-gadget from kSAT to Max3SAT, where $b_1, b_2, \ldots, b_{k-4}, b_{k-3}$ are fresh auxiliary variables.

By Lemma 1, the concatenation of the $(k-2, k-2)$-gadget from kSAT to Max3SAT (in Lemma 5) with any (α, β)-gadget from Max3SAT to Max2SAT results into a $(\alpha(k-2), \beta(k-2))$-gadget from kSAT to Max2SAT.

Table 2 shows the result of concatenating the $(k-2, k-2)$-gadget in Lemma 5 with the $(7, 10)$-gadget from [Garey *et al.*, 1976] (Lemma 2), with the refined $(5, 6)$-gadget in Lemma 3 and the $(3.5, 4)$-gadget from [Trevisan *et al.*, 2000] (Lemma 4), compared with the Regular gadget that we describe below.

Now, we introduce a new gadget from kSAT into Max2SAT. The gadget is inspired in the *regular* encoding [Ansótegui and Manyà, 2004] for cardinality constraints. It is not optimal, but easy to generalize for any size of clauses k, it only contains clauses with unit weight and, as we will

α	β	Source
$7\,(k-2)$	$10\,(k-2)$	[Garey *et al.*, 1976]
$5\,(k-2)$	$6\,(k-2)$	Lemma 3
$3.5\,(k-2)$	$4\,(k-2)$	[Trevisan *et al.*, 2000]
$2.5\,(k-2)+1$	$3\,(k-2)+1$	Refined Regular

Table 2: Some (α, β)-gadgets from kSAT to Max2SAT.

see later, it can be transformed into a more efficient gadget applying MaxSAT resolution.

Theorem 1 (Regular Gadget). *Given a kSAT clause $x_1 \vee \ldots \vee x_k$, the set of Max2SAT clauses, after replacing b_{k-1} by x_k:*

$$\begin{array}{ll} {}_{(1)}\, x_i & i = 1, \ldots, k \\ {}_{(1)}\, \overline{x}_i \vee \overline{b}_i & i = 1, \ldots, k-1 \\ {}_{(1)}\, \overline{x}_{i+1} \vee b_i & i = 1, \ldots, k-2 \\ {}_{(1)}\, b_i \vee \overline{b}_{i+1} & i = 1, \ldots, k-2 \end{array}$$

defines the regular $(3\,(k-2)+2, 4\,(k-2)+3)$-gadget from kSAT to Max2SAT, where $b_1, \ldots, b_{k-2}$ are fresh auxiliary variables.

Proof. When we replace x_i by false, for $i = 1, \ldots, k$, and simplify, we get k copies of the empty clause $\square$, plus $2k-2$ satisfied clauses, and the formula $\{b_1 \vee \overline{b}_2, \ldots, b_{k-3} \vee \overline{b}_{k-2}\}$ that is trivially satisfiable. Hence, the maximal number of satisfied clauses is $3\,(k-2)+1$.

When some of the x_i's are set to true, the number of cases to analyze is bigger. To make it simpler, consider the formula:

$$\varphi = \{x_i, \overline{x}_i \vee \overline{b}_i, \overline{x}_i \vee b_{i-1}, b_{i-1} \vee \overline{b}_i \mid i = 1, \ldots, k\}$$

For any assignment to the original variables, by setting the additional variables $b_0 = 1$, $b_k = 0$ and $b_{k-1} = x_k$, the number of unsatisfied clauses in φ is the same as in the original formula (simply, there are 4 more satisfied clauses). Let $S = \{i_1, \ldots, i_r\}$ the ordered list of indexes of variables x_i's set to true, where $r \geq 1$. When we replace the x_i's by their values in the formula, and simplify, we get $\varphi' = \{b_{i-1}, \overline{b}_i \mid i \in S\} \cup \{b_0 \vee \overline{b}_1, \ldots, b_{k-1} \vee \overline{b}_k\}$, plus $k - |S|$ empty clauses, plus some satisfied clauses. For every two consecutive indexes $i_j, i_{j+1} \in S$, we get a pairwise disjoint set of minimally unsatisfiable clauses $\varphi'_j = \{\overline{b}_{i_j}, b_{i_j} \vee \overline{b}_{i_j+1}, \ldots, b_{i_{j+1}-2} \vee \overline{b}_{i_{j+1}-1}, b_{i_{j+1}-1}\}$, and the rest of clauses of φ' not included in any of these subsets is satisfiable. Therefore, the minimal number of falsified clauses in φ' is $|S|-1$. Therefore, for any assignment I setting at least one x_i to true, the minimal number of falsified clauses in $I(\varphi)$ is $(k - |S|) + (|S| - 1) = k - 1$. The same holds for the original set of clauses. Hence the number of satisfied clauses is $\beta-(k-1) = (4(k-2)+3)-(k-1) = 3(k-2)+2 = \alpha$. $\square$

Now, we show how the process MaxSAT resolution and empty-clause elimination described in Section 4, applied to the clauses of this regular gadget (Theorem 1), results into the more efficient gadget described in Theorem 2.

We start transforming

$$\dfrac{{}_{(1)}\, x_1 \qquad {}_{(1)}\, \overline{x}_1 \vee \overline{b}_1}{{}_{(1)}\, \overline{b}_1}$$
$${}_{(1)}\, x_1 \vee b_1$$

Now, we iterate, for $i = 1, \ldots, k-2$, the following MaxSAT resolution (that results from applying 4 times the MaxSAT resolution rule):

$$\dfrac{\begin{array}{c} {}_{(1)}\, x_{i+1} \\ {}_{(1)}\, \overline{b}_i \\ {}_{(1/2)}\, \overline{x}_{i+1} \vee b_i \\ {}_{(1/2)}\, \overline{x}_{i+1} \vee \overline{b}_{i+1} \\ {}_{(1/2)}\, b_i \vee \overline{b}_{i+1} \end{array}}{\begin{array}{c} {}_{(1/2)}\, \square \\ {}_{(1)}\, \overline{b}_{i+1} \\ {}_{(1/2)}\, x_{i+1} \vee \overline{b}_i \\ {}_{(1/2)}\, x_{i+1} \vee b_{i+1} \\ {}_{(1/2)}\, \overline{b}_i \vee b_{i+1} \end{array}}$$

Since x_k and b_{k-1} denote the same variable, we transform

$$\dfrac{{}_{(1)}\, x_k \qquad {}_{(1)}\, \overline{b}_{k-1}}{{}_{(1)}\, \square}$$

Finally, we remove the clause ${}_{(k/2)}\, \square$.

Theorem 2 (Refined Regular Gadget). *Given a kSAT clause $x_1 \vee \ldots \vee x_k$, the set of Max2SAT clauses, after replacing b_{k-1} by x_k:*

$$\begin{array}{lll} {}_{(1)}\, x_1 \vee b_1 & & \\ {}_{(1/2)}\, x_{i+1} \vee \overline{b}_i, & {}_{(1/2)}\, \overline{x}_{i+1} \vee b_i & i = 1, \ldots, k-2 \\ {}_{(1/2)}\, x_{i+1} \vee b_{i+1}, & {}_{(1/2)}\, \overline{x}_{i+1} \vee \overline{b}_{i+1} & \\ {}_{(1/2)}\, b_i \vee \overline{b}_{i+1}, & {}_{(1/2)}\, \overline{b}_i \vee b_{i+1} & \end{array}$$

defines the (refined) regular $(2.5\,(k-2)+1, 3\,(k-2)+1)$-gadget from kSAT to Max2SAT, where $b_1, \ldots, b_{k-2}$ are fresh auxiliary variables.

Proof. The correctness of this gadget is a consequence of Theorem 1 and the fact that MaxSAT resolution and elimination of empty clauses preserve gadgets. The value of β can be easily computed adding the weights of all clauses. The value of alpha can be computed from the original gadget $\alpha' = 3(k-2)+2$ and removing the weight of the removed empty clauses $\alpha = \alpha' - k/2 = 2.5(k-2)+1$. $\square$

Notice that the refined regular gadget, for $k = 3$, is the same as the [Trevisan *et al.*, 2000] gadget of Lemma 4. Moreover, for $k = 3, 4$ and 5 the refined regular gadget coincides with the gadgets found automatically in Section 5.

7 Why Reducing SAT to Max2SAT?

Suppose that we were able to find a (α, β)-gadget from SAT to Max2SAT satisfying $\alpha = \beta$. It would allow us to translate any k-clause into a set of binary clauses that is satisfiable when the original clause is satisfied, and unsatisfiable, when

the original clause is unsatisfied. In other words, we were able to prove that P=NP. The gadget in Lemma 5 satisfies the property $\alpha = \beta$ reducing SAT to 3SAT, but (obviously) the same is not possible from 3SAT (or SAT) to 2SAT.

The following lemma shows us how we can reduce the problem of obtaining (by resolution or similar methods) an empty clause from a formula φ, to the problem of obtaining several empty clauses from a 2SAT formula φ' (by MaxSAT resolution or similar methods). In other words, how to solve SAT with a MaxSAT solver.

Lemma 6. *For any SAT formula φ, let φ' be the formula resulting from replacing every k-clause, with $k > 2$, by a set of binary clauses using a kSAT to Max2SAT (α_k, β_k)-gadget. Then φ is unsatisfiable if, and only if, the minimal number of unsatisfiable clauses in φ' is at least*

$$1 + \sum_{\substack{C \in \varphi \\ |C|>2}} (\beta_{|C|} - \alpha_{|C|})$$

In the next sections, we will show that this reduction of SAT to Max2SAT makes sense from a practical point of view. Here, we show that it also has some advantages from a theoretical perspective. In particular, we prove that the pigeonhole principle PHP^n_{n-1}, which requires exponentially long resolution proofs, when translated into Max2SAT, can be proved applying $\mathcal{O}(n^3)$ iterations of the MaxSAT resolution rule.

The idea that MaxSAT resolution can be more efficient than the classical resolution to prove some formula was first proposed by [Ignatiev *et al.*, 2017; Bonet *et al.*, 2018; Morgado *et al.*, 2019]. There, they prove that after translating the PHP^n_m principle using the dual-rail encoding, it has polynomial proofs using MaxSAT resolution. [Larrosa and Rollon, 2020] prove the same result for MaxSAT resolution extended with the split rule and using negative weights. [Atserias and Lauria, 2019] also prove it for circular resolution. Finally, [Bonet and Levy, 2020] prove the equivalence of these last two proof systems and the subsumption of weighted dual-rail. In this paper, we take a step forward to the efficient automatization of MaxSAT to solve SAT.

Lemma 7. *MaxSAT resolution obtains $_{(n-1)}\,\square$ from $\bigcup_{i=1}^{n}\{_{(1)}\, x_i\} \cup \bigcup_{\substack{i,j=1 \\ i\neq j}}^{n}\{_{(\infty)}\, \overline{x}_i \vee \overline{x}_j\}$ in $\frac{(n+2)(n-1)}{2}$ steps.*

Proof. First, we prove that we can construct the MaxSAT resolution proof:

$$\frac{\begin{array}{c} _{(1)}\, x_1 \vee \cdots \vee x_r \\ _{(1)}\, x_{r+1} \\ _{(\infty)}\, \overline{x}_i \vee \overline{x}_{r+1}, i = 1, \ldots, r \end{array}}{\begin{array}{c} _{(1)}\, \square \\ _{(1)}\, x_1 \vee \cdots \vee x_{r+1} \end{array}}$$

in $r+1$ steps: Cutting $_{(1)}\, x_1 \vee \cdots \vee x_r$ and $_{(\infty)}\, \overline{x}_1 \vee \overline{x}_{r+1}$ we get, among other clauses, $_{(1)}\, x_1 \vee \cdots \vee x_{r+1}$ and $_{(1)}\, x_2 \vee \cdots \vee x_r \vee \overline{x}_{r+1}$. Now, iteratively, we cut $_{(1)}\, x_j \vee \cdots \vee x_r \vee \overline{x}_{r+1}$ and $_{(\infty)}\, \overline{x}_j \vee \overline{x}_{r+1}$ to obtain, among other clauses, $_{(1)}\, x_{j+1} \vee \cdots x_r \vee \overline{x}_{r+1}$, for $j = 2, \ldots, r$. Finally we cut $_{(1)}\, \overline{x}_{r+1}$ and $_{(1)}\, x_{r+1}$ to obtain an empty clause.

To obtain the desired proof, we repeat the previous proof iteratively, for $r = 1, ..., n-1$, obtaining $_{(n-1)}\,\square$. The total number of rule applications is $\sum_{r=1}^{n-1} r+1 = \frac{(n+2)(n-1)}{2}$. □

Consider the Pigeon-Hole Problem PHP^n_m with n pigeons, m holes and $n > m$ where Boolean var $x_{i,j}$ set to true means pigeon i goes in hole j. We have two sets of clauses: a set of At-Least-One (ALO) clauses meaning that a pigeon must go on at least one hole, and a set of At-Most-One (AMO) clauses meaning that no two pigeons can go to the same hole. We can encode this problem into SAT as follows:

$$\bigcup_{i=1}^{n}\{x_{i,1} \vee \cdots \vee x_{i,m}\} \cup \bigcup_{j=1}^{m} \bigcup_{\substack{i,i'=1 \\ i\neq i'}}^{n} \{\overline{x}_{i,j} \vee \overline{x}_{i',j}\}$$

The translation of the ALO clauses using the (original) regular gadget results into the set of unary clauses $\bigcup_{i=1}^{n}\bigcup_{j=1}^{m}\{_{(1)}\, x_{i,j}\}$ plus some binary clauses. The set of AMO clauses does not need to be translated since they are already binary. Moreover, they can be assigned infinite weight since they must be satisfied by the MaxSAT solver. In order to conclude the unsatisfiability of the original PHP^n_m principle, the MaxSAT solver must obtain at least $1 + n(\beta_m - \alpha_m) = 1 + n(4(m-2)+3-(3(m-2)+2) = 1 + n(m-1)$ empty clauses.

Theorem 3. *The translation of PHP^n_m principle to Max2SAT using the original Regular gadget has MaxSAT refutation proofs of size $\mathcal{O}(m\, n^2)$.*

Proof. In the proof, we only need to consider the unary clauses and the binary infinite-weighted clauses (the other binary clauses are not needed in the proof). We can decompose the formula as $\varphi = \bigcup_{j=1}^{m} \varphi_j$, where $\varphi_j = \bigcup_{i=1}^{n}\{_{(1)}\, x_{i,j}\} \cup \bigcup_{\substack{i,i'=1 \\ i\neq i'}}^{n}\{\overline{x}_{i,j} \vee \overline{x}_{i',j}\}$. According to Lemma 7, from each φ_j we can obtain $_{(n-1)}\,\square$ in $\frac{(n+2)(n-1)}{2}$ MaxSAT refutation steps. For all the formulas, we will get $_{(m(n-1))}\,\square$ in $m\frac{(n+2)(n-1)}{2}$ steps. When $n > m$, the number of empty clauses obtained $m(n-1)$ is bigger than or equal to the number of required empty clauses $1+n(m-1)$, which proves the unsatisfiability of PHP^n_m. □

8 Solving SAT Through MaxSAT in Practice

In this section, we explore the potential of gadgets from SAT into Max2SAT in order to solve SAT formulas that are hard to Resolution. We evaluate the usefulness of gadgets on the Pigeon Hole Principle. For the experiments, we used a machine of 2.1GHz and 5GB RAM memory.

In table 3, we show the experiments we have conducted. In particular, we experimented with SAT solver CaDiCaL (version 1.0.3), winner of the SAT Race 2019 [Biere, 2019] and the top performing three MaxSAT solvers from the weighted category of the MaxSAT Evaluation 2020 [Bacchus *et al.*, 2019]: RC2 [Ignatiev *et al.*, 2019], MaxHS [Davies and Bacchus, 2011] and Uwrmaxsat [Piotrów, 2020].

The CaDiCaL SAT solver was run on the SAT instance (say φ) that encodes the Pigeon-Hole principle as described in

n	12	13	14	15	20	30	50	100
SAT								
CaDiCaL	17.6	91	500	3526	-	-	-	-
MaxHS	3236	-	-	-	-	-	-	-
UWr	988	-	-	-	-	-	-	-
Max2SAT [Garey *et al.*, 1976]								
RC2	0.1	0.1	0.13	0.23	0.74	4.48	43.75	1347
MaxHS	0.1	0.1	0.11	0.11	0.43	2.20	58.34	1174
UWr	0.1	0.1	0.13	0.19	0.68	3.97	35.66	1046
Max2SAT [Trevisan *et al.*, 2000]								
RC2	32	218	11602	-	-	-	-	-
MaxHS	0.1	0.14	0.2	5569	-	-	-	-
UWr	380	12835	4288	-	-	-	-	-
Max2SAT Regular								
RC2	0	0	0	0.1	0.1	0.22	0.93	10.5
MaxHS	0	0	0	0	0	0	0.15	1.35
UWr	0	0.1	0.4	0	0	0.1	0.16	1.39

Table 3: Run times for PHP^n_{n-1} encodings. '-' stands for timeout.

Algorithm 1: SAT through MaxSAT

```
1  Input: CNF φ, (α, β)-gadget G, MaxSAT solver MS
2  lb ← 0
3  φ ← ∅
4  while ⟨S, φ⟩ ← next_subset(φ) do
5      for C ∈ S do
6          if harden(C) then
7              φ ← φ ∪ {(∞) C}
8          else
9              φ ← φ ∪ G(C)
10             lb ← lb + β_|C| − α_|C|
11     opt ← MS(φ)
12     if opt > lb then
13         return Unsat
14 return Sat
```

Section 7. In particular, we created PHP^n_{n-1} SAT instances for $n \in \{11, 12, 13, 14, 15, 20, 30, 50, 100\}$.

The MaxSAT solvers were run on a MaxSAT formula built from the generated PHP^n_{n-1} SAT instance φ as follows: the 2SAT clauses in φ are assigned ∞ weight and the rest $(n-1)$-ary clauses are translated into MaxSAT using a gadget.

In the first 3 rows of table 3, we can see the result of applying the CaDiCaL solver and the MaxSAT solvers on the original SAT instances. For the MaxSAT solvers, we use as input the MaxSAT formula that incorporates with weight ∞ all the clauses of the SAT instance (RC2 was not able to accept this input). As we can see, the MaxSAT solvers are not as powerful as the CaDiCaL SAT solver. CaDiCaL can to solve the SAT instances, in less than 8 hours, up to $n = 15$.

In the next 6 rows, we show the behavior of the MaxSAT solvers on the concatenation of the gadget from Lemma 5 and the gadgets from [Garey *et al.*, 1976] and [Trevisan *et al.*, 2000], respectively. As we can see, MaxSAT solvers with the [Garey *et al.*, 1976] gadget work much better than the SAT approach with CaDiCaL, and they are able to solve $n = 100$ in around 20 minutes. On the other hand, with [Trevisan *et al.*, 2000] we see better results than with CaDiCaL but the impact is less obvious. There is also an erratic behaviour of MaxHS on $n = 14$ and 15, and of Uwrmaxsat on $n = 13$.

In the last 3 rows, we present the results with the Regular gadget. Clearly, it is the best performing one and allows all the MaxSAT solvers to solve $n = 100$ in less than 2 seconds. For the regular gadget and $n = 100$, we additionally shuffled the instance 100 times, obtaining the (median, max) results: Uwrmaxsat $(3.14, 3.49)$, MaxHS $(2.80, 3.19)$ which confirm the goodness of the gadget and, RC2 $(22.4, 383.25)$ that seems to be affected by the shuffling.

Finally, based on our findings we present Algorithm 1 that intends to be a conceptual base for a new generation of SAT solvers. It takes as input the SAT CNF instance to be solved, an (α, β)-gadget and a MaxSAT solver.

Algorithm 1 relies on two points. First, to find a good balance between which clauses we keep as hard and which are translated with the gadget. Second, to preserve as hard that subset of clauses that is not too hard to solve, for example, any polynomial class, like 2SAT, Horn, etc.

Second, when we apply the gadget to a subset of clauses, we immediately get a lower bound on the cost the MaxSAT solver will find for that particular set of clauses. That is the sum of $\beta_{|C|} - \alpha_{|C|}$, for each clause C we translate to Max2SAT through the gadget. Moreover, if for this subset of clauses the MaxSAT solver is able to find a greater cost than the mentioned lower bound then we can conclude that the original set of kSAT clauses is unsatisfiable. This way we make SAT solvers approaches able to count, being the absence of this capacity what precisely lies at the very heart of the drawbacks of Resolution.

In particular, Algorithm 1 iterates on subsets of the original input SAT instance (line 4) till all the formula has been processed. Then, for every clause in the current subset, it decides whether it has to be hard (function *harden* at line 6) or it has to be translated into a set of Max2SAT clauses through the input gadget. The resulting MaxSAT clauses are added to the working formula φ (lines 7 and 9). Whenever a clause is translated through the gadget the lower bound lb is increased according to $\beta_{|C|} - \alpha_{|C|}$ (line 10). If the MaxSAT solver executed on φ returns a cost greater than lb we can stop and declare the input SAT formula is unsatisfiable (line 13).

9 Conclusions

In this paper, we have contributed to filling the gap between SAT and MaxSAT towards the promise of MaxSAT to boost SAT. We have presented a constructive method for generating a new (α,β)-gadget from kSAT into Max2SAT. Although gadgets have been used in the literature to improve approximability and non-approximability results, here we have introduced a new application. In particular, we have shown how these gadgets can be applied to solve efficiently SAT formulas that are hard for resolution, which can constitute the base for a new generation of SAT solvers.

Acknowledgements

Supported by projects PROOFS (PID2019-109137GB-C21) and EU-H2020-RIP LOGISTAR (No. 769142).

References

[Ansótegui and Manyà, 2004] Carlos Ansótegui and Felip Manyà. Mapping problems with finite-domain variables into problems with boolean variables. In *SAT 2004 - The Seventh International Conference on Theory and Applications of Satisfiability Testing, 10-13 May 2004, Vancouver, BC, Canada, Online Proceedings*, 2004.

[Atserias and Lauria, 2019] Albert Atserias and Massimo Lauria. Circular (yet sound) proofs. In Mikolás Janota and Inês Lynce, editors, *Theory and Applications of Satisfiability Testing - SAT 2019 - 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9-12, 2019, Proceedings*, volume 11628 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2019.

[Bacchus *et al.*, 2019] Fahiem Bacchus, Matti Järvisalo, and Ruben Martins. MaxSAT Evaluation 2019 : Solver and Benchmark Descriptions. Technical Report Department of Computer Science Report Series B-2019-2, University of Helsinki, 2019.

[Biere, 2019] Armin Biere. CaDiCaL at the SAT Race 2019. In Marijn Heule, Matti Järvisalo, and Martin Suda, editors, *Proc. of SAT Race 2019 – Solver and Benchmark Descriptions*, volume B-2019-1 of *Department of Computer Science Series of Publications B*, pages 8–9. University of Helsinki, 2019.

[Bonet and Levy, 2020] Maria Luisa Bonet and Jordi Levy. Equivalence between systems stronger than resolution. In Luca Pulina and Martina Seidl, editors, *Theory and Applications of Satisfiability Testing - SAT 2020 - 23rd International Conference, Alghero, Italy, July 3-10, 2020, Proceedings*, volume 12178 of *Lecture Notes in Computer Science*, pages 166–181. Springer, 2020.

[Bonet *et al.*, 2006] Maria Luisa Bonet, Jordi Levy, and Felip Manyà. A complete calculus for max-sat. In Armin Biere and Carla P. Gomes, editors, *Theory and Applications of Satisfiability Testing - SAT 2006, 9th International Conference, Seattle, WA, USA, August 12-15, 2006, Proceedings*, volume 4121 of *Lecture Notes in Computer Science*, pages 240–251. Springer, 2006.

[Bonet *et al.*, 2007] Maria Luisa Bonet, Jordi Levy, and Felip Manyà. Resolution for Max-SAT. *Artif. Intell.*, 171(8-9):606–618, 2007.

[Bonet *et al.*, 2018] Maria Luisa Bonet, Sam Buss, Alexey Ignatiev, João Marques-Silva, and António Morgado. Maxsat resolution with the dual rail encoding. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *AAAI*, pages 6565–6572, 2018.

[Davies and Bacchus, 2011] Jessica Davies and Fahiem Bacchus. Solving MAXSAT by solving a sequence of simpler SAT instances. In Jimmy Ho-Man Lee, editor, *Principles and Practice of Constraint Programming - CP 2011 - 17th International Conference, CP 2011, Perugia, Italy, September 12-16, 2011. Proceedings*, volume 6876 of *Lecture Notes in Computer Science*, pages 225–239. Springer, 2011.

[Feige and Goemans, 1995] U. Feige and M. Goemans. Approximating the value of two power proof systems, with applications to MAX 2SAT and MAX DICUT. In *Proceedings Third Israel Symposium on the Theory of Computing and Systems*, pages 182–189, 1995.

[Garey *et al.*, 1976] M. R. Garey, David S. Johnson, and Larry J. Stockmeyer. Some simplified np-complete graph problems. *Theor. Comput. Sci.*, 1(3):237–267, 1976.

[Ignatiev *et al.*, 2017] Alexey Ignatiev, António Morgado, and João Marques-Silva. On tackling the limits of resolution in SAT solving. In *Proc. of the 20th Int. Conf. on Theory and Applications of Satisfiability Testing, SAT'17*, pages 164–183, 2017.

[Ignatiev *et al.*, 2019] Alexey Ignatiev, António Morgado, and João Marques-Silva. RC2: an efficient maxsat solver. *J. Satisf. Boolean Model. Comput.*, 11(1):53–64, 2019.

[Larrosa and Heras, 2005] Javier Larrosa and Federico Heras. Resolution in max-sat and its relation to local consistency in weighted csps. In *IJCAI*, pages 193–198, 2005.

[Larrosa and Rollon, 2020] Javier Larrosa and Emma Rollon. Augmenting the power of (partial) maxsat resolution with extension. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 1561–1568. AAAI Press, 2020.

[Larrosa *et al.*, 2008] Javier Larrosa, Federico Heras, and Simon de Givry. A logical approach to efficient max-sat solving. *Artif. Intell.*, 172(2-3):204–233, 2008.

[Morgado *et al.*, 2019] António Morgado, Alexey Ignatiev, Maria Luisa Bonet, João Marques-Silva, and Sam Buss. Drmaxsat with maxhs: First contact. In Mikolás Janota and Inês Lynce, editors, *Theory and Applications of Satisfiability Testing - SAT 2019 - 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9-12, 2019, Proceedings*, volume 11628 of *Lecture Notes in Computer Science*, pages 239–249. Springer, 2019.

[Piotrów, 2020] Marek Piotrów. Uwrmaxsat: Efficient solver for maxsat and pseudo-boolean problems. In *32nd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2020, Baltimore, MD, USA, November 9-11, 2020*, pages 132–136. IEEE, 2020.

[Trevisan *et al.*, 2000] Luca Trevisan, Gregory B. Sorkin, Madhu Sudan, and David P. Williamson. Gadgets, approximation, and linear programming. *SIAM J. Comput.*, 29(6):2074–2097, 2000.