On Tackling the Limits of Resolution in SAT Solving *

Alexey Ignatiev[1,2], Antonio Morgado[1], and Joao Marques-Silva[1]

[1] LASIGE, Faculty of Science, University of Lisbon, Portugal
{aignatiev,ajmorgado,jpms}@ciencias.ulisboa.pt
[2] ISDCT SB RAS, Irkutsk, Russia

Abstract. The practical success of Boolean Satisfiability (SAT) solvers stems from the CDCL (Conflict-Driven Clause Learning) approach to SAT solving. However, from a propositional proof complexity perspective, CDCL is no more powerful than the resolution proof system, for which many hard examples exist. This paper proposes a new problem transformation, which enables reducing the decision problem for formulas in conjunctive normal form (CNF) to the problem of solving maximum satisfiability over Horn formulas. Given the new transformation, the paper proves a polynomial bound on the number of MaxSAT resolution steps for pigeonhole formulas. This result is in clear contrast with earlier results on the length of proofs of MaxSAT resolution for pigeonhole formulas. The paper also establishes the same polynomial bound in the case of modern core-guided MaxSAT solvers. Experimental results, obtained on CNF formulas known to be hard for CDCL SAT solvers, show that these can be efficiently solved with modern MaxSAT solvers.

1 Introduction

Boolean Satisfiability (SAT) solvers have made remarkable progress over the last two decades. Unable to solve formulas with more than a few hundred variables in the early 90s, SAT solvers are now capable of routinely solving formulas with a few million variables [13,17]. The success of SAT solvers is supported by the CDCL (Conflict-Driven Clause Learning) [17, Chapter 04] paradigm, and the ability of SAT solvers to learn clauses from induced conflicts [17]. Nevertheless, being no more powerful than the general resolution proof system [59], CDCL SAT solvers are also known not to scale for specific formulas, which are hard for resolution [23, 39, 68]. Recent work has considered different forms of extending CDCL with techniques adapted from more powerful proof systems as well as others [7, 12, 16, 22, 40, 67], with success in some settings. Nevertheless, for pigeonhole formulas [27], and with the exception of the lingeling SAT solver [14] on specific encodings, modern CDCL SAT solvers are unable to prove unsatisfiability even for a fairly small numbers of pigeons.

This paper proposes an alternative path to tackle the difficulties of the resolution proof system, by developing an approach that aims to complement existing SAT solvers, and which also builds upon efficient CDCL SAT solvers. The motivation is to transform the original problem, from one clausal form to another, the latter enconding a restricted maximum satisfiability problem, but in such a way that CDCL SAT solvers can still be exploited.

* This work was supported by FCT funding of post-doctoral grants SFRH/BPD/103609/2014, SFRH/BPD/120315/2016, and LASIGE Research Unit, ref. UID/CEC/00408/2013.

Given any CNF formula $\mathcal{F}$, the paper shows how to encode the problem as Horn Maximum Satisfiability (HornMaxSAT), more concretely by requiring a given cost on the HornMaxSAT formulation. This enables solving the modified problem with either a MaxSAT solver or with a dedicated HornMaxSAT solver. The paper then shows that for propositional encodings of the pigeonhole principle [27], transformed to HornMaxSAT, there exists a polynomially time bounded sequence of MaxSAT resolution steps which enables deriving a number of falsified clauses that suffices for proving unsatisfiable the original PHP formula. Similarly, the paper also proves that for modern core-guided MaxSAT solvers there exist sequences of unsatisfiable cores that enable reaching the same conclusion in polynomial time. This in turn suggests that MaxSAT algorithms [54] can be effective in practice when applied to such instances.

Experimental results, obtained on different encodings of the pigeonhole principle, but also on other instances that are well-known to be hard for resolution [68], confirm the theoretical result. Furthermore, a recently-proposed family of MaxSAT solvers [29, 63], based on iterative computation of minimum hitting sets, is also shown to be effective in practice and on a wider range of classes of instances.

The paper is organized as follows. Section 2 introduces the definitions and notation used throughout the paper. Section 3 develops a simple encoding from SAT into HornMaxSAT. Section 4 derives a polynomial bound on the number and size of MaxSAT-resolution steps to establish the unsatisfiability of propositional formulas encoding the pigeonhole principle transformed into HornMaxSAT. The section also shows that there are executions of core-guided MaxSAT solvers that take polynomial time to establish a lower bound of the cost of the MaxSAT solution which establishes the unsatisfiability of the original CNF formula. Experimental results on formulas encoding the pigeonhole principle, but also on other formulas known to be hard for CDCL SAT solvers [68] are analyzed in Section 5.2. The paper concludes in Section 6.

2 Preliminaries

The paper assumes definitions and notation standard in propositional satisfiability (SAT) and maximum satisfiability (MaxSAT) [17]. Propositional variables are taken from a set $X = \{x_1, x_2, \ldots\}$. A Conjunctive Normal Form (CNF) formula is defined as a conjunction of disjunctions of literals, where a literal is a variable or its complement. CNF formulas can also be viewed as sets of sets of literals, and are represented with calligraphic letters, $\mathcal{A}$, $\mathcal{F}$, $\mathcal{H}$, etc. A truth assignment is a map from variables to $\{0, 1\}$. Given a truth assignment, a clause is satisfied if at least one of its literals is assigned value 1; otherwise it is falsified. A formula is satisfied if all of its clauses are satisfied; otherwise it is falsified. If there exists no assignment that satisfies a CNF formula $\mathcal{F}$, then $\mathcal{F}$ is referred to as *unsatisfiable*. (Boolean) Satisfiability (SAT) is the decision problem for propositional formulas, i.e. to decide whether a given propositional formula is satisfiable. Since the paper only considers propositional formulas in CNF, throughout the paper SAT refers to the decision problem for propositional formulas in CNF.

To simplify modeling with propositional logic, one often represents more expressive constraints. Concrete examples are cardinality constraints and pseudo-Boolean constraints [17]. A cardinality constraint of the form $\sum x_i \leq k$ is referred to as an $\mathsf{AtMost}k$ constraint, whereas a cardinality constraint of the form $\sum x_i \geq k$ is referred to as an

$\mathsf{AtLeast}k$ constraint. The study of propositional encodings of cardinality and pseudo-Boolean constraints is an area of active research [1, 4, 5, 9, 10, 17, 25, 34, 58, 65, 69].

A clause is Horn if it contains at most one positive literal. A Horn clause is a *goal* clause if it has no positive literals; otherwise it is a *definite* clause. The decision problem for Horn formulas is well-known to be in P, with linear time algorithms since the 80s [32, 52]. A number of function problems defined on Horn formulas can be solved in polynomial time [48]. These include computing the lean kernel, finding a minimal unsatisfiable subformula and finding a maximal satisfiable subformula.

2.1 Propositional Encodings of the Pigeonhole Principle

The propositional encoding of the pigeonhole hole principle is well-known [27].

Definition 1 (Pigeonhole Principle, PHP [27])**.** *The pigeonhole principle states that if* $m+1$ *pigeons are distributed by* m *holes, then at least one hole contains more than one pigeon. A more formal formulation is that there exists no injective function mapping from* $\{1, 2, ..., m+1\}$ *to* $\{1, 2, ..., m\}$*, for* $m \geq 1$.

Propositional formulations of PHP encode the negation of the principle, and ask for an assignment such that the $m+1$ pigeons are placed into m holes. The propositional encoding of the PHP_m^{m+1} problem can be derived as follows. Let the variables be x_{ij}, with $1 \leq i \leq m+1, 1 \leq j \leq m$, with $x_{ij} = 1$ iff the i^{th} pigeon is placed in the j^{th} hole. The constraints are that each pigeon must be placed in at least one hole, and each hole must not have more than one pigeon.

$$\bigwedge_{i=1}^{m+1} \mathsf{AtLeast1}(x_{i1}, \ldots, x_{im}) \wedge \bigwedge_{j=1}^{m} \mathsf{AtMost1}(x_{1j}, \ldots, x_{m+1,j}) \tag{1}$$

An $\mathsf{AtLeast1}$ constraint can be encoded with a single clause. For the $\mathsf{AtMost1}$ constraint there are different encodings, including [17, 34, 65]. For example, the pairwise encoding [17] of $\mathsf{AtMost1}(x_{1j}, \ldots, x_{m+1,j})$ uses no auxiliary variables and the clauses $\wedge_{r=2}^{m+1} \wedge_{s=1}^{r-1} (\neg x_{rj} \vee \neg x_{sj})$. It is well-known that resolution has an exponential lower bound for PHP [11, 39, 61].

2.2 MaxSAT, MaxSAT Resolution & MaxSAT Algorithms

MaxSAT. For unsatisfiable formulas, the maximum satisfiability (MaxSAT) problem is to find an assignment that maximizes the number of satisfied clauses (given that not all clauses can be satisfied). There are different variants of the MaxSAT problem [17, Chapter 19]. Partial MaxSAT allows for *hard* clauses (which must be satisfied) and *soft* clauses (which represent a preference to satisfy those clauses). There are also weighted variants, in which soft clauses are given a weight, and for which hard clauses (if any) have a weight of $\top$ (meaning clauses that must be satisfied). The notation (c, w) will be used to represent a clause c with w denoting the cost of falsifying c. The paper considers partial MaxSAT instances, with hard clauses, for which $w = \top$, and soft clauses, for which $w = 1$. The notation $\langle \mathcal{H}, \mathcal{S} \rangle$ is used to denote partial MaxSAT problems with sets of hard ($\mathcal{H}$) and soft ($\mathcal{S}$) clauses. Throughout the paper, a MaxSAT *solution* represents either a maximum cardinality set of satisfied soft clauses or an assignment that satisfies all hard clauses and also maximizes (or minimizes, resp.) the number of satisfied (or falsified, resp.) soft clauses.

Table 1: Example MaxSAT-resolution steps

Clause 1	Clause 2	Derived Clauses
$(x \vee A, u)$	$(\neg x \vee B, w)$	$(A \vee B, m)$, $(x \vee A, u \ominus m)$, $(\neg x \vee B, w \ominus m)$, $(x \vee A \vee \neg B, m)$, $(\neg x \vee \neg A \vee B, m)$
$(x \vee A, 1)$	$(\neg x, \top)$	$(A, 1)$, $(\neg x, \top)$, $(\neg x \vee \neg A, 1)$

MaxSAT Resolution [18,46]. In contrast with SAT, the MaxSAT resolution operation requires the introduction of additional clauses other than the resolvent, and resolved clauses cannot be resolved again. Let $(x \vee A, u)$ and $(\neg x \vee B, w)$ be two clauses, and let $m \triangleq \min(u, w)$, $u \ominus w \triangleq (u == \top)\,?\,\top\,:\,u - w$, with $u \geq w$. The (non-clausal) MaxSAT resolution step [46] is shown in Table 1. (We could have used the clausal formulation [18], but it is more verbose and unnecessary for the purposes of the paper. It suffices to mention that clausal MaxSAT resolution adds at most $2n$ clauses at each resolution step, where the number of variables is n and the number of literals in each clause does not exceed n.) It is well-known that MaxSAT-resolution is unlikely to improve propositional resolution [18]. For the original PHP_m^{m+1} formulas, there are known exponential lower bounds on the size of deriving one empty clause by MaxSAT-resolution (given that the remaining clauses are satisfiable) [18, Corollary 18].

MaxSAT Algorithms. Many algorithms for MaxSAT have been proposed over the years [17, Chapter 19]. The most widely investigated can be broadly organized into branch and bound [17, Chapter 19], iterative-search [12,36,45], core-guided [2,36,49, 50,53,54,56], and minimum hitting sets [29,63]. In most proposed algorithms, core-guided and minimum hitting sets MaxSAT algorithms iteratively determine formulas to be unsatisfiable, until satisfiability is reached for a formula that relaxes clauses of minimum cost. This paper analyzes the operation of core-guided MaxSAT algorithms, concretely the MSU3 algorithm [49] [1]. Moreover, and to our best knowledge, the relationship between core-guided MaxSAT algorithms and MaxSAT resolution was first investigated in [56].

2.3 Related Work

The complexity of resolution on pigeonhole formulas has been studied by different authors, e.g. see [11,27,39,57,61] and references therein, among others. It is well-known that for other proof systems, including cutting planes and extended resolution, PHP has polynomial proofs [6,20,21,26,28,64]. Different authors have looked into extending CDCL (and so resolution) with the goal of solving formulas for which resolution has known exponential lower bounds [7,12,13,16,37,38,40,43,44,67]. Some SAT solvers apply pattern matching techniques [14], but these are only effective for specific propositional encodings. Furthermore, there has been limited success in applying cutting planes and extended resolution in practical SAT solvers.

[1] Different implementations of the MSU3 have been proposed over the years [2,49,50,54], which often integrate different improvements. A well-known implementation of MSU3 is OpenWBO [50], one of the best MaxSAT solvers in the MaxSAT Evaluations since 2014.

3 Reducing SAT to HornMaxSAT

The propositional satisfiability problem for CNF formulas can be reduced to HornMaxSAT, more concretely to the problem of deciding whether for some target Horn formula there exists an assignment that satisfies a given number of soft clauses.

Let $\mathcal{F}$ be a CNF formula, with N variables $\{x_1 \ldots, x_N\}$ and M clauses $\{c_1, \ldots, c_M\}$. Given $\mathcal{F}$, the reduction creates a Horn MaxSAT problem with hard clauses $\mathcal{H}$ and soft clauses $\mathcal{S}$, $\langle \mathcal{H}, \mathcal{S} \rangle = \mathsf{HEnc}(\mathcal{F})$. For each variable $x_i \in X$, create new variables p_i and n_i, where $p_i = 1$ iff $x_i = 1$, and $n_i = 1$ iff $x_i = 0$. Thus, we need a hard clause $(\neg p_i \vee \neg n_i)$, to ensure that we do not simultaneously assign $x_i = 1$ and $x_i = 0$. (Observe that the added clause is Horn.) This set of hard Horn clauses is referred to as $\mathcal{P}$. For each clause c_j, we require c_j to be satisfied, by requiring that one of its literals *not* to be falsified. For each literal x_i use $\neg n_i$, and for each literal $\neg x_i$ use $\neg p_i$. Thus, c_j is encoded with a new (hard) clause c'_j with the same number of literals as c_j, but with only negative literals on the p_i and n_i variables, and so the resulting clause is also Horn. The set of soft clauses $\mathcal{S}$ is given by (p_i) and (n_i) for each of the original variables x_i. If the resulting Horn formula has a HornMaxSAT solution with at least N variables assigned value 1, then the original formula is satisfiable; otherwise the original formula is unsatisfiable. (Observe that, by construction, the HornMaxSAT solution cannot assign value 1 to more than N variables. Thus, unsatisfiability implies being unable to satisfy more than $N - 1$ soft clauses.) Clearly, the encoding outlined in this section can be the subject of different improvements.

Example 1. Let the CNF formula be:

$$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \tag{2}$$

The new variables are $\{n_1, p_1, n_2, p_2, n_3, p_3\}$. Preventing simultaneous assignment to 0 and 1 is guaranteed with the hard clauses:

$$(\neg n_1 \vee \neg p_1) \wedge (\neg n_2 \vee \neg p_2) \wedge (\neg n_3 \vee \neg p_3) \tag{3}$$

The original clauses are reencoded as hard clauses as follows:

$$(\neg n_1 \vee \neg p_2 \vee \neg n_3) \wedge (\neg n_2 \vee \neg n_3) \wedge (\neg p_1 \vee \neg p_3) \tag{4}$$

Finally, the soft clauses are $\mathcal{S} = \{(n_1), (p_1), (n_2), (p_2), (n_3), (p_3)\}$.

The transformation proposed above can be related with the well-known dual-rail encoding, used in different settings [19, 42, 47, 60, 62]. To our best knowledge, the use of a dual-rail encoding for deriving a pure Horn formula has not been proposed in earlier work.

Lemma 1. *Given $\langle \mathcal{H}, \mathcal{S} \rangle = \mathsf{HEnc}(\mathcal{F})$, there can be no more than N satisfied soft clauses.*

Proof. By construction of $\langle \mathcal{H}, \mathcal{S} \rangle$, for any x_i, there is no assignment that satisfies $\mathcal{H}$ with $n_i = 1$ and $p_i = 1$. □

Lemma 2. *Let $\mathcal{F}$ have a satisfying assignment ν. Then, there exists an assignment that satisfies $\mathcal{H}$ and N soft clauses in $\langle \mathcal{H}, \mathcal{S} \rangle = \mathsf{HEnc}(\mathcal{F})$.*

Proof. Given ν, we create an assignment ν' to the n_i and p_i variables that satisfies the clauses in $\mathcal{H}$, and N clauses in $\mathcal{S}$. For each x_i, if $\nu(x_i) = 1$, then set $p_i = 1$; otherwise set $n_i = 1$. Thus, there will be N satisfied clauses in $\mathcal{S}$. For each clause $c_j \in \mathcal{F}$, pick a literal l_k assigned value 1. If $l_k = x_k$, then c'_k contains literal $\neg n_k$, and so it is satisfied. If $l_k = \neg x_k$, then c'_k contains literal $\neg p_k$, and so it is satisfied. Thus every clause in $\mathcal{H}$ is satisfied, and N soft clauses are satisfied. □

Lemma 3. *Let ν' be an assignment that satisfies the clauses in $\mathcal{H}$ and N clauses in $\mathcal{S}$. Then there exists an assignment ν that satisfies $\mathcal{F}$.*

Proof. By construction of $\langle \mathcal{H}, \mathcal{S} \rangle$, for each x_i, either n_i is assigned value 1, or p_i is assigned value 1, but not both. Let $\nu(x_i) = 1$ if $\nu'(p_i) = 1$ and $\nu(x_i) = 0$ if $\nu'(n_i) = 1$. All variables x_i are either assigned value 0 or 1. For clause c'_j, let l_k be a literal assigned value 1. If $l_k = \neg n_k$, then x_k is a literal in c_j and since $\nu(x_i) = 1$, then the clause c_j is satisfied. Otherwise, if $l_k = \neg p_k$, then $\neg x_k$ is a literal in c_j and since $\nu(x_i) = 0$, then the clause c_j is satisfied. □

Lemma 1, Lemma 2 and Lemma 3 yield the following.

Theorem 1. *$\mathcal{F}$ is satisfiable if and only if there exists an assignment that satisfies $\mathcal{H}$ and N clauses in $\mathcal{S}$.*

The reduction of SAT into HornMaxSAT can also be applied to the PHP_m^{m+1} problem.

Example 2 (Pigeonhole Principle). With each variable x_{ij}, $1 \le i \le m+1, 1 \le j \le m$, we associate two new variables: n_{ij} and p_{ij}. The set of clauses $\mathcal{P}$ prevents a variable x_i from being assigned value 0 and 1 simultaneously: $\mathcal{P} = \{(\neg n_{ij} \vee \neg p_{ij}) \,|\, 1 \le i \le m+1, 1 \le j \le m\}$. $\mathcal{L}_i$ represents the encoding of each $\mathsf{AtLeast1}$ constraint, concretely $\mathcal{L}_i = (\neg n_{i1} \vee \ldots \vee \neg n_{im})$. $\mathcal{M}_j$ represents the encoding of each $\mathsf{AtMost1}$ constraint, which will depend on the encoding used. The soft clauses $\mathcal{S}$ are given by,

$$\begin{gathered}\{(n_{11}), \ldots, (n_{1m}), \ldots, (n_{m+1\,1}), \ldots, (n_{m+1\,m}),\\ (p_{11}), \ldots, (p_{1m}), \ldots, (p_{m+1\,1}), \ldots, (p_{m+1\,m})\}\end{gathered}$$

with $|\mathcal{S}| = 2m(m+1)$. Thus, the complete reduction of PHP into MaxSAT becomes:

$$\mathsf{HEnc}\left(\text{PHP}_m^{m+1}\right) \triangleq \langle \mathcal{H}, \mathcal{S} \rangle = \left\langle \wedge_{i=1}^{m+1} \mathcal{L}_i \wedge \wedge_{j=1}^{m} \mathcal{M}_j \wedge \mathcal{P}, \mathcal{S} \right\rangle \tag{5}$$

Clearly, given $\mathcal{P}$, one cannot satisfy more the $m(m+1)$ soft clauses. By Theorem 1, PHP_m^{m+1} is satisfiable if and only if there exists an assignment that satisfies the hard clauses $\mathcal{H}$ and $m(m+1)$ soft clauses from $\mathcal{S}$.

4 Short MaxSAT Proofs for PHP

This section shows that the reduction of PHP_m^{m+1} to HornMaxSAT based on a dual-rail encoding enables both existing core-guided MaxSAT algorithms and also MaxSAT resolution, to prove in polynomial time that the original problem formulation[2] is unsatisfiable. Recall from Theorem 1, that PHP_m^{m+1} is satisfiable if and only if, given (5), there exists an assignment that satisfies $\mathcal{H}$ and $m(m+1)$ soft clauses in $\mathcal{S}$. This section shows that for both core-guided algorithms and for MaxSAT resolution, we can con-

[2] This section studies the original *pairwise* encoding of PHP_m^{m+1}. However, a similar argument can be applied to PHP_m^{m+1} provided any encoding of AtMost1 constraints $\mathcal{M}_j$, as confirmed by the experimental results in Section 5.2.

Table 2: Partitioned core-guided unit propagation steps

Constraint	Hard clause(s)	Soft clause(s)	Relaxed clauses	Updated AtMost$k$ Constraints	LB increase
$\mathcal{L}_i$	$(\neg n_{i1} \vee \ldots \vee \neg n_{im})$	$(n_{i1}), \ldots, (n_{im})$	$(r_{il} \vee n_{i1})$, $1 \leq l \leq m$	$\sum_{l=1}^{m} r_{il} \leq 1$	1
$\mathcal{M}_j$	$(\neg p_{1j} \vee \neg p_{2j})$	$(p_{1j}), (p_{2j})$	$(r_{1j} \vee p_{1j})$, $(r_{2j} \vee p_{2j})$	$\sum_{l=1}^{2} r_{lj} \leq 1$	1
$\mathcal{M}_j$	$(\neg p_{1j} \vee \neg p_{3j})$, $(\neg p_{2j} \vee \neg p_{3j})$, $(r_{1j} \vee p_{1j})$, $(r_{2j} \vee p_{2j})$, $\sum_{l=1}^{2} r_{lj} \leq 1$	(p_{3j})	$(r_{3j} \vee p_{3j})$	$\sum_{l=1}^{3} r_{lj} \leq 2$	1
		$\ldots$			
$\mathcal{M}_j$	$(\neg p_{1j} \vee \neg p_{m+1j}), \ldots,$ $(\neg p_{mj} \vee \neg p_{m+1j})$, $(r_{1j} \vee p_{1j}), \ldots,$ $(r_{mj} \vee p_{mj})$, $\sum_{l=1}^{m} r_{lj} \leq m-1$	(p_{m+1j})	$(r_{m+1j} \vee p_{m+1j})$	$\sum_{l=1}^{m+1} r_{lj} \leq m$	1

clude in polynomial time that satisfying $\mathcal{H}$ requires falsifying at least $m(m+1)+1$ soft clauses, thus proving PHP_m^{m+1} to be unsatisfiable.

The results in this section should be contrasted with earlier work [18], which proves that MaxSAT resolution requires an exponentially large proof to produce an empty clause, this assuming the *original* propositional encoding for PHP_m^{m+1}.

4.1 A Polynomial Bound on Core-Guided MaxSAT Algorithms

This section shows that a core-guided MaxSAT algorithm will conclude in polynomial time that more than $m(m+1)$ clauses must be falsified, when the hard clauses are satisfied, thus proving the original PHP_m^{m+1} to be unsatisfiable. The analysis assumes the operation of basic core-guided algorithm, MSU3 [49], but similar analyses could be carried out for other families of core-guided algorithms[3].

The following observations about (5) are essential to prove the bound on the run time. First, the clauses in the $\mathcal{L}_i$ constraints do not share variables in common with the clauses in the $\mathcal{M}_j$ constraints. Second, each constraint $\mathcal{L}_i$ is of the form $(\neg n_{i1} \vee \ldots \vee \neg n_{im})$ and so its variables are disjoint from any other $\mathcal{L}_k$, $k \neq i$. Third, assuming a pairwise encoding, each constraint $\mathcal{M}_j$ is of the form $\wedge_{r=2}^{m+1} \wedge_{s=1}^{r-1} (\neg p_{rj} \vee \neg p_{sj})$, and so its variables are disjoint from any other $\mathcal{M}_l$, $l \neq j$. Since the sets of variables for each constraint are disjoint from the other sets of variables, we can exploit this partition of the clauses, and run a MaxSAT solver *separately* on each one. (Alternatively, we could assume the MSU3 MaxSAT algorithm to work with disjoint unsatisfiable cores.)

Table 2 summarizes the sequence of unit propagation steps that yields a lower bound on the number of falsified clauses larger than $m(m+1)$. For each $\mathcal{L}_i$, the operation is summarized in the second row of Table 2. Unit propagation yields a conflict between m soft clauses and the corresponding hard clause. This means that at least one of these

[3] Basic knowledge of core-guided MaxSAT algorithms is assumed. The reader is referred to recent surveys for more information [2,54].

Table 3: Analysis of $\mathcal{M}_j$, iteration k

Clauses	Unit Propagation
$(p_{k+1\,j})$	$p_{k+1\,j} = 1$
$(\neg p_{1j} \vee \neg p_{k+1\,j}), \ldots, (\neg p_{kj} \vee \neg p_{k+1\,j})$	$p_{1j} = \ldots = p_{kj} = 0$
$(r_{1j} \vee p_{1j}), \ldots, (r_{kj} \vee p_{kj})$	$r_{1j} = \ldots = r_{kj} = 1$
$\sum_{l=1}^{k} r_{lj} \leq k-1$	$\left(\sum_{l=1}^{k} r_{lj} \leq k-1\right) \vdash_1 \bot$

soft clauses must be falsified. Since there are $m+1$ constraints $\mathcal{L}_i$, defined on disjoint sets of variables, then each will contribute at least one falsified soft clause, which puts the lower bound on the number of falsified clauses at $m+1$.

For each $\mathcal{M}_j$ the operation is summarized in rows 3 to last of Table 2. Each row indicates a sequence of unit propagation steps that produces a conflict, each on a distinct set of soft clauses. Observe that each soft clause (p_{kj}), $k \geq 2$, induces a sequence of unit propagation steps, that causes the AtMost$\{k-1\}$ constraint to become inconsistent. Concretely, for iteration k (where row 3 corresponds to iteration 1), the sequence of unit propagation steps is summarized in Table 3 [4]. Since there are m such rows, then each $\mathcal{M}_j$ contributes at least m falsified soft clauses. Moreover, the number of $\mathcal{M}_j$ constraints is m, and so the $\mathcal{M}_j$ constraints increase the bound by $m \cdot m$.

Given the above, in total we are guaranteed to falsify at least $m+1+m \cdot m = m(m+1)+1$ clauses, thus proving that one cannot satisfy $m(m+1)$ soft clauses if the hard clauses are satisfied. In turn, this proves that the PHP_m^{m+1} problem is unsatisfiable.

We can also measure the run time of the sequence of unit propagation steps. For each $\mathcal{L}_i$, the run time is $\mathcal{O}(m)$, and there will be m such unit propagation steps, for a total $\mathcal{O}(m^2)$. For each $\mathcal{M}_j$ there will be m unit propagation steps, with run time between $\mathcal{O}(1)$ and $\mathcal{O}(m)$. Thus, the run time of the sequence of unit propagation steps for each $\mathcal{M}_j$ is $\mathcal{O}(m^2)$. Since there are m constraints $\mathcal{M}_j$, then the total run time is $\mathcal{O}(m^3)$.

Proposition 1. *Given* (5)*, and for a core-guided MSU3-like MaxSAT solver, there is a sequence of unit propagation steps such that a lower bound of* $m(m+1)+1$ *is computed in* $\mathcal{O}(m^3)$ *time.*

Proof. (Sketch) The discussion above. □

Moreover, it is important to observe that the unit propagation steps considered in the analysis above avoid the clauses in $\mathcal{P}$, i.e. only the clauses in $\mathcal{L}_i$, $\mathcal{M}_j$, $\mathcal{S}$, and relaxed clauses, are used for deriving the lower bound of $m(m+1)+1$ on the minimum number of falsified soft clauses. As shown in Section 5.2, and for the concrete case of PHP, the clauses in $\mathcal{P}$ are unnecessary and actually impact negatively the performance of core-guided MaxSAT solvers. Finally, and although the proof above assumes an MSU3-like core-guided algorithm, similar ideas could be considered in the case of other variants of core-guided MaxSAT algorithms [2, 36, 54, 56].

[4] The notation $\Phi \vdash_1 \bot$ indicates that inconsistency (i.e. a falsified clause) is derived by unit propagation on the propositional encoding of Φ. This is the case with existing encodings of AtMost$k$ constraints.

4.2 A Polynomial Bound on MaxSAT Resolution

We can now exploit the intuition from the previous section to identify the sequence of MaxSAT resolution steps that enable deriving $m(m+1)+1$ empty clauses, thereby proving that *any* assignment that satisfies the hard clauses must falsify at least $m(m+1)+1$ soft clauses, and therefore proving that the propositional encoding of PHP is unsatisfiable. As before, we assume that the pairwise encoding is used to encode each constraint $\mathcal{M}_j$. As indicated earlier in Section 2.2, we consider a simplified version of MaxSAT resolution [46], which is non-clausal. As explained below, this is not problematic, as just a few clauses are of interest. For the clausal version of MaxSAT resolution, the other clauses, which our analysis ignores, are guaranteed to be linear in the number of variables at each step, and will *not* be considered again.

Table 4 summarizes the essential aspects of the MaxSAT resolution steps used to derive $m(m+1)+1$ empty clauses. (Also, Section 4.1 clarifies that the formula can be partitioned if $\mathcal{P}$ is ignored.) Similarly to the previous section, the $\mathcal{L}_i$ constraints serve to derive $m+1$ empty clauses, whereas each $\mathcal{M}_j$ constraint serves to derive m empty clauses. In total, we derive $m(m+1)+1$ empty clauses, getting the intended result. As shown in Table 4, for each constraint $\mathcal{L}_i$, start by applying MaxSAT resolution between the hard clause $\mathcal{L}_i \triangleq (\neg n_{i1} \vee \ldots \vee \neg n_{im})$ and soft clause (n_{i1}) to get soft clause $(\neg n_{i2} \vee \ldots \vee \neg n_{im})$, and a few other clauses (which are irrelevant for our purposes). Next, apply $m-1$ additional MaxSAT resolution steps, resolving soft clause $(\neg n_{ik} \vee \ldots \vee \neg n_{im})$ with soft clause (n_{ik}) to get soft clause $(\neg n_{ik+1} \vee \ldots \vee \neg n_{im})$. Clearly, the final MaxSAT resolution step will yield an empty clause. Therefore, over all $m+1$ $\mathcal{L}_i$ constraints, we derive $m+1$ empty clauses.

Table 4 also illustrates the application of the MaxSAT resolution steps to the pairwise encoding of $\mathcal{M}_j$. At iteration i, with $2 \leq i \leq m+1$, we apply i MaxSAT resolution steps to derive another empty clause. In total, we derive m empty clauses for each $\mathcal{M}_j$. An essential aspect is selecting the initial clause from which each sequence of MaxSAT resolution steps is executed. These reused clauses are highlighted in Table 4, and are crucial for getting the right sequence of MaxSAT resolution steps. For each $\mathcal{M}_j$, the MaxSAT resolution steps can be organized in m phases, each yielding an empty soft clause. For phase l, the previous phase $l-1$ produces the clause $(p_{1j} \vee p_{2j} \vee \ldots \vee p_{lj}, 1)$, which is then iteratively simplified, using unit soft clauses, until the empty soft clause for phase l is derived. It should be noted that the first phase uses two unit soft clauses to produce $(p_{1j} \vee p_{2j}, 1)$, which is then used in the second phase. As in Section 4.1, is is immediate that each soft clause is *never* reused.

Regarding the run time complexity, observe that each MaxSAT resolution step runs in time linear on the number of literals in the clauses. The clauses in the problem formulation have no more than $\mathcal{O}(m)$ literals. This also holds true as MaxSAT resolution steps are applied. By analogy with the analysis of the core-guided algorithm, a total of $\mathcal{O}(m^2)$ empty soft clauses will be derived. From the analysis above, summarized in Table 4, deriving the $\mathcal{O}(m^2)$ empty clauses requires a total of $\mathcal{O}(m^3)$ MaxSAT resolution steps. For non-clausal MaxSAT resolution, since the number of generated (non-clausal) terms is constant for each MaxSAT resolution step, then the run time is $\mathcal{O}(m^3)$. In contrast, for clausal MaxSAT resolution [18, Definition 1], since the number of literals for each resolution step is $\mathcal{O}(m^2)$, then the run time becomes $\mathcal{O}(m^5)$.

Table 4: Simplified MaxSAT resolution steps

Constraint	Clauses	Resulting clause(s)
$\mathcal{L}_i$	$(\neg n_{i1} \vee \ldots \vee \neg n_{im}, \top)$, $(n_{i1}, 1)$	$\boxed{(\neg n_{i2} \vee \ldots \vee \neg n_{im}, 1)}, \ldots$
$\mathcal{L}_i$	$(\neg n_{i2} \vee \ldots \vee \neg n_{im}, 1)$, $(n_{i2}, 1)$	$\boxed{(\neg n_{i3} \vee \ldots \vee \neg n_{im}, 1)}, \ldots$
	$\ldots$	
$\mathcal{L}_i$	$(\neg n_{im}, 1)$, $(n_{im}, 1)$	$\boxed{(\bot, 1)}, \ldots$
$\mathcal{M}_j$	$(\neg p_{1j} \vee \neg p_{2j}, \top)$, $(p_{1j}, 1)$	$(\neg p_{2j}, 1), (\neg p_{1j} \vee \neg p_{2j}, \top), \boxed{(p_{1j} \vee p_{2j}, 1)}$
$\mathcal{M}_j$	$(\neg p_{2j}, 1)$, $(p_{2j}, 1)$	$\boxed{(\bot, 1)}$
$\mathcal{M}_j$	$(\neg p_{1j} \vee \neg p_{3j}, \top)$, $(p_{1j} \vee p_{2j}, 1)$	$\boxed{(p_{2j} \vee \neg p_{3j}, 1)}, (\neg p_{1j} \vee \neg p_{3j}, \top)$, $(\neg p_{1j} \vee \neg p_{3j} \vee \neg p_{2j}, 1), \boxed{(p_{1j} \vee p_{2j} \vee p_{3j}, 1)}$
$\mathcal{M}_j$	$(\neg p_{2j} \vee \neg p_{3j}, \top)$, $(p_{2j} \vee \neg p_{3j}, 1)$	$\boxed{(\neg p_{3j}, 1)}, (\neg p_{2j} \vee \neg p_{3j}, \top)$
$\mathcal{M}_j$	$(\neg p_{3j}, 1)$, $(p_{3j}, 1)$	$\boxed{(\bot, 1)}$
	$\ldots$	
$\mathcal{M}_j$	$(\neg p_{1j} \vee \neg p_{m+1j}, \top)$, $(p_{1j} \vee \ldots \vee p_{mj}, 1)$	$\boxed{(p_{2j} \ldots p_{mj} \vee \neg p_{m+1j}, 1)}, \ldots$
$\mathcal{M}_j$	$(\neg p_{2j} \vee \neg p_{m+1j}, \top)$, $(p_{2j} \vee \ldots \vee p_{mj} \vee \neg p_{m+1j}, 1)$	$\boxed{(p_{3j} \ldots p_{mj} \vee \neg p_{m+1j}, 1)}, \ldots$
	$\ldots$	
$\mathcal{M}_j$	$(\neg p_{mj} \vee \neg p_{m+1j}, \top)$, $(p_{mj} \vee \neg p_{m+1j}, 1)$	$\boxed{\neg p_{m+1j}, 1)}, \ldots$
$\mathcal{M}_j$	$(p_{m+1j}, 1)$, $(\neg p_{m+1j}, 1)$	$\boxed{(\bot, 1)}$

Proposition 2. *For the HornMaxSAT encoding of* PHP_m^{m+1}*, there exists a polynomial sequence of MaxSAT resolution steps, each producing a number of constraints polynomial in the size of the problem formulation, that produces* $m(m+1)+1$ *soft empty clauses.*

Proof. (Sketch) The discussion above. □

4.3 Integration in SAT Solvers

This section shows that off-the-shelf MaxSAT solvers, which build on CDCL SAT solvers, can solve PHP_m^{m+1} in polynomial time, provided the right order of conflicts is chosen. In turn, this motivates integrating core-guided MaxSAT reasoning into SAT solvers. Similarly, one could consider integrating MaxSAT resolution (or a mix of both [56]) but, like resolution, MaxSAT resolution is harder to implement in practice. The proposed problem transformation can be applied on demand, and the operation of CDCL can be modified to integrate some form of core-guided reasoning. In contrast to other attempts at extending CDCL, the use of MaxSAT reasoning, will build also on CDCL itself.

MaxHS-like Horn MaxSAT. The reduction to Horn MaxSAT also motivates the development of dedicated MaxSAT solvers. One approach is to build upon MaxHS-solvers [29, 63], since in this case the SAT checks can be made to run in linear time, e.g. using an implementation of LTUR [52]. As indicated above, similar technique can possibly be integrated into SAT solvers.

Handling $\mathcal{P}$ clauses. The $\mathcal{P}$ clauses prevent assigning a variable simultaneously value 0 and value 1. As the analysis for the PHP instances suggests, and the experimental results confirm, these clauses can be responsible for non-polynomial run times. One can envision attempting to solve problems without considering the $\mathcal{P}$ clauses, and then adding these clauses on demand, as deemed necessary to block non-solutions. The operation is similar to the well-known counterexample-guided abstraction refinement paradigm (CEGAR) [24].

5 Experimental Evaluation

This section evaluates the ideas proposed in the paper in practice, for the case of formulas that are known to be hard for resolution-based reasoning. Concretely, the experimental evaluation shows that the performance gains are provided by the proposed problem transformation and the follow-up core-guided MaxSAT solving.

5.1 Experimental Setup

To illustrate the main points of the paper, a number of solvers were tested. However and in order to save space, the results are detailed below only for some of the tested competitors.[5] The families of the evaluated solvers as well as the chosen representatives for the families are listed in Table 5. The family of CDCL SAT solvers comprises MiniSat 2.2 (*minisat*) and Glucose 3 (*glucose*) while the family of SAT solvers strengthened with the use of other powerful techniques (e.g. Gaussian elimination, GA and/or cardinality-based reasoning, CBR) includes lingeling (*lgl*) and CryptoMiniSat (*crypto*). The MaxSAT solvers include the known tools based on implicit minimum-size hitting set enumeration, i.e. MaxHS (*maxhs*) and LMHS (*lmhs*), and also a number of core-guided solvers shown to be best for industrial instances in a series of recent MaxSAT Evaluations[6], e.g. MSCG (*mscg*), OpenWBO16 (*wbo*) and WPM3 (*wpm3*), as well as the recent MaxSAT solver Eva500a (*eva*) based on MaxSAT resolution. Other competitors considered include CPLEX (*lp*), OPB solvers cdcl-cuttingplanes (*cc*) and Sat4j (*sat4j*) as well as a solver based on ZBDDs called ZRes (*zres*).

[5] The discussion focuses on the results of the best performing *representatives* of the considered families of solvers. Solvers that are missing in the discussion are meant to be "dominated" by their representatives, i.e. these solve fewer instances.

[6] http://www.maxsat.udl.cat

Table 5: Families of solvers considered in the evaluation (their best performing representatives are written in *italics*). *SAT+* stands for SAT strengthened with other techniques, *IHS MaxSAT* is for implicit hitting set based MaxSAT, *CG MaxSAT* is for core-guided MaxSAT, *MRes* is for MaxSAT resolution, *MIP* is for mixed integer programming, *OPB* is for pseudo-Boolean optimization, *BDD* is for binary decision diagrams.

SAT		**SAT+**		**IHS MaxSAT**		**CG MaxSAT**			**MRes**	**MIP**	**OPB**		**BDD**
minisat	*glucose*	*lgl*	crypto	*maxhs*	*lmhs*	*mscg*	*wbo*	wpm3	*eva*	*lp*	*cc*	sat4j	*zres*
[33]	[8]	[14, 15]	[66, 67]	[29–31]	[63]	[55]	[51]	[3]	[56]	[41]	[35]	[12]	[22]

Note that three configurations of CPLEX were tested: (1) the default configuration and the configurations used in (2) MaxHS and (3) LMHS. Given the overall performance, we decided to present the results for one best performing configuration, which turned out to be the default one. Also, the performance of CPLEX was measured for the following two types of LP instances: (1) the instances encoded to LP directly from the original CNF formulas (see *lp-cnf*) and (2) the instances obtained from the HornMaxSAT formulas (*lp-wcnf*). A similar remark can be made with regard to the cc solver: it can deal with the original CNF formulas as well as their OPB encodings (the corresponding configurations of the solver are *cc-cnf* and *cc-opb*[7], respectively).

Regarding the IHS-based MaxSAT solvers, both MaxHS and LMHS implement the *Eq-Seeding* constraints [30]. Given that all soft clauses constructed by the proposed HornMaxSAT transformation are *unit* and that the set of all variables of HornMaxSAT formulas is *covered* by the soft clauses, these eq-seeding constraints replicate the complete MaxSAT formula on the MIP side. As a result, after all disjoint unsatisfiable cores are enumerated by MaxHS or LMHS, only one call to an MIP solver is needed to compute the optimum solution. In order to show the performance of an IHS-based MaxSAT solver with this feature disabled, we additionally considered another configuration of LMHS called *lmhs-nes*.[8]

All the conducted experiments were performed in Ubuntu Linux on an Intel Xeon E5-2630 2.60GHz processor with 64GByte of memory. The time limit was set to 1800s and the memory limit to 10GByte for each individual process to run.

5.2 Experimental Results

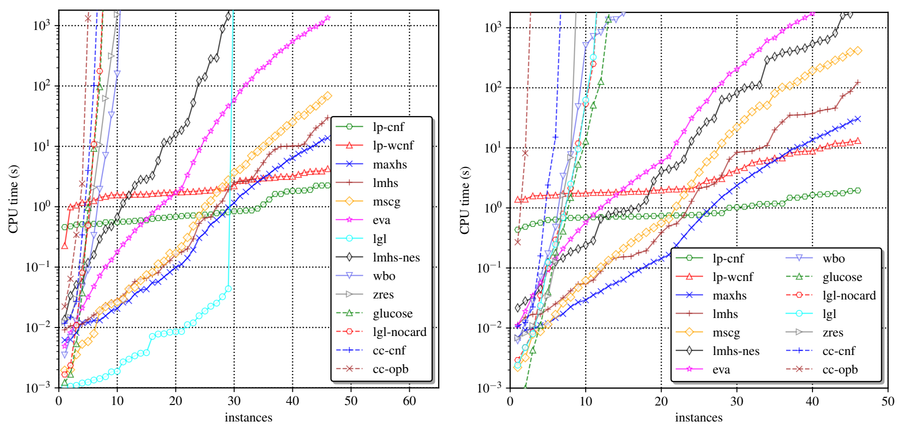
The efficiency of the selected competitors was assessed on the benchmark suite consisting of 3 sets: (1) pigeonhole formulas (PHP) [27], (2) Urquhart formulas (*URQ*) [68], and (3) their combinations (*COMB*).

Pigeonhole Principle benchmarks. The set of PHP formulas contains 2 families of benchmarks differing in the way AtMost1 constraints are encoded: (1) standard pairwise-encoded (*PHP-pw*) and (2) encoded with sequential counters [65] (*PHP-sc*). Each of the families contains 46 CNF formulas encoding the pigeonhole principle for 5 to 100 pigeons. Figure 1[9] shows the performance of the solver on sets PHP-pw and PHP-sc. As can be seen, the MaxSAT solvers (except *eva* and *wbo*) and also *lp-*∗ are able to solve all

[7] The two tested versions of *cc-opb* (implementing linear and binary search) behave almost identically with a minor advantage of linear search. As a result, *cc-opb* stands for the linear search version of the solver.

[8] We chose LMHS (not MaxHS) because it has a command-line option to disable eq-seeding.

[9] Note that all the shown cactus plots below scale the Y axis logarithmically.



(a) PHP-pw (pairwise) (b) PHP-sc (sequential counter)

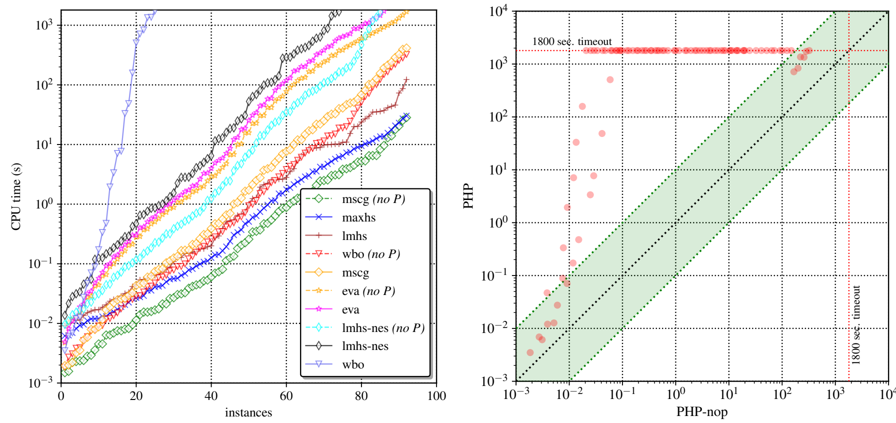
Fig. 1: Performance of the considered solvers on pigeonhole formulas.

instances. As expected, CDCL SAT solvers perform poorly for PHP with the exception of lingeling, which in some cases detects cardinality constraints in PHP-pw. However, disabling cardinality constraints reasoning or considering the PHP-sc benchmarks impairs its performance tremendously. Also note that we were unable to reproduce the performance of *zres* applied to PHP reported in [22].

On discarding $\mathcal{P}$ clauses. To confirm the conjecture that the $\mathcal{P}$ clauses can hamper a MaxSAT solver's ability to get *good* unsatisfiable cores, we also considered both PHP-pw and PHP-sc instances *without* the $\mathcal{P}$ clauses. Figure 2 compares the performance of the MaxSAT solvers working on PHP formulas w/ and w/o the $\mathcal{P}$ clauses. The lines with *(no P)* denote solvers working on the formulas w/o $\mathcal{P}$ (except *maxhs* and *lmhs* whose performance is not affected by removal of $\mathcal{P}$). As detailed in Figure 2b, the efficiency of *wbo* is improved by a few orders of magnitude if the $\mathcal{P}$ clauses are discarded. Also, *mscg* gets about an order of magnitude performance improvement outperforming all the other solvers.

Urquhart benchmarks and combined instances. The URQ instances are known to be hard for resolution [68], but not for BDD-based reasoning [22]. Here, we follow the encoding of [22] to obtain the formulas of varying size given the parameter n of the encoder. In the experiments, we generated 3 CNF formulas for each n from 3 to 30 (i.e. $\text{URQ}_{n,i}$ for $n \in \{3, \ldots, 30\}$ and $i \in \{1, 2, 3\}$), which resulted in 84 instances. As expected, the best performance on the URQ instances is demonstrated by zres. Both *maxhs* and *lmhs* are not far behind. Note that both *maxhs* and *lmhs* do exactly 1 call to CPLEX (due to eq-seeding) after enumerating disjoint unsatisfiable cores. This contrasts sharply with the poor performance of *lp-wcnf*, which is fed with the same problem instances. Lingeling if augmented with Gaussian elimination (GA, see *lgl* in Figure 3a) performs reasonably well being able to solve 29 instances. However, as the result for *lgl-nogauss* suggests, GA is crucial for *lgl* to efficiently decide URQ. Note that *lp-cnf* and *cc-opb* are not shown in Figure 3a due to their inability to solve any instance.

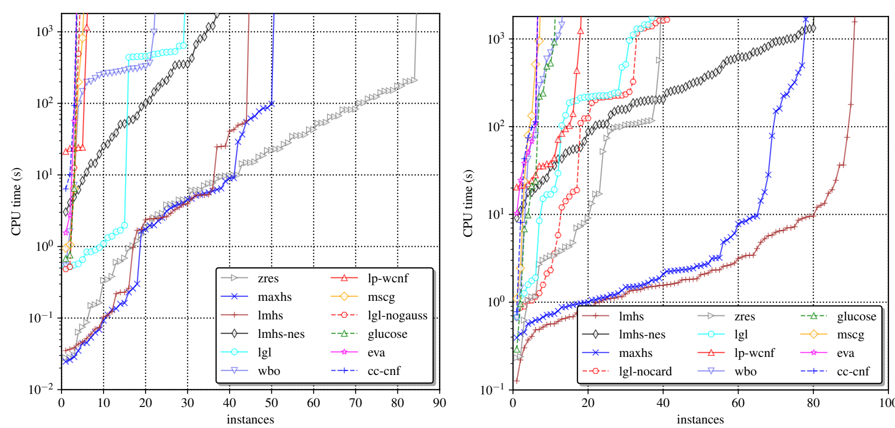
The COMB benchmark set is supposed to inherit the complexity of both PHP and URQ instances and contains formulas $\text{PHP}^{m+1}_{m} \vee \text{URQ}_{n,i}$ with the PHP part being



(a) Cactus plot

(b) Performance of *wbo* w/ and w/o $\mathcal{P}$ clauses

Fig. 2: Performance of MaxSAT solvers on PHP-pw ∪ PHP-sc w/ and w/o $\mathcal{P}$ clauses.



(a) URQ instances

(b) COMB instances

Fig. 3: Performance of the considered solvers on URQ and combined formulas.

pairwise-encoded, where $m \in \{7, 9, 11, 13\}$, $n \in \{3, \ldots, 10\}$, and $i \in \{1, 2, 3\}$, i.e. $|\text{COMB}| = 96$. As one can observe in Figure 3b, even these small m and n result in instances that are hard for most of the competitors. All IHS-based MaxSAT solvers (*maxhs*, *lmhs*, and *lmhs-nes*) perform well and solve most of the instances. Note that *lgl* is confused by the structure of the formulas (neither CBR nor GA helps it solve these instances). The same holds for *zres*. As for CPLEX, while *lp-cnf* is still unable to solve any instance from the COMB set, *lp-wcnf* can also solve only 18 instances. The opposite observation can be made for *cc-cnf* and *cc-opb*.

Summary. As shown in Table 6, given all the considered benchmarks sets, the proposed problem transformation and the follow-up IHS-based MaxSAT solving can cope with by far the largest number of instances overall (see the data for *maxhs*, *lmhs*, and *lmhs-nes*). The core-guided and also resolution based MaxSAT solvers generally per-

Table 6: Number of solved instances per solver.

		glucose	lgl	lgl-no[10]	maxhs	lmhs	lmhs-nes	mscg	wbo	eva	lp-cnf	lp-wcnf	cc-cnf	cc-opb	zres
PHP-pw	(46)	7	29	7	**46**	**46**	29	**46**	10	**46**	**46**	**46**	6	5	10
PHP-sc	(46)	13	11	11	**46**	**46**	45	**46**	15	40	**46**	**46**	6	2	8
URQ	(84)	3	29	4	50	44	37	5	22	3	0	6	3	0	**84**
COMB	(96)	11	37	41	78	**91**	80	7	13	6	0	18	6	0	39
Total	(272)	34	106	63	220	**227**	191	104	60	95	92	116	21	7	141

form well on the pigeonhole formulas (except *wbo*, and this has to be investigated further), which supports the theoretical claims of papers. However, using them does not help solving the URQ and also COMB benchmarks. Also, as shown in Figure 2, the $\mathcal{P}$ clauses can be harmful for MaxSAT solvers. As expected, SAT solvers cannot deal with most of the considered formulas as long as they do not utilize more powerful reasoning (e.g. GA or CBR). However, and as the COMB instances demonstrate, it is easy to construct instances that are hard for the state-of-the-art SAT solvers strengthened with GA and CBR. Finally, one should note the performance gap between *maxhs* (also *lmhs*) and *lp-wcnf* given that they solve the same instances by one call to the same MIP solver with the only difference being the disjoint cores precomputed by *maxhs* and *lmhs*.

6 Conclusions & Research Directions

Resolution is at the core of CDCL SAT solving, but it also represents its Achilles' heel. Many crafted formulas are known to be hard for resolution, with pigeonhole formulas representing a well-known example [27]. More importantly, some of these examples can occur naturally in some practical settings. In the context of MaxSAT, researchers have proposed a dedicated form of resolution, i.e. MaxSAT resolution [18,46], which was also shown not to be more powerful than propositional resolution [18] for the concrete case of pigeonhole formulas [27].

This paper proposes a general transformation for CNF formulas, by encoding the SAT decision problem as a MaxSAT problem over Horn formulas. The transformation is based on the well-known dual-rail encoding, but it is modified such that all clauses are Horn. More importantly, the paper shows that, on this modified formula, MaxSAT resolution can identify in polynomial time a large enough number of empty soft clauses such that this number implies the unsatisfiability of the original pigeonhole formula. Furthermore, the paper shows that the same argument can be used to prove a polynomial run time for the well-known class of core-guided MaxSAT solvers [54].

Experimental results, obtained on formulas known to be hard for SAT solvers, show that different families of MaxSAT solvers perform far better than the best performing SAT solvers, and also ILP solvers, on these instances.

As the paper also hints at, future work will investigate effective mechanisms for integrating Horn MaxSAT problem transformation and MaxSAT reasoning techniques into SAT solvers. In contrast to cutting planes or extended resolution, MaxSAT algorithms already build on CDCL SAT solvers; this is expected to facilitate integration. Another research direction is to investigate similar transformations for the many other examples for which resolution has exponential lower bounds, but also when to opt to apply such transformations.

[10] This represents *lgl-nogauss* for URQ and *lgl-nocard* for PHP-pw, PHP-sc, and COMB.

References

1. I. Abío, R. Nieuwenhuis, A. Oliveras, and E. Rodríguez-Carbonell. BDDs for pseudo-Boolean constraints - revisited. In *SAT*, pages 61–75, 2011.
2. C. Ansótegui, M. L. Bonet, and J. Levy. SAT-based MaxSAT algorithms. *Artif. Intell.*, 196:77–105, 2013.
3. C. Ansótegui, F. Didier, and J. Gabàs. Exploiting the structure of unsatisfiable cores in maxsat. In *IJCAI*, pages 283–289, 2015.
4. R. Asín, R. Nieuwenhuis, A. Oliveras, and E. Rodríguez-Carbonell. Cardinality networks and their applications. In *SAT*, pages 167–180, 2009.
5. R. Asín, R. Nieuwenhuis, A. Oliveras, and E. Rodríguez-Carbonell. Cardinality networks: a theoretical and empirical study. *Constraints*, 16(2):195–221, 2011.
6. A. Atserias, P. G. Kolaitis, and M. Y. Vardi. Constraint propagation as a proof system. In *CP*, pages 77–91, 2004.
7. G. Audemard, G. Katsirelos, and L. Simon. A restriction of extended resolution for clause learning SAT solvers. In *AAAI*, 2010.
8. G. Audemard, J. Lagniez, and L. Simon. Improving Glucose for incremental SAT solving with assumptions: Application to MUS extraction. In *SAT*, pages 309–317, 2013.
9. O. Bailleux and Y. Boufkhad. Efficient CNF encoding of Boolean cardinality constraints. In *CP*, pages 108–122, 2003.
10. O. Bailleux, Y. Boufkhad, and O. Roussel. New encodings of pseudo-Boolean constraints into CNF. In *SAT*, pages 181–194, 2009.
11. P. Beame and T. Pitassi. Simplified and improved resolution lower bounds. In *FOCS*, pages 274–282, 1996.
12. D. L. Berre and A. Parrain. The Sat4j library, release 2.2. *JSAT*, 7(2-3):59–6, 2010.
13. A. Biere. Picosat essentials. *JSAT*, 4(2-4):75–97, 2008.
14. A. Biere. Lingeling, plingeling and treengeling entering the SAT competition 2013. In A. Balint, A. Belov, M. Heule, and M. Järvisalo, editors, *Proceedings of SAT Competition 2013*, volume B-2013-1 of *Department of Computer Science Series of Publications B*, pages 51–52. University of Helsinki, 2013.
15. A. Biere. Lingeling essentials, A tutorial on design and implementation aspects of the SAT solver lingeling. In *Pragmatics of SAT workshop*, page 88, 2014.
16. A. Biere, D. L. Berre, E. Lonca, and N. Manthey. Detecting cardinality constraints in CNF. In *SAT*, pages 285–301, 2014.
17. A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.
18. M. L. Bonet, J. Levy, and F. Manyà. Resolution for Max-SAT. *Artif. Intell.*, 171(8-9):606–618, 2007.
19. R. E. Bryant, D. L. Beatty, K. S. Brace, K. Cho, and T. J. Sheffler. COSMOS: A compiled simulator for MOS circuits. In *DAC*, pages 9–16, 1987.
20. S. R. Buss. Polynomial size proofs of the propositional pigeonhole principle. *J. Symb. Log.*, 52(4):916–927, 1987.
21. S. R. Buss and G. Turán. Resolution proofs of generalized pigeonhole principles. *Theor. Comput. Sci.*, 62(3):311–317, 1988.
22. P. Chatalic and L. Simon. Multiresolution for SAT checking. *International Journal on Artificial Intelligence Tools*, 10(4):451–481, 2001.
23. V. Chvátal and E. Szemerédi. Many hard examples for resolution. *J. ACM*, 35(4):759–768, 1988.
24. E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement for symbolic model checking. *J. ACM*, 50(5):752–794, 2003.
25. M. Codish and M. Zazon-Ivry. Pairwise cardinality networks. In *LPAR*, pages 154–172, 2010.

26. S. A. Cook. A short proof of the pigeon hole principle using extended resolution. *ACM SIGACT News*, 8(4):28–32, 1976.
27. S. A. Cook and R. A. Reckhow. The relative efficiency of propositional proof systems. *J. Symb. Log.*, 44(1):36–50, 1979.
28. W. J. Cook, C. R. Coullard, and G. Turán. On the complexity of cutting-plane proofs. *Discrete Applied Mathematics*, 18(1):25–38, 1987.
29. J. Davies and F. Bacchus. Solving MAXSAT by solving a sequence of simpler SAT instances. In *CP*, pages 225–239, 2011.
30. J. Davies and F. Bacchus. Exploiting the power of mip solvers in maxsat. In *SAT*, pages 166–181, 2013.
31. J. Davies and F. Bacchus. Postponing optimization to speed up MAXSAT solving. In *CP*, pages 247–262, 2013.
32. W. F. Dowling and J. H. Gallier. Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *J. Log. Program.*, 1(3):267–284, 1984.
33. N. Eén and N. Sörensson. An extensible SAT-solver. In *SAT*, pages 502–518, 2003.
34. N. Eén and N. Sörensson. Translating pseudo-Boolean constraints into SAT. *JSAT*, 2(1-4):1–26, 2006.
35. Jan Elffers' personal webpage. http://www.csc.kth.se/~elffers.
36. Z. Fu and S. Malik. On solving the partial MAX-SAT problem. In *SAT*, pages 252–265, 2006.
37. E. Goldberg. Testing satisfiability of CNF formulas by computing a stable set of points. In *CADE*, pages 161–180, 2002.
38. E. Goldberg. Testing satisfiability of CNF formulas by computing a stable set of points. *Ann. Math. Artif. Intell.*, 43(1):65–89, 2005.
39. A. Haken. The intractability of resolution. *Theor. Comput. Sci.*, 39:297–308, 1985.
40. J. Huang. Extended clause learning. *Artif. Intell.*, 174(15):1277–1284, 2010.
41. IBM ILOG: CPLEX optimizer 12.7.0. http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer, 2016.
42. S. Jabbour, J. Marques-Silva, L. Sais, and Y. Salhi. Enumerating prime implicants of propositional formulae in conjunctive normal form. In *JELIA*, pages 152–165, 2014.
43. D. Jovanovic and L. M. de Moura. Cutting to the chase solving linear integer arithmetic. In *CADE*, pages 338–353, 2011.
44. D. Jovanovic and L. M. de Moura. Cutting to the chase - solving linear integer arithmetic. *J. Autom. Reasoning*, 51(1):79–108, 2013.
45. M. Koshimura, T. Zhang, H. Fujita, and R. Hasegawa. QMaxSAT: A partial Max-SAT solver. *JSAT*, 8(1/2):95–100, 2012.
46. J. Larrosa, F. Heras, and S. de Givry. A logical approach to efficient Max-SAT solving. *Artif. Intell.*, 172(2-3):204–233, 2008.
47. V. M. Manquinho, P. F. Flores, J. Marques-Silva, and A. L. Oliveira. Prime implicant computation using satisfiability algorithms. In *ICTAI*, pages 232–239, 1997.
48. J. Marques-Silva, A. Ignatiev, C. Mencía, and R. Peñaloza. Efficient reasoning for inconsistent Horn formulae. In *JELIA*, pages 336–352, 2016.
49. J. Marques-Silva and J. Planes. On using unsatisfiability for solving maximum satisfiability. *CoRR*, abs/0712.1097, 2007.
50. R. Martins, S. Joshi, V. M. Manquinho, and I. Lynce. Incremental cardinality constraints for MaxSAT. In *CP*, pages 531–548, 2014.
51. R. Martins, V. M. Manquinho, and I. Lynce. Open-WBO: A modular MaxSAT solver,. In *SAT*, pages 438–445, 2014.
52. M. Minoux. LTUR: A simplified linear-time unit resolution algorithm for Horn formulae and computer implementation. *Inf. Process. Lett.*, 29(1):1–12, 1988.
53. A. Morgado, C. Dodaro, and J. Marques-Silva. Core-guided MaxSAT with soft cardinality constraints. In *CP*, pages 564–573, 2014.

54. A. Morgado, F. Heras, M. H. Liffiton, J. Planes, and J. Marques-Silva. Iterative and core-guided MaxSAT solving: A survey and assessment. *Constraints*, 18(4):478–534, 2013.
55. A. Morgado, A. Ignatiev, and J. Marques-Silva. MSCG: Robust core-guided MaxSAT solving. *JSAT*, 9:129–134, 2015.
56. N. Narodytska and F. Bacchus. Maximum satisfiability using core-guided MaxSAT resolution. In *AAAI*, pages 2717–2723, 2014.
57. J. Nordström. On the interplay between proof complexity and SAT solving. *SIGLOG News*, 2(3):19–44, 2015.
58. T. Ogawa, Y. Liu, R. Hasegawa, M. Koshimura, and H. Fujita. Modulo based CNF encoding of cardinality constraints and its application to MaxSAT solvers. In *ICTAI*, pages 9–17, 2013.
59. K. Pipatsrisawat and A. Darwiche. On the power of clause-learning SAT solvers as resolution engines. *Artif. Intell.*, 175(2):512–525, 2011.
60. A. Previti, A. Ignatiev, A. Morgado, and J. Marques-Silva. Prime compilation of non-clausal formulae. In *IJCAI*, pages 1980–1988, 2015.
61. A. A. Razborov. Proof complexity of pigeonhole principles. In *DLT*, pages 100–116, 2001.
62. J. Roorda and K. Claessen. A new SAT-based algorithm for symbolic trajectory evaluation. In *CHARME*, pages 238–253. Springer, 2005.
63. P. Saikko, J. Berg, and M. Järvisalo. LMHS: A SAT-IP hybrid MaxSAT solver. In *SAT*, pages 539–546, 2016.
64. N. Segerlind. The complexity of propositional proofs. *Bulletin of Symbolic Logic*, 13(4):417–481, 2007.
65. C. Sinz. Towards an optimal CNF encoding of Boolean cardinality constraints. In *CP*, pages 827–831, 2005.
66. M. Soos. Enhanced gaussian elimination in dpll-based SAT solvers. In *POS@SAT*, pages 2–14, 2010.
67. M. Soos, K. Nohl, and C. Castelluccia. Extending SAT solvers to cryptographic problems. In *SAT*, pages 244–257, 2009.
68. A. Urquhart. Hard examples for resolution. *J. ACM*, 34(1):209–219, 1987.
69. J. P. Warners. A linear-time transformation of linear inequalities into conjunctive normal form. *Inf. Process. Lett.*, 68(2):63–69, 1998.