Propositional Proof Systems Based on Maximum Satisfiability[☆]

Maria Luisa Bonet[a], Sam Buss[b,1], Alexey Ignatiev[c], Antonio Morgado[d], Joao Marques-Silva[e]

[a] *Computer Science Department, Universidad Politécnica de Cataluña, Barcelona, Spain*
[b] *Department of Mathematics, University of California, San Diego, USA*
[c] *Faculty of IT, Monash University, Melbourne, Australia*
[d] *INESC-ID/IST, University of Lisbon, Lisbon, Portugal*
[e] *IRIT, CNRS, Toulouse, France*

Abstract

The paper describes the use of dual-rail MaxSAT systems to solve Boolean satisfiability (SAT), namely to determine if a set of clauses is satisfiable. The MaxSAT problem is the problem of satisfying the maximum number of clauses in an instance of SAT. The dual-rail encoding adds extra variables for the complements of variables, and allows encoding an instance of SAT as a Horn MaxSAT problem. We discuss three implementations of dual-rail MaxSAT: core-guided systems, minimal hitting set (MaxHS) systems, and MaxSAT resolution inference systems. All three of these can be more efficient than resolution and thus than conflict-driven clause learning (CDCL). All three systems can give polynomial size refutations for the pigeonhole principle, the doubled pigeonhole principle and the mutilated chessboard principles. The dual-rail MaxHS MaxSat system can give polynomial size proofs of the parity principle. However, dual-rail MaxSAT resolution requires exponential size proofs for the parity principle; this is proved by showing that constant depth Frege augmented with the pigeonhole principle can polynomially simulate dual-rail MaxSAT resolution. Consequently, dual-rail MaxSAT resolution does not simulate cutting planes. We further show that core-guided dual-rail MaxSAT and weighted dual-rail MaxSAT resolution polynomially simulate resolution. Finally, we report the results of experiments with core-guided dual-rail MaxSAT and MaxHS dual-rail MaxSAT showing strong performance by these systems.

Keywords: Propositional Proof Systems, Maximum Satisfiability, Clause Learning, Resolution

[☆] This work builds on the following conference papers [38],[17] and [55].
[1] Supported in part by Simons Foundation grant 578919.

1. Introduction

The decision problem for propositional logic, i.e., the propositional satisfiability (SAT) problem, is a well-known NP-complete problem [24]. As a result, to the best of our knowledge, any complete algorithm for the SAT problem problem may require exponential time in the worst-case. Nevertheless, in the case of SAT, practice defies theory, and implementations of SAT algorithms (i.e., SAT solvers) have made remarkable progress over the last two and a half decades [46]. Capable of solving formulas with a few hundred variables in the early 1990s, and so widely perceived as an academic curiosity, SAT solvers now routinely solve formulas with millions of variables. The reason for this success is almost exclusively explained by the development of Conflict-Driven Clause Learning (CDCL) SAT solvers starting in the mid 1990s [48, 49, 46]. Along with the practical deployment of CDCL and its widespread industry adoption, there has been work on understanding the theoretical power of the underlying clause learning proof system associated with CDCL. Over the last decade and a half, a sequence of results eventually proved that CDCL with restarts polynomially simulates the general resolution proof system, and vice-versa [10, 34, 60, 6, 12]. (It is still an open problem with CDCL without restarts polynomially simulates resolution; however, a result of [12] proves that CDCL without restarts simulates resolution if you allow first modifying the input formula). The progress made has also motivated a number of additional challenges; a concrete example being whether it is possible to devise practically efficient implementations of proof systems that are more powerful than resolution, and hence more powerful than CDCL. Recent years have witnessed a growing number of attempts at implementing proof systems stronger than resolution [36, 7, 31, 73, 35], aiming at eventually replacing present-day CDCL SAT solvers.

This paper reports one concrete effort on developing SAT solving tools using propositional proof systems stronger than CDCL. The proposed approach first transforms a problem with the *dual-rail encoding* to an instance of Horn Maximum Satisfiability (Horn MaxSAT), and then using a MaxSAT solver. We refer to this approach as *dual-rail MaxSAT* [38, 17, 55]. The dual-rail MaxSAT problem reduction allows a wide range of decision and optimization problems to be reduced to Horn MaxSAT [45], a special case of MaxSAT (which is also NP-hard [41]). In the case of decision problems, a MaxSAT problem formulation can be obtained by checking whether the optimum cost corresponds to satisfying (or not satisfying) the original formula. Currently, the most efficient MaxSAT solvers are based on sequences of calls to a SAT solver (or oracle). This suggests that a proof system stronger than CDCL could be obtained by building on highly optimized SAT solvers.

Since the dual-rail encodings can be solved using different MaxSAT solving algorithms, dual-rail MaxSAT is not a single proof system; instead, it is a framework for multiple possible proof systems, depending on what MaxSAT algorithm is used. Concretely, the present paper considers three possible instantiations of the dual-rail MaxSAT proof system: dual-rail MaxSAT resolution (an inference system), core-guided dual-rail MaxSAT algorithms, and minimum hitting set (MaxHS-like) dual-rail MaxSAT algorithms. These systems all use the dual-rail encoding, but then solve the resulting Horn MaxSAT instance using MaxSAT resolution, a core-guided MaxSAT solver, or a MaxHS-like MaxSAT solver (respectively).

Dual-rail MaxSAT resolution is a propositional proof system in the traditional sense of having derivations formed with inference rules. Core-guided MaxSAT and MaxHS-like MaxSAT do not correspond to traditional proof systems with inferences; instead, they are algorithms for solving MaxSAT. They depend on calls to a SAT solver and, in the case of MaxHS-like algorithms, calls to a minimum hitting set algorithm. Nonetheless, they can be viewed as abstract proof systems in the sense of Cook-Reckhow [25], provided that the calls to the SAT solvers and the minimum hitting set algorithm return certificates of correctness. When we talk about the existence of polynomial size proofs (see Figure 1) in these theories, we mean that the algorithms can run in polynomial time for suitably chosen versions of SAT solvers and minimum hitting set algorithms with the appropriate (non-deterministic) choices for unsatisfiable cores and minimum size hitting sets. It is permitted that the "suitably chosen" algorithms are tailored to the specific principles being proved. The experimental results in Section 7, however, use generic SAT solvers and generic minimum hitting set algorithms. The dual-rail MaxSAT algorithms in our experiments improve on the usual CDCL-based methods, but they are still exponential time, not polynomial time, even though our theorems give the possibility of polynomial time.

The fact that the dual-rail MaxSAT proof system can be *configured* with different MaxSAT algorithms enables studying related proof systems of possibly different powers. In fact, as shown in Figure 1, MaxHS-like dual-rail MaxSAT and dual-rail MaxSAT resolution have an exponential separation for the parity principle.

The outline of the paper is as follows. Section 2 first defines MaxSAT and weighted MaxSAT; then describes MaxSAT resolution, core-guided MaxSAT and MaxHS-like MaxSAT; and finally defines the pigeonhole principle, the doubled pigeonhole principle, mutilated chessboard principle, and the parity principle. Section 3 defines the dual-rail encoding in general, and explicitly gives the dual-rail encodings for the four combinatorial principles. Section 4 gives explicit polynomial size proofs for the four combinatorial principles in the three dual-rail MaxSAT proof sytems, except that the parity principle is only shown to have polynomial size MaxHS-like dual-rail MaxSAT proofs. Section 5 proves that core-guided dual-rail MaxSAT and weighted dual-rail MaxSAT resolution both polynomially simulate general resolution.

Section 6 proves that constant depth Frege proofs augmented with instances of the pigeonhole principle can polynomially simulate dual-rail MaxSAT resolution. This gives a nearly tight characterization of the power of dual-rail MaxSAT resolution. As corollaries, we obtain that dual-rail MaxSAT resolution does not have polynomial size proofs of the parity principle, and does not polynomially simulate cutting planes. Section 7 provides experimental results, providing empirical evidence supporting the results in earlier sections.

	Core-guided MaxSAT	MaxHS-like MaxSAT	MaxSAT resolution
PHP	Poly, Theorem 11	Poly, Theorem 18	Poly, Theorem 8
2PHP	Poly, Theorem 13	Poly, Theorem 23	Poly, Theorem 10
Parity	?	Poly, Theorem 26	Exp, Corollary 34
Mutilated chessboard	Poly, Theorem 12	Poly, Theorem 19	Poly, Theorem 9

Figure 1: The strengths of the three systems for the combinatorial principles. "Poly" means there are polynomial size proofs. "Exp" means that exponential size proofs are required.

Figure 1 shows which of the four combinatorial principles have polynomial size proofs in which of the three dual-rail MaxSAT systems.

This paper builds on earlier work [38, 17, 55], but extends this earlier work in several new directions. Concretely, Sections 4.1.1, 4.2.1, 7.2 and 7.6 provide a more detailed account of the work in [38]. Sections 4.1.3, 5.1, 5.2, 5.3, 6 and 7.3 provide a more detailed account of the work in [17]. Sections 4.3.1 and 4.3.2 provide a more detailed account of the work in [55]. Finally, Sections 4.1.2, 4.2.2, 4.3.3, 7.4 and 7.5 contain novel results.

2. Preliminaries

2.1. MaxSAT and Weighted MaxSAT

MaxSAT is the problem of finding a truth assignment that minimizes the number of falsified clauses of a CNF formula. MaxSAT has several generalizations. To define them, we need to give weights to clauses, with the weight indicating the "cost" of falsifying the clause. A *weighted* clause is written (A, w) where A is a clause and $w \in \{1, 2, 3, \ldots\} \cup \{\top\}$. The value $\top$ is viewed as equaling infinity, but we write "$\top$" instead of "∞". A typical use of weighted clauses is for *Partial MaxSAT*, where the clauses of Γ are partitioned into *soft* clauses and *hard* clauses. Soft clauses may be falsified and have weight 1; hard clauses may not be falsified and have weight $\top$. So *Partial MaxSAT* is the problem of finding an assignment that satisfies all the hard clauses and minimizes the number of falsified soft clauses. In *Weighted Partial MaxSAT*, the soft clauses may have any (finite integer) weight ≥ 1. *Weighted Partial MaxSAT* is the problem of finding an assignment that satisfies all the hard clauses and minimizes the sum of the weights of falsified soft clauses.

2.2. Weighted MaxSAT: Inference Systems and Algorithms

This section describes three systems for solving (partial) MaxSAT: first MaxSAT Resolution, then Core-guided MaxSAT algorithms, and finally Minimum Hitting Set based MaxSAT algorithms (MaxHS-like algorithms). These will be used for three different instantiations of dual-rail MaxSAT.

2.2.1. MaxSAT Resolution

The *MaxSAT resolution* calculus is a sound and complete calculus for MaxSAT based on resolution. This system was first defined by [43], and proven complete by [18]. A similar calculus can also be defined for Partial MaxSAT and Weighted Partial MaxSAT. (Weighted) (Partial) MaxSAT resolution is based on inference rules. In classical resolution, every application of the resolution rule adds a new clause to the system. The inference rule for (Weighted) (Partial) MaxSAT, however, *replaces* its hypothesis clauses by a different set of clauses. In other words, a clause may be used only once as a hypothesis of a (Weighted) (Partial) MaxSAT resolution inference.

Considering the case of two clauses with weight one, the MaxSAT resolution rule is:

$$\frac{\begin{array}{c}(x \vee A, 1)\\(\overline{x} \vee B, 1)\end{array}}{\begin{array}{c}(A \vee B, 1)\\(x \vee A \vee \overline{B}, 1)\\(\overline{x} \vee \overline{A} \vee B, 1)\end{array}} \tag{1}$$

The notation $x \vee A \vee \overline{B}$, where $A = a_1 \vee \cdots \vee a_s$ and $B = b_1 \vee \cdots \vee b_t$, is the abbreviation of the set of clauses (which depends on the ordering of the literals in clause B)

$$\begin{array}{l}x \vee a_1 \vee \cdots \vee a_s \vee \overline{b}_1\\x \vee a_1 \vee \cdots \vee a_s \vee b_1 \vee \overline{b}_2\\\qquad\vdots\\x \vee a_1 \vee \cdots \vee a_s \vee b_1 \vee \cdots \vee b_{t-1} \vee \overline{b_t}\end{array} \tag{2}$$

When $t = 0$, $\overline{B}$ is the constant true, so $x \vee A \vee \overline{B}$ denotes the empty set of clauses. $\overline{x} \vee \overline{A} \vee B$ is defined similarly.

Observe that in the MaxSAT rule in Equation (1) at most one of the premises is false, and similarly at most one of the conclusions is false. Thus by construction, the rule maintains the total weight of falsified clauses.

In the general case of clauses with finite weights w_1 and w_2, the inference rule is:

$$\frac{\begin{array}{c}(x \vee A, w_1)\\(\overline{x} \vee B, w_2)\end{array}}{\begin{array}{c}(A \vee B, k)\\(x \vee A, w_1 - k)\\(\overline{x} \vee B, w_2 - k)\\(x \vee A \vee \overline{B}, k)\\(\overline{x} \vee \overline{A} \vee B, k)\end{array}} \tag{3}$$

where $1 \leq k \leq min(w_1, w_2)$. In the rule, conclusion clauses with weight 0 are omitted; e.g., at least one of the second or third conclusions is omitted when $k = min(w_1, w_2)$; both are omitted if $k = w_1 = w_2$.

If one or both weights are $\top$ and $1 \leq k \leq w$, the following rules apply

$$\frac{\begin{array}{c}(x \vee A, w)\\(\overline{x} \vee B, \top)\end{array}}{\begin{array}{c}(A \vee B, k)\\(x \vee A, w-k)\\(x \vee A \vee \overline{B}, k)\\(\overline{x} \vee B, \top)\end{array}} \qquad \text{and} \qquad \frac{\begin{array}{c}(x \vee A, \top)\\(\overline{x} \vee B, \top)\end{array}}{\begin{array}{c}(A \vee B, \top)\\(x \vee A, \top)\\(\overline{x} \vee B, \top)\end{array}} \tag{4}$$

for finite w. The second rule is just the ordinary resolution inference, as the premises are still available as conclusions.

After applying the rule, we remove tautologies, and collapse repeated occurrences of variables in clauses. As noted, for MaxSAT inferences the premises are *replaced* with the conclusions. Note that these inferences depend on the ordering of the literals $b_1, \dots, b_t$. This means that, in general, there are multiple ways to apply the rule to a given pair of clauses.

It is easy to check that if a truth assignment τ falsifies the formula $x \vee A \vee \overline{B}$, then it falsifies exactly one of the clauses in (2), and similarly for $\overline{x} \vee \overline{A} \vee B$. Also, if τ makes one of the premises of (3) with weight w false, then the sum of the weights of the falsified conclusions is w. Likewise, if τ satisfies both premises of (3), then it satisfies all the conclusions. Thus we have shown that for any fixed truth assignment, the total weight of the falsified clauses (at most one) in the premises of (3) is equal to the total weight of the falsified clauses in the conclusion of (3). Similar considerations apply to inferences shown in (4). The soundness of the Weighted MaxSAT rules (3) and (4) follows immediately.

A (Weighted) (Partial) MaxSAT refutation starts with a multiset Γ of clauses. After each inference, the multiset of clauses is updated by removing the rule's premises and adding its conclusions. The MaxSAT refutation ends with a multiset containing $k > 0$ occurrences of the empty clause $\bot$, possibly with weights.

The rules give a sound and complete system for Weighted Partial MaxSAT [18]. Given a set Γ of weighted clauses and a truth assignment τ, the cost of τ is the sum of weights of the clauses that τ falsifies; the cost is infinite if some hard clause is falsified. The following are the soundness and completeness theorem statements for Weighted Partial MaxSAT.

Theorem 1. *Soundness: if there is a derivation from Γ of a set of empty clauses with weights summing up to w, then there is no assignment of cost $< w$.*

Theorem 2. *Completeness: if w is the minimum cost of an assignment for Γ, then there is a derivation from Γ of empty clauses with weights adding up to w.*

It is useful to also have the following two rules when dealing with soft clauses with weights bigger than 1.

$$\text{Extraction: } \frac{(A, w_1{+}w_2)}{(A, w_1) \qquad (A, w_2)}$$

$$\text{Contraction: } \frac{(A, w_1) \qquad (A, w_2)}{(A, w_1{+}w_2)}$$

The contraction and extraction rules allow w_1 and w_2 to be finite or $\top$, under the convention that $\top + w = w + \top = \top$.

Our convention thus is that dual-rail MaxSAT resolution has all of the inference rules resolution, extraction and contraction. With the presence of extraction and contraction, the resolution inference for finite w can be formulated simply as

$$\frac{\begin{array}{c}(x \vee A, w)\\ (\overline{x} \vee B, w)\end{array}}{\begin{array}{c}(A \vee B, w)\\ (x \vee A \vee \overline{B}, w)\\ (\overline{x} \vee \overline{A} \vee B, w)\end{array}} \tag{5}$$

The MaxSAT resolution system is unusual in that its rules have multiple conclusions. This can have unexpected consequences. For example, one might expect that since soft clauses cannot be reused, this means that the portion of a MaxSAT refutation that uses soft clauses is tree-like. This is not true however, because an inference may have multiple soft clauses among its conclusions, which can be used at different points in the refutation.

2.2.2. Core-Guided MaxSAT Algorithms

This section describes core-guided MaxSAT algorithms [32, 47, 33, 54, 4, 57, 53, 50]. These algorithms are used as refutation systems for Partial MaxSAT. Our constructions will use the MSU3 [47] algorithm, shown in Algorithm 1, which is a core-guided MaxSAT algorithm used in many state-of-the-art MaxSAT solvers. Other core-guided algorithms could be considered and are present in the experimental section. The idea of MSU3 [47] is to iteratively call a SAT solver on a working formula, and to refine a lower bound on the number of soft clauses that must be falsified in order to achieve satisfiability. The pseudo-code of MSU3 is shown in Algorithm 1. Initially the lower bound λ is set to 0 (no soft clauses need to be falsified), and the working formula $\mathcal{F}_W$ is set to the original formula (line 2). In each iteration, the satisfiability of the working formula $\mathcal{F}_W$ is decided with a SAT solver[2] (by the call to `SAT`(.) in line 4). In case the formula is satisfiable, the algorithm stops. The minimum number of falsified clauses corresponds to the lower bound λ, which is returned together with the assignment $\mathcal{A}$ provided by the SAT solver (line 5).

In case the formula is unsatisfiable, the SAT solver produces a set of unsatisfiable clauses, referred to as an *unsatisfiable core* ($\mathcal{C}$ in line 4).[3] At this point

[2] Note that any SAT solver can be used in MSU3, as long as the SAT solver provides a satisfying truth assignment if the input formula is satisfiable, and an unsatisfiable core if the input formula is unsatisfiable.

[3] An unsatisfiable core is a subset of the input formula that is unsatisfiable. Note that the MSU3 algorithm does not require the unsatisfiable core to be minimal. The same observation applies to *any* core-guided algorithm, as argued in earlier work [32, 47, 33, 54].

Algorithm 1: The MSU3 core-guided MaxSAT algorithm [47]

1 **Input:** $\mathcal{F} = \mathcal{S} \cup \mathcal{H}$, *MaxSAT formula with soft clauses* $\mathcal{S}$ *and hard clauses* $\mathcal{H}$
2 $(R, \mathcal{F}_W, \lambda) \leftarrow (\emptyset, \mathcal{S} \cup \mathcal{H}, 0)$
3 **while** true **do**
4     $(\text{st}, \mathcal{C}, \mathcal{A}) \leftarrow \texttt{SAT}(\mathcal{F}_W)$
5   **if** st **then return** $\lambda, \mathcal{A}$
6     $\lambda \leftarrow \lambda + 1$
7   **for** $c \in \mathcal{C} \cap \mathcal{S}$ **do**
8         $R \leftarrow R \cup \{r_c\}$ `//` r_c `is a fresh variable`
9         $\mathcal{S} \leftarrow \mathcal{S} \setminus \{c\}$
10         $\mathcal{H} \leftarrow \mathcal{H} \cup \{c \cup \{r_c\}\}$
11     $\mathcal{F}_W \leftarrow \mathcal{S} \cup \mathcal{H} \cup \texttt{CNF}(\sum_{r \in R} r \leq \lambda)$

the algorithm will increase the lower bound by one (line 6), and for each soft clause in the core, a new literal is added to the soft clause. The new fresh literal is referred to as a *relaxation variable* and the resulting *relaxed clause* is marked as hard (lines 7 to 10). Note that each soft clause can be relaxed at most once (that is, it will contain at most one relaxation variable), as it is marked as hard after being relaxed.

The new working formula consists of the reduced set of soft clauses, the augmented set of hard clauses, and additionally the CNF encoding a cardinality constraint (line 11), represented with hard clauses. This cardinality constraint contains all the relaxation variables added so far (including from previous iterations), and encodes that the sum of the relaxation variables assigned to true has to be smaller or equal to the current lower bound. Limiting the number of relaxation variables assigned to true to be at most equal to the lower bound, implies that the number of (original) soft clauses falsified is at most equal to the lower bound. The cardinality constraint is encoded as a set of hard clauses before adding it to the working formula (through the function CNF(.) in line 11 using an existing cardinality constraint encoding [16]). Note that in the next iteration, the SAT solver will be called with the new working formula (created fresh in line 11), as such considering the new set of reduced soft clauses, the new set of augmented hard clauses, and the new cardinality constraint, disregarding all the previous ones.

It is worth discussing more the cardinality constraints $\texttt{CNF}(\sum_{r \in R} r \leq \lambda)$. We always assume a constraint is expressed as a CNF formula, so it can be used by a SAT solver. The simplest is to let the cardinality constraint be the straightforward translation of the condition $\sum_{r \in R} r \leq \lambda$ to a CNF formula. In all the applications in the present paper (see Section 4.2), this turns the cardinality constraints into polynomial size CNF formulas. In more general settings, one could introduce additional variables to express a cardinality constraint in order to avoid exponentially large CNF formulas. The main property needed is that if more than λ many variables $r \in R$ are set true, then unit propagation from the clauses in $\texttt{CNF}(\sum_{r \in R} r \leq \lambda)$ yields a contradiction.

As discussed earlier, the MSU3 core-guided algorithm is not a traditional proof system with inferences, but nonetheless can be viewed as an abstract proof system. Theorems 11, 12 and 13 give upper bounds on the run time of MSU3 by stating it "is able to conclude in polynomial time" that the input is unsatisfiable. What this means is that there is a choice of actions for the the SAT solver when lead the MSU3 to halt within polynomial time. For the proofs of our theorems it is permitted that the SAT solver is customized to the specific family of combinatorial principles under consideration. Of course, for practical applications, we are much more interested in the behavior of MSU3 with coupled with a generic SAT solver; for this, see the experiments in Section 5.

2.2.3. Minimum Hitting Sets MaxSAT Algorithms

The third system for MaxSAT is based on MaxHS-like MaxSAT algorithms. Similar to the core-guided MaxSAT algorithms, MaxHS-like MaxSAT algorithms can be viewed as refutation systems for (Weighted) Partial MaxSAT. MaxHS-like MaxSAT algorithms are based on Minimum Hitting Sets. Let $\mathcal{F}$ be an unsatisfiable CNF formula. A formula $M \subseteq \mathcal{F}$ is a *Minimal Unsatisfiable Subformula* (MUS) of $\mathcal{F}$ if:

(i) M is unsatisfiable,

(ii) $\forall C \in M$, $M \setminus \{C\}$ is satisfiable

The set of MUS's of $\mathcal{F}$ is denoted by MUS($\mathcal{F}$). Dually, a formula $S \subseteq \mathcal{F}$ is a *Maximal Satisfiable Subformula* (MSS) of $\mathcal{F}$ if:

(i) S is satisfiable,

(ii) $\forall C \in \mathcal{F} \setminus S$, $S \cup \{C\}$ is unsatisfiable

The set of MSS's of $\mathcal{F}$ is denoted by MSS($\mathcal{F}$). Finally, a formula $R \subseteq \mathcal{F}$ is a *Minimal Correction Subset* (MCS), or, co-MSS of $\mathcal{F}$, if $\mathcal{F} \setminus R \in$ MSS($\mathcal{F}$), or, explicitly, if:

(i) $\mathcal{F} \setminus R$ is satisfiable,

(ii) $\forall C \in R$, $(\mathcal{F} \setminus R) \cup \{C\}$ is unsatisfiable

The set of MCS's of $\mathcal{F}$ is denoted by MCS($\mathcal{F}$).

The MUS's, MSS's and MCS's of a given unsatisfiable formula $\mathcal{F}$ are connected via the so-called Hitting Sets Duality theorem, first proved in [64]. The theorem states that M is an MUS of $\mathcal{F}$ if and only if M is an irreducible hitting set[4] of MCS($\mathcal{F}$), and vice versa: $R \in$ MCS($\mathcal{F}$) iff R is an irreducible hitting set of MUS($\mathcal{F}$).

The idea of the minimum hitting set MaxSAT algorithm is to guess an MCS C of the given MaxSAT formula $\mathcal{F} = \mathcal{S} \cup \mathcal{H}$. The guesses C are made in increasing size using a Minimum Hitting Set solver, and then a SAT solver is used for testing if C is indeed a MCS of $\mathcal{F}$. If the SAT solver says true then a solution has been found, otherwise a new unsatisfiable core has been discovered. The unsatisfiable cores are used by the Minimum Hitting Set solver for making

[4]For a given collection $\mathcal{S}$ of arbitrary sets, a set H is called a hitting set of $\mathcal{S}$ if for all $S \in \mathcal{S}$, $H \cap S \neq \emptyset$. A hitting set H is irreducible, if no $H' \subset H$ is a hitting set of $\mathcal{S}$.

Algorithm 2: Pseudo-code of the basic MaxHS algorithm [27]

1 **Input:** $\mathcal{F} = \mathcal{S} \cup \mathcal{H}$, *MaxSAT formula with soft clauses* $\mathcal{S}$ *and hard clauses* $\mathcal{H}$
2 $K \leftarrow \emptyset$
3 **while** true **do**
4     $h \leftarrow \texttt{MinimumHS}(K)$
5     $(st, \mathcal{C}, \mathcal{A}) \leftarrow \texttt{SAT}(\mathcal{H} \cup (\mathcal{S} \setminus h))$
6   **if** st **then return** $|h|, \mathcal{A}$
7   **else** $K \leftarrow K \cup \{\mathcal{C} \cap \mathcal{S}\}$

new guesses hitting all the unsatisfiable cores thus discovered (based on the Hitting Sets Duality theorem [64]).

The minimum hitting set MaxSAT algorithm used in this work is referred to as *basic MaxHS* [27]. Its setup is shown in Algorithm 2. The algorithm maintains a set K containing the "soft parts" of the unsatisfiable cores found so far; more precisely, each unsatisfiable core is intersected with the set of soft clauses and added to K. In each iteration, a minimum size hitting set (MHS) h of the set K is computed (line 4). Note that h, like the members of K, is a set of soft clauses. The algorithm then checks if h is an MCS of the formula $\mathcal{F}$ by testing whether $\mathcal{H} \cup (\mathcal{S} \setminus h)$ is satisfiable. (This is done by the the SAT solver[5] call in line 5). If h is an MCS of the formula $\mathcal{F}$, then the algorithm (in line 6) returns as a MaxSAT solution, the size $|h|$ of the hitting set, together with the truth assignment $\mathcal{A}$ provided by the SAT solver. Otherwise, a new unsatisfiable core $\mathcal{C}$ is returned by the SAT solver. In this case, $\mathcal{C}$ is intersected with the set of soft clauses and added to K (line 7), and the algorithm proceeds.

Similarly to the core-guided MaxSAT algorithm, the basic MaxHS is not a traditional proof system with inferences, but nonetheless can be viewed as an abstract proof system. Theorems 18, 19, 23 and 26 give upper bounds on the run time of basic MaxHS by stating it "is able to conclude in polynomial time" that the input is unsatisfiable. What this means is that there is a choice of actions for the the SAT solver and the minimum hitting set algorithm which which lead basic MaxHS to halt within polynomial time. For the proofs of our theorems it is permitted that the SAT solver and minimum hitting set algorithm are customized to the specific family of combinatorial principles under consideration. Of course, for practical applications, we are much more interested in the behavior of basic MaxHS with coupled with a generic SAT solver and minimum hitting set solver; for this, again see the experiments in Section 5.

[5] As before, any SAT solver can be considered, as long as the SAT solver provides a satisfying truth assignment if the input formula is satisfiable, and an unsatisfiable core if the input formula is unsatisfiable.

2.3. Combinatorial Principles

The present paper uses (unweighted) dual-rail encodings of several combinatorial principles.

2.3.1. Pigeonhole Principle and Doubled Pigeonhole Principle

The *Pigeonhole Principle* states that if $m+1$ pigeons are mapped to m holes then some hole contains at least two pigeons. This is encoded with the following clauses PHP_m^{m+1}:

$$\begin{array}{ll} \bigvee_{j=1}^{m} x_{i,j} & \text{for } i \in [m+1] \\ \overline{x_{i,j}} \vee \overline{x_{k,j}} & \text{for distinct } i,k \in [m+1] \text{ and } j \in [m]. \end{array}$$

The variable $x_{i,j}$ means that pigeon i goes to hole j.

The second combinatorial principle is the *Doubled Pigeonhole Principle*, also called the "Two Pigeons Per Hole Principle", which states that if $2m+1$ pigeons are mapped to m holes then some hole contains at least three pigeons [14]. This is encoded with the following clauses $2\mathrm{PHP}_m^{2m+1}$:

$$\begin{array}{ll} \bigvee_{j=1}^{m} x_{i,j} & \text{for } i \in [2m+1] \\ \overline{x_{i,j}} \vee \overline{x_{k,j}} \vee \overline{x_{\ell,j}} & \text{for distinct } i,k,\ell \in [2m+1] \text{ and } j \in [m]. \end{array}$$

Note that the pigeonhole principle can can be viewed as a special case of the doubled pigeonhole principle; namely, if the first m pigeons are restricted to map sequentially to the first m holes (by setting $x_{i,i}$ to true for $i \in [m]$), then the remaining pigeons provide an instance of the pigeonhole principle. The pigeonhole principle and the doubled pigeonhole principle can be generalized to any number $\ell m + 1$ of pigeons. Such principle would express the fact that if $\ell m+1$ pigeons are mapped to m holes then some hole contains at least $\ell + 1$ pigeons.

2.3.2. Mutilated Chessboard Principle

Given an even number n, consider an $n \times n$ chessboard where two diagonal positions $(1,1)$ and (n,n) are removed (i.e. of the same color). The principle says that one cannot cover the mutilated chessboard by domino tiles. We can define a graph G_n from the chessboard, by considering the positions of the board to be the nodes of the graph; and for two positions u and v of the chessboard, (u,v) (or equivalently (v,u)) is an edge of $E(G_n)$, if u and v are adjacent (vertically or horizontally) on the board. The boolean encoding considers variables $x_{u,v}$ for edges (u,v) of G_n. $x_{u,v}$ has value true if and only if a domino tile is placed on top of positions u and v. We will identify $x_{u,v}$ with $x_{v,u}$. The following is the set of clauses:

$$\begin{array}{ll} \bigvee_{v,(u,v)\in E(G_n)} x_{u,v} & \text{for } u \in G_n \\ \overline{x_{u,v}} \vee \overline{x_{u,w}} & \text{for } u,v,w \in G_n \text{ s.t. } (u,v),(u,w) \in E(G_n), v \neq w. \end{array}$$

The number of domino pieces that can be placed on the board horizontally is $2(n-2)+(n-1)(n-2) = (n-2)(n+1)$. The number of domino pieces that can be placed vertically is the same, so the total number of variables is $2(n-2)(n+1) = 2n^2 - 2n - 4$.

Resolution lower bounds for the chessboard principle were proven in [2].

2.3.3. Parity Principle

The *Parity Principle*, expresses a kind of mod 2 counting, which states that no graph on m odd nodes consists of a complete perfect matching [1, 9, 11].

The propositional version of the parity principle, uses $\binom{m}{2}$ variables $x_{i,j}$, where $i \neq j$ and $x_{i,j}$ is identified with $x_{j,i}$. The intuitive meaning of $x_{i,j}$ is that there is an edge between vertex i and vertex j. The parity principle has the following sets of clauses:

$$\bigvee_{j \neq i} x_{i,j} \qquad \text{for } i \in [m]$$

$$\overline{x_{i,j}} \vee \overline{x_{k,j}} \qquad \text{for } i, j, k \text{ distinct members of } [m].$$

These clauses state that each vertex has degree one.

2.4. AC^0-Frege and Cutting Planes Proof Systems

To be able to compare dual-rail MaxSAT with resolution, AC^0-Frege and Cutting Planes, we need the following terminology. Proof length is measured in terms of the total number of symbols appearing in the proof. A proof system $\mathcal{P}$ is said to *simulate* another proof system $\mathcal{Q}$ provided that there is a polynomial $p(n)$ so that any $\mathcal{Q}$-proof of a formula of size N can be transformed (by a polynomial time construction) into a $\mathcal{P}$-proof of the same formula of size $\leq p(N)$. For more information on proof complexity, see e.g. the surveys [20, 63].

A *Frege* system is a textbook-style proof system, usually defined to have modus ponens as its only rule of inference [26]. For convenience in defining the depth of formulas, we can treat an implication $A \to B$ as being an abbreviation for $\neg A \vee B$. The depth of propositional formula is measured in terms of alternations: assume a formula φ uses only the connectives $\vee$, $\wedge$ and $\neg$. Using deMorgan's rules, there is a canonical transformation of φ into a formula φ' in "negation normal form", i.e., with negations applied only to variables. Viewing φ' as a tree, the *depth* of φ is the maximum number of blocks of adjacent $\vee$'s and adjacent $\wedge$'s along any branch in the tree φ'. Notice that this definition is not the standard definition of the depth of a tree. A depth d Frege proof is a Frege proof in which every formula has depth $\leq d$. An AC^0-Frege proof is a proof with a constant upper bound on the depth of formulas appearing in the proof.

The cutting planes system is a pseudo-Boolean propositional proof system. It uses variables x_i which take on 0/1 values, indicating Boolean values *False* and *True*. The lines of a cutting planes proof are inequalities of the form

$$a_1x_1 + a_2x_2 + \cdots + a_nx_n \geq a_{n+1},$$

where the a_i's are integers. Logical axioms include $x_i \geq 0$ and $-x_i \geq -1$; inference rules include addition, multiplication by a integer, and a division rule. A cutting planes proof refuting a set Γ of clauses has axioms expressing the truth of the clauses in Γ, and has $0 \geq 1$ as its last line. The cutting planes system CP uses integers a_i written in binary; the system CP* uses the integers a_i written in unary notation. The *size* of a CP or CP* proof is the total number of symbols in the proof, including the bits used for representing the values of the coefficients a_i. For more on cutting planes, see e.g. [62, 21].

3. Dual-Rail Encoding and New Proof Systems for Satisfiability

3.1. Dual-rail MaxSAT.

We now define the *dual-rail MaxSAT* system [38] for refuting a set of clauses Γ. The dual-rail MaxSAT system is based on MaxSAT solving, but as already mentioned is strictly stronger than resolution.

Let Γ be a set of clauses (viewed as hard clauses) over the variables $\{x_1, \ldots, x_s\}$. The dual-rail encoding Γ^{dr} of Γ, uses $2s$ variables $n_1, \ldots, n_s$ and $p_1, \ldots, p_s$ in place of the s variables x_i. The intent is that p_i is true if x_i is true, and that n_i is true if x_i is false. The dual-rail encoding C^{dr} of a clause C is defined by replacing each (unnegated) variable x_i in C with $\overline{n_i}$, and replacing each (negated) literal $\overline{x_i}$ in C with $\overline{p_i}$. For example, if C is $\{x_1, \overline{x_3}, x_4\}$, then C^{dr} is $\{\overline{n_1}, \overline{p_3}, \overline{n_4}\}$. Note that every literal in C^{dr} is negated.

The dual-rail encoding Γ^{dr} of Γ contains the following clauses: (1) the hard clause C^{dr} for each $C \in \Gamma$; (2) the hard clauses $(\overline{p_i} \vee \overline{n_i})$ for $1 \leq i \leq s$; and (3) the soft clauses (p_i) and (n_i) for $1 \leq i \leq s$. Γ^{dr} is equivalently represented as a set of weighted clauses:

$$\begin{array}{cl} (C^{\mathrm{dr}}, \top) & \text{for } C \in \Gamma \\ (\overline{p_i} \vee \overline{n_i}, \top) & \text{for } 1 \leq i \leq s \\ (p_i, 1) & \text{for } 1 \leq i \leq s \\ (n_i, 1) & \text{for } 1 \leq i \leq s. \end{array}$$

Following [38], the clauses $\overline{p_i} \vee \overline{n_i}$ are called the $\mathcal{P}$ *clauses*.

Note that all clauses of Γ^{dr} are Horn: the hard clauses contain only negated literals and the soft clauses are unit clauses. The transformation proposed can be related to the well-known dual-rail encoding, used in different settings [19, 44, 66, 40, 61].

A dual-rail MaxSAT refutation of Γ is defined as a MaxSAT derivation of a multiset of clauses containing $\geq s{+}1$ many copies of the empty clause $\perp$ from Γ^{dr}. This is based on the fact that Γ is satisfiable if and only if there is a truth assignment τ which makes all the hard clauses of Γ^{dr} true, and only s of the soft clauses false [38]. Let us justify this.

Lemma 3. *Given Γ and the corresponding dual-rail encoding Γ^{dr}, there can be no more than s satisfied soft clauses.*

Proof. There is no assignment that satisfies all hard clauses $\overline{p_i} \vee \overline{n_i}$ with $n_i = 1$ and $p_i = 1$ for some i. $\square$

Lemma 4. *If Γ is satisfiable, then there exists an assignment that satisfies the hard clauses and s soft clauses of Γ^{dr}.*

Proof. Suppose ν satisfies Γ. Create an assignment ν' to the n_i and p_i variables the following way: For each x_i, if $\nu(x_i) = 1$, then set $p_i = 1$ and $n_i = 0$; otherwise set $n_i = 1$ and $p_i = 0$. Thus, there will be s soft clauses satisfied. Also, it is clear that all the hard clauses $\overline{p_i} \vee \overline{n_i}$ are satisfied. For each clause $C \in \Gamma$, pick a literal l_k assigned value 1 by ν. If $l_k = x_k$, then C^{dr} contains literal $\overline{n_k}$, and it is satisfied by ν'. If $l_k = \overline{x_k}$, then C^{dr} contains literal $\overline{p_k}$, and so it is satisfied by ν'. $\square$

Lemma 5. *Let ν' be an assignment that satisfies all the hard clauses in Γ^{dr} and s soft clauses. Then there exists an assignment ν that satisfies Γ.*

Proof. Because ν' satisfies the hard clauses $\overline{p_i} \vee \overline{n_i}$, and it satisfies s many soft clauses, for each 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 $C' \in \Gamma^{\mathrm{dr}}$, let l_k be a literal in C' assigned value 1. If $l_k = \overline{n_k}$, then x_k is a literal in $C \in \Gamma$ and since $\nu(x_i) = 1$, then the clause C is satisfied. Otherwise, if $l_k = \overline{p_k}$, then $\overline{x_k}$ is a literal in C and since $\nu(x_i) = 0$, then the clause C is satisfied. $\square$

Lemmas 3, 4 and 5 yield the following.

Theorem 6. ([38]) *Γ is satisfiable if and only if there exists an assignment that satisfies all the hard clauses of Γ^{dr} and s soft clauses.*

As a consequence of Theorems 1, 2 and 6, the propositional proof systems for satisfiability of CNF formulas consisting of translating them to the dual-rail encoding and then using either the MaxSAT resolution rule, or a core-guided algorithm, or a minimum hitting set algorithm, are sound and complete proof systems.

Theorem 7. *Let Γ be a CNF formula with s variables.*

Soundness: if there is a MaxSAT derivation of a set of $s+1$ empty clauses from Γ^{dr}, Γ is unsatisfiable.

Completeness: if Γ is unsatisfiable, then there is a MaxSAT derivation of $s+1$ empty clauses from Γ^{dr}.

An example. We present a very simple example of a dual-rail MaxSAT resolution refutation which refutes the three clauses $\overline{x_1} \vee x_2$, x_1 and $\overline{x_2}$. This is almost the simplest possible example, but still reveals interesting aspects. The dual-rail encoding has the five hard clauses

$$\overline{p_1} \vee \overline{n_2} \qquad \overline{n_1} \qquad \overline{p_2} \qquad \overline{p_1} \vee \overline{n_1} \qquad \overline{p_2} \vee \overline{n_2},$$

plus the four soft unit clauses

$$p_1 \qquad n_1 \qquad p_2 \qquad n_2.$$

Since there are two variables, a dual-rail MaxSAT refutation must derive a multiset containing three copies of the empty clause $\bot$. The following four inferences will be used to form the refutation (the weights 1 and $\top$ are used for soft and hard clauses, respectively):

$$\begin{array}{c}(\overline{n_1},\top)\\(n_1,1)\\\hline(\bot,1)\\(\overline{n_1},\top)\end{array}\qquad\qquad\begin{array}{c}(\overline{p_2},\top)\\(p_2,1)\\\hline(\bot,1)\\(\overline{p_2},\top)\end{array}$$

$$\begin{array}{c}(p_1,1)\\(\overline{p_1}\vee\overline{n_2},\top)\\\hline(\overline{n_2},1)\\(p_1\vee n_2,1)\\(\overline{p_1}\vee\overline{n_2},\top)\end{array}\qquad\qquad\begin{array}{c}(\overline{n_2},1)\\(n_2,1)\\\hline(\bot,1)\end{array}$$

We describe a dual-rail MaxSAT refutation using these four inferences; its "lines" consist of five multisets of clauses $\Gamma_0, \Gamma_1, \Gamma_2, \Gamma_3, \Gamma_4$. The initial multiset Γ_0 contains the nine clauses given above. Since the set of hard clauses never changes, each Γ_i has the form $\Gamma_i = S_i \cup H$ where H is the set of five hard clauses above, and S_i is a multiset of soft (weight 1) clauses. Namely,

$$\begin{array}{rcl}S_0 &=& \{p_1,\ n_1,\ p_2,\ n_2\}\\ S_1 &=& \{p_1,\ \ \bot,\ p_2,\ n_2\}\\ S_2 &=& \{p_1,\ \ \bot,\ \ \bot,\ n_2\}\\ S_3 &=& \{\overline{n_2},\ p_1\vee n_2,\ \ \bot,\ \ \bot,\ n_2\}\\ S_4 &=& \{\bot,\ p_1\vee n_2,\ \ \bot,\ \ \bot\}.\end{array}$$

Here S_0 is the four initial soft clauses; and S_4 contains three copies of $\bot$ as needed for a valid dual-rail MaxSAT refutation.

There is a couple interesting observations about even such a simple derivation. First, it splits neatly into three independent parts: one that uses n_1 and $\overline{n_1}$ to derive $\bot$, one that uses p_2 and $\overline{p_2}$ to derive $\bot$, and one that uses the other clauses to derive a third copy of $\bot$. This splitting is part of the reason that dual-rail MaxSAT can give simpler proofs than ordinary resolution, say for PHP. Second, there is an extra soft clause $p_1 \vee n_2$ that is derived but not used; this is a common feature of dual-rail MaxSAT refutations.

We can also define a weighted version of the dual-rail encoding. Given a set of finite positive weights $w_1, \ldots, w_s$, the *weighted dual-rail encoding* Γ^{wdr} of Γ

is defined as the set of clauses

$$\begin{array}{ll} (C^{\mathrm{dr}}, \top) & \text{for } C \in \Gamma \\ (\overline{p_i} \vee \overline{n_i}, \top) & \text{for } 1 \le i \le s \\ (p_i, w_i) & \text{for } 1 \le i \le s \\ (n_i, w_i) & \text{for } 1 \le i \le s. \end{array}$$

Let $k = \sum_i w_i$, a *weighted dual-rail MaxSAT* refutation is a MaxSAT derivation of a set of empty clauses with total weight at least $k{+}1$, from Γ^{wdr}.

When the weights w_i are all small (i.e., polynomially bounded), then it is convenient to work with the *multiple dual-rail MaxSAT* system. In this system, instead of including the clauses (p_i, w_i) and (n_i, w_i) with weights w_i possibly larger than 1, we introduce w_i many copies of the soft clauses p_i and n_i, each of weight 1. The resulting set of clauses is denoted by Γ^{mdr}. Any MaxSAT derivation from Γ^{mdr} is readily converted into a MaxSAT derivation from Γ^{wdr}. Conversely, if there is polynomial upper bound on the values w_i, then the size of a MaxSAT derivation from Γ^{wdr} can be converted into a MaxSAT derivation from Γ^{mdr} with size only polynomially bigger. This means that the weighted dual-rail MaxSAT system is a strengthening of the multiple dual-rail MaxSAT system. For the present paper, the main advantage of working with the multiple dual-rail MaxSAT system, instead of with the weighted dual-rail MaxSAT system is that it simplifies notation for the proof of Theorem 29 by letting us discuss soft and hard clauses without explicitly writing their weights.

3.2. Dual-Rail Encodings of Various Principles.

The present paper uses (unweighted) dual-rail encodings of several combinatorial principles already defined in Section 2.3. These are defined following the definition of Γ^{dr} above.

The dual-rail encoding, $(\mathrm{PHP}^{m+1}_m)^{\mathrm{dr}}$, of PHP^{m+1}_m contains the hard clauses

$$\begin{array}{ll} \bigvee_{j=1}^m \overline{n_{i,j}} & \text{for } i \in [m{+}1] \\[1ex] \overline{p_{i,j}} \vee \overline{p_{k,j}} & \text{for } j \in [m] \text{ and distinct } i, k \in [m{+}1]. \end{array}$$

It also contains the hard $\mathcal{P}$ clauses $\overline{p_{i,j}} \vee \overline{n_{i,j}}$. The soft clauses are the unit clauses $n_{i,j}$ and $p_{i,j}$ for all $i{\in}[m{+}1]$ and $j{\in}[m]$. There are $(m{+}1)m$ positive variables $p_{i,j}$ and likewise $(m{+}1)m$ negative variables $n_{i,j}$, for a total of $2(m{+}1)m$ many soft clauses. A dual-rail MaxSAT refutation for PHP^{m+1}_m must produce $(m{+}1)m{+}1$ many empty clauses ($\bot$'s) from $(\mathrm{PHP}^{m+1}_m)^{\mathrm{dr}}$. This is because by Theorem 6, PHP^{m+1}_m is satisfiable if and only if there exists an assignment that satisfies all the hard clauses of $(\mathrm{PHP}^{m+1}_m)^{\mathrm{dr}}$ and $m(m+1)$ soft clauses from $(\mathrm{PHP}^{m+1}_m)^{\mathrm{dr}}$.

The second combinatorial principle for which we will need the dual-rail encoding, is the *Doubled Pigeonhole Principle*, The dual-rail encoding, $(2\mathrm{PHP}^{2m+1}_m)^{\mathrm{dr}}$, of $2\mathrm{PHP}^{2m+1}_m$ contains the hard clauses

$$\begin{array}{ll} \bigvee_{j=1}^m \overline{n_{i,j}} & \text{for } i \in [2m{+}1] \\[1ex] \overline{p_{i,j}} \vee \overline{p_{k,j}} \vee \overline{p_{\ell,j}} & \text{for } j \in [m] \text{ and distinct } i, k, \ell \in [2m{+}1]. \end{array}$$

It also contains the hard $\mathcal{P}$ clauses $\overline{p_{i,j}} \vee \overline{n_{i,j}}$. The soft clauses are the unit clauses $n_{i,j}$ and $p_{i,j}$ for all $i \in [2m+1]$ and $j \in [m]$. There are $(2m+1)m$ positive variables $p_{i,j}$ and likewise $(2m+1)m$ negative variables $n_{i,j}$, for a total of $2(2m+1)m$ many soft clauses. A dual-rail MaxSAT refutation for $2\mathrm{PHP}_m^{2m+1}$ must produce $(2m+1)m + 1$ many empty clauses ($\perp$'s) from $(2\mathrm{PHP}_m^{2m+1})^{\mathrm{dr}}$.

The third combinatorial principle for which we will need the dual-rail encoding, is the *Mutilated Chessboard Principle* The dual-rail encoding contains the hard clauses

$$\begin{array}{ll} \bigvee_{v,(u,v) \in E(G_n)} \overline{n_{u,v}} & \text{for } u \in G_n \\ \overline{p_{u,v}} \vee \overline{p_{u,w}} & \text{for } u, v, w \in G_n \text{ s.t. } (u,v), (u,w) \in E(G_n),\ v \neq w. \end{array}$$

It also contains the hard $\mathcal{P}$ clauses $\overline{p_{u,v}} \vee \overline{n_{u,v}}$. As before the soft clauses are the unit clauses $n_{i,j}$ and $p_{i,j}$ for any i and j adjacent positions. There are $2n^2 - 2n - 4$ positive variables $p_{i,j}$ and likewise $2n^2 - 2n - 4$ negative variables $n_{i,j}$, for a total of $2(2n^2 - 2n - 4)$ many soft clauses. A dual-rail MaxSAT refutation for the mutilated chessboard principle must produce at least $2n^2 - 2n - 3$ many empty clauses ($\perp$'s).

We will also consider the *Parity Principle*. The variables of the dual-rail encoding of the parity principle are $\{n_{i,j} : 1 \leq i < j \leq m\}$ and $\{p_{i,j} : 1 \leq i < j \leq m\}$. The soft clauses of the dual-rail encoding are the unit clauses $n_{i,j}$ and $p_{i,j}$ for any $1 \leq i < j \leq m\}$. The hard clauses are:

$$\begin{array}{ll} \bigvee_{j \neq i} \overline{n_{i,j}} & \text{for } i \in [m] \\ \overline{p_{i,j}} \vee \overline{p_{k,j}} & \text{for } i, j, k \text{ distinct members of } [m], \end{array}$$

and the $\mathcal{P}$ clauses $\overline{p_{i,j}} \vee \overline{n_{i,j}}$.

The dual-railing encodings above include $\mathcal{P}$ clauses, in keeping with the earlier definition of Γ^{dr}. However, it is sometimes convenient to omit the $\mathcal{P}$ clauses; indeed the results shown in Figure 1 all still hold if the $\mathcal{P}$ clauses are omitted. Furthermore, as reported in Section 8, some solvers obtain better results when the $\mathcal{P}$ clauses are omitted.

4. Upper Bounds

This section shows that the dual-rail encoding enables MaxSAT resolution, core-guided MaxSAT algorithms, and MaxHS-like algorithms to prove in polynomial time the unsatisfiability of the CNF encodings of both the pigeonhole principle and the doubled pigeonhole principle. The results in this section should be contrasted with the resolution exponential lower bounds for the pigeonhole principle, and 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} (not dual-rail). Additionally, we present upper bound results for dual-rail MaxSAT resolution refutations of the mutilated chessboard principle.

Finally, we prove an upper bound result for the dual-rail encoding of the parity principle using MaxHS-like algorithms. This is an interesting upper bound, since Section 6 will prove exponential size lower bounds for the dual-rail encoding of the parity principle using MaxSAT resolution. As a consequence, this principle shows that dual-rail MaxSAT resolution cannot simulate dual-rail MaxHS-like algorithms. Whether the dual-rail encoding of the parity principle has polynomial size proofs using core-guided MaxSAT algorithms remains an open problem.

Let us remark that none of the upper bounds that we prove in this section need to use the $\mathcal{P}$ clauses to show their unsatisfiability. Therefore, unless otherwise stated, $\mathcal{P}$ clauses will be disregarded.

4.1. Polynomial Bounds with MaxSAT Resolution

This section develops upper bounds for the propositional encodings of the pigeonhole principle, the doubled pigeonhole principle, and the mutilated chessboard problem when using MaxSAT resolution with the dual-rail encoding. All these upper bounds benefit from the fact that the dual-rail initial clauses do not mix $n_{i,j}$ and $p_{i,j}$ variables. This allows the MaxSAT refutations to split into two independent parts: one part derives a number of $\bot$'s from the literals $n_{i,j}$; the other part derives the remaining needed $\bot$'s from the clauses that use $p_{i,j}$.

4.1.1. Pigeonhole Principle

Theorem 8. *There are polynomial size MaxSAT resolution refutations of the dual-rail encoding of the PHP_m^{m+1} clauses.*

Proof. To show unsatisfiability of the pigeonhole principle under the dual-rail encoding, we need to produce $m(m+1)+1$ 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 the pigeonhole principle is unsatisfiable.

The MaxSAT refutation first derives $m+1$ empty clauses $\bot$, one for each pigeon $i \in [m+1]$, by resolving the hard clause $\bigvee_{j=1}^{m} \overline{n_{i,j}}$ against the soft unit clauses $\{n_{i,j}\}$ to obtain the clause $\bot$. These inferences derive other clauses as well, but they are not needed for the refutation, so we just ignore them in what follows. For a fixed pigeon i, consider the hard clause $\bigvee_{j=1}^{m} \overline{n_{i,j}}$ and the soft clause $\{n_{i,1}\}$. Resolving these two clauses results in two soft clauses $\bigvee_{j=2}^{m} \overline{n_{i,j}}$ and $n_{i,1} \vee \overline{\bigvee_{j=2}^{m} \overline{n_{i,j}}}$, together with the (original) hard clause $\bigvee_{j=1}^{m} \overline{n_{i,j}}$. Now the obtained soft clause $\bigvee_{j=2}^{m} \overline{n_{i,j}}$ can be resolved with the soft clause $\{n_{i,2}\}$, and ignoring the other clauses (obtained in the first resolution step). The new clauses from the second resolution step are the soft clauses $\bigvee_{j=3}^{m} \overline{n_{i,j}}$ and $n_{i,2} \vee \overline{\bigvee_{j=3}^{m} \overline{n_{i,j}}}$. The last resolution step is repeated in a similar way with the soft unit clauses $\{n_{i,3}\}, \ldots, \{n_{i,m}\}$, until the empty clause $\bot$ is obtained. The following shows the MaxSAT resolution steps described.

$$
\begin{array}{c}
(\bigvee_{j=1}^{m} \overline{n_{i,j}}, \top) \qquad\qquad (n_{i,1}, 1) \\ \hline
(\bigvee_{j=2}^{m} \overline{n_{i,j}}, 1) \;\; (n_{i,1} \vee \overline{\bigvee_{j=2}^{m} \overline{n_{i,j}}}, 1) \qquad (n_{i,2}, 1) \\ \hline
(\bigvee_{j=3}^{m} \overline{n_{i,j}}, 1) \;\; (n_{i,2} \vee \overline{\bigvee_{j=3}^{m} \overline{n_{i,j}}}, 1) \;\; (n_{i,1} \vee \overline{\bigvee_{j=2}^{m} \overline{n_{i,j}}}, 1) \qquad (n_{i,3}, 1) \\ \hline
\vdots \qquad\qquad\qquad (n_{i,m}, 1) \\ \hline
(\bot, 1) \cdots
\end{array}
$$

Now to derive empty clauses from the hole clauses, we fix a hole j. We inductively describe the construction of the MaxSAT derivation of m empty clauses from the clauses involving literals $p_{i,j}$. The construction will be repeated (independently) for each $j \in [m]$. The general idea is to derive $I-1$ many $\bot$'s from the first I pigeons, namely using only the literals $p_{i,j}$ for $i \leq I$. The construction proceeds in stages, one for each value $I = 2, 3, \ldots, m{+}1$.

The base case is stage $I = 2$. We start by resolving the soft unit clause $p_{1,j}$ against the hard clause $\overline{p_{1,j}} \vee \overline{p_{2,j}}$ to obtain the soft clauses $\overline{p_{2,j}}$ and $p_{1,j} \vee p_{2,j}$. Next we resolve $\overline{p_{2,j}}$ against the soft clause $p_{2,j}$, obtaining $\bot$. Again, other clauses are obtained, but we can ignore them. What is important for us at this point is that we have obtained $p_{1,j} \vee p_{2,j}$ and $\bot$. The following shows the previous MaxSAT resolution steps.

$$
\begin{array}{c}
(\overline{p_{1,j}} \vee \overline{p_{2,j}}, \top) \qquad\qquad (p_{i,1}, 1) \\ \hline
(\overline{p_{2,j}}, 1) \;\; (p_{1,j} \vee p_{2,j}, 1) \;\; (\overline{p_{1,j}} \vee \overline{p_{2,j}}, \top) \qquad (p_{2,j}, 1) \\ \hline
(\bot, 1) \;\; (p_{1,j} \vee p_{2,j}, 1) \;\; (\overline{p_{1,j}} \vee \overline{p_{2,j}}, \top)
\end{array}
$$

We now present the construction for stage $I > 2$. The induction hypothesis is that the previous stage has derived the soft clause $p_{1,j} \vee \cdots \vee p_{I-1,j}$ and $I-2$ clauses $\bot$. Stage I will use the hard clauses $\{\overline{p_{1,j}} \vee \overline{p_{I,j}}, \ldots, \overline{p_{I-1,j}} \vee \overline{p_{I,j}}\}$, the soft clause $p_{I,j}$ and the soft clause $p_{1,j} \vee \cdots \vee p_{I-1,j}$ derived in the previous step. To start Stage I, we resolve $p_{1,j} \vee \cdots \vee p_{I-1,j}$ against $\overline{p_{1,j}} \vee \overline{p_{I,j}}$, obtaining among other soft clauses $\overline{p_{I,j}} \vee p_{2,j} \vee \cdots \vee p_{I-1,j}$ and $p_{1,j} \vee \cdots \vee p_{I,j}$. The latter clause is saved for use in Stage $I{+}1$. Stage I then iteratively resolves $\overline{p_{I,j}} \vee p_{s,j} \vee \cdots \vee p_{I-1,j}$ with the hard clause $\overline{p_{s,j}} \vee \overline{p_{I,j}}$, obtaining $\overline{p_{I,j}} \vee p_{s+1,j} \vee \cdots \vee p_{I-1,j}$, for $s = 3, \ldots, I{-}1$. Eventually we are left with $\overline{p_{I,j}}$, that we resolve against $p_{I,j}$, obtaining $\bot$. Thus, after finishing Stage $I = m+1$, we have obtained m clauses $\bot$.

$$
\begin{array}{c}
(\bigvee_{i=1}^{I-1} p_{i,j}, 1) \qquad\qquad (\overline{p_{1,j}} \vee \overline{p_{I,j}}, \top) \\ \hline
(\bigvee_{i=2}^{I-1} p_{i,j} \vee \overline{p_{I,j}}, 1) \; (\bigvee_{i=1}^{I} p_{i,j}, 1) \; (\overline{p_{1,j}} \vee \overline{p_{I,j}}, \top) \quad (\overline{p_{2,j}} \vee \overline{p_{I,j}}, \top) \\ \hline
(\bigvee_{i=3}^{I-1} p_{i,j} \vee \overline{p_{I,j}}, 1) \; (\bigvee_{i=1}^{I} p_{i,j}, 1) \; (\overline{p_{1,j}} \vee \overline{p_{I,j}}, \top) \; (\overline{p_{2,j}} \vee \overline{p_{I,j}}, \top) \quad (\overline{p_{3,j}} \vee \overline{p_{I,j}}, \top) \\ \hline
\vdots \\
(\overline{p_{I,j}}, 1)(\bigvee_{i=1}^{I} p_{i,j}, 1) \cdots \qquad\qquad (p_{I,j}, 1) \\ \hline
(\bot, 1)(\bigvee_{i=1}^{I} p_{i,j}, 1) \cdots
\end{array}
$$

In summary, working with the $n_{i,j}$ soft clauses we have obtained one $\bot$ per pigeon clause $\bigvee_{j=1}^{m} \overline{n_{i,j}}$, making a total of $m+1$ clauses $\bot$. On the other hand, for every pigeon j, we obtain m clauses $\bot$, making a total of $m \cdot m$. Adding up these numbers we have $m + 1 + m \cdot m = m(m+1) + 1$, and by Theorem 6 we have proved that the pigeonhole principle is unsatisfiable. By inspection we can see that the proof is linear in the size of the dual-rail encoded principle. □

4.1.2. Mutilated Chessboard

The upper bound for the dual-rail encoding of the mutilated chessboard problem using MaxSAT resolution follows the argument given for the pigeonhole principle. For the sake of clarity in the explanation, we will consider a chessboard with $n \times n$ positions, where two opposite positions will be taken away, the $(1,1)$ and the (n,n). First we will number all the positions, zigzagging through the board. We start with position $(1,1)$ and we number all of them going rightward until reaching position $(1,n)$. After the position $(1,n)$ we have the position $(2,n)$, and decrease until the position $(2,1)$. Then we continue with $(3,1)$ in ascending order for the third row. Following this ordering, we can number the positions from 1 to n^2, and the skipped positions correspond to numbers 1 and $n^2 - n + 1$. This way of numbering the positions of the board has the explanatory advantage of having every even position being surrounded by odd positions, and vice versa. At this point we can consider the mutilated chessboard problem as a restricted pigeonhole principle, where the pigeons are the even numbered positions ($\frac{n^2}{2}$ many of them), and the holes are the odd number positions ($\frac{n^2}{2} - 2$ many). In this restricted pigeonhole, every pigeon can go to 2 or 3 or 4 holes, and every hole can receive either 3 or 4 pigeons. Using this intuition, we can proceed with the proof.

Theorem 9. *There are polynomial size MaxSAT resolution refutations of the dual-rail encoding of the mutilated chessboard clauses.*

Proof. The MaxSAT refutation first derives $\frac{n^2}{2}$ clauses $\bot$, one for each clause on the variables $n_{i,j}$ focused on the even positions. This is done by resolving the hard clause $\bigvee_{j,(i,j)\in E(G_n)} \overline{n_{i,j}}$ for i in an even position, against the soft unit clauses $\{n_{i,j}\}$, to obtain the clause $\bot$. Notice that these clauses do not have variables in common, and we can ignore the clauses on the variables $n_{i,j}$ focused on the odd positions. These inferences derive other clauses as well, but we just ignore them.

In the case of the $p_{i,j}$ clauses, we will ignore the clauses focused on the even numbered positions of the board. Every odd position will generate 2 $\bot$'s if it belongs to the boundary, or it will generate 3 $\bot$'s if it belongs to the interior of the board. The argument is identical to the one used for the $p_{i,j}$ variables in the pigeonhole principle, given that the $p_{i,j}$ clauses on the odd positions do not share any variables. There are $4(\frac{n}{2} - 1) = 2n - 4$ odd positions on the boundaries, and $\frac{(n-2)(n-2)}{2} = \frac{n^2}{2} - 2n + 2$ odd positions in the interior of the board. Therefore the number of $\bot$'s that you get from the $p_{i,j}$ clauses is $2(2n-4) + 3(\frac{n^2}{2} - 2n + 2) = \frac{3}{2}n^2 - 2n - 2$.

Summing up the $\perp$ from the two types of variables and clauses we get $\frac{n^2}{2} + \frac{3}{2}n^2 - 2n - 2 = 2n^2 - 2n - 2$, which is two more than the number of variables $2n^2 - 2n - 4$, and therefore we prove that the mutilated chessboard is unsatisfiable. ☐

4.1.3. Doubled Pigeonhole Principle

This section discusses the "doubled" pigeonhole principle which states that if $2m + 1$ pigeons are mapped to m holes then some hole contains at least three pigeons [14]. These principles were defined in Section 2.

Theorem 10. *There are polynomial size MaxSAT resolution refutations of the dual-rail encoding of the* $2PHP^{2m+1}_m$ *clauses.*

Proof. The MaxSAT refutation first derives $2m+1$ clauses $\perp$, one for each pigeon $i \in [2m+1]$, by resolving the hard clause $\bigvee_{j=1}^m \overline{n_{i,j}}$ against the soft unit clauses $\{n_{i,j}\}$ to obtain the clause $\perp$. These inferences derive other clauses as well, but they are not needed for the refutation, so we just ignore them. The remainder of the MaxSAT refutation is more complex and derives $2m-1$ empty clauses for each hole $j \in [m]$. This gives a total of $(2m-1)m$ additional $\perp$'s and, since $2m+1+(2m-1)m$ is equal to $(2m+1)m+1$, suffices to complete the MaxSAT refutation.

Fix a hole j. We describe the construction of the MaxSAT derivation of $2m - 1$ empty clauses from the clauses involving literals $p_{i,j}$. The construction will be repeated (independently) for each $j \in [m]$. The general idea is to inductively derive $I - 2$ many $\perp$'s from the first I pigeons, namely using only the literals $p_{i,j}$ for $i \leq I$.

The construction (for fixed j) proceeds in $2m - 1$ stages, one for each value $I = 3, 4, \ldots, 2m+1$. As described below, each stage will have two phases. The first phase of stage I will start with the hard clause $\overline{p_{1,j}} \vee \overline{p_{I-1,j}} \vee \overline{p_{I,j}}$, and a set of soft clauses (denoted C_i^{I-1} for $1 \leq i < I$) carried over from the previous stage, and will generate soft clauses D_i^I (for $1 \leq i \leq I$) to be used in the second phase, and clauses C_i^I (for $1 \leq i \leq I$) to be carried over to the next stage. The second phase of stage I will use the clauses D_i^I obtained in the first phase, the other hard clauses $\overline{p_{i,j}} \vee \overline{p_{k,j}} \vee \overline{p_{I,j}}$ and the soft unit clause $p_{I,j}$ to derive an empty clause $\perp$.

As a visual aid, the clauses C_i^I will be typeset in a solid box to indicate they are used in the next stage, and the clauses D_i^I will be typeset in a dotted box to indicate they are derived in the first phase and used in the second phase.

The base case is stage $I = 3$. The first phase starts by resolving the soft unit clause $p_{1,j}$ against the hard clause $\overline{p_{1,j}} \vee \overline{p_{2,j}} \vee \overline{p_{3,j}}$ to obtain the soft clauses

$$\overline{p_{2,j}} \vee \overline{p_{3,j}} \qquad \boxed{p_{1,j} \vee p_{2,j}} \qquad p_{1,j} \vee \overline{p_{2,j}} \vee p_{3,j}$$

Recall that the dashed box around the first clause (D_2^3) indicates that it will be used in the second phase of this stage, and the solid box around the second clause (C_3^3) indicates it will be carried forward to the next stage, when $I = 4$. The first phase then resolves the soft unit clause $p_{2,j}$ against the third clause $p_{1,j} \vee \overline{p_{2,j}} \vee p_{3,j}$ to derive the soft clauses C_2^3, D_1^3 and C_1^3:

$$\boxed{p_{1,j} \vee p_{3,j}} \qquad p_{2,j} \vee \overline{p_{3,j}} \qquad \boxed{\overline{p_{1,j}} \vee p_{2,j} \vee p_{3,j}}$$

The second phase of stage $I = 3$ resolves $\overline{p_{2,j}} \vee \overline{p_{3,j}}$ against $p_{2,j} \vee \overline{p_{3,j}}$ to obtain the unit clause $\overline{p_{3,j}}$. This is resolved against the soft initial unit clause $p_{3,j}$ to obtain the desired empty clause $\perp$.

The clauses formed during stage $I = 3$ were:

Clause	Literals		
C_1^3	$\overline{p_{1,j}}$	$p_{2,j}$	$p_{3,j}$
C_2^3	$p_{1,j}$		$p_{3,j}$
C_3^3	$p_{1,j}$	$p_{2,j}$	

Clause	Literals	
D_1^3	$p_{2,j}$	$\overline{p_{3,j}}$
D_2^3	$\overline{p_{2,j}}$	$\overline{p_{3,j}}$

The end result of stage $I = 3$ is the derivation of one $\perp$ and C_1^3, C_2^3, C_3^3.

We now sketch the construction for stage $I > 3$. The induction hypothesis is that the previous stage has derived the following soft clauses:

Clause	Literals					
C_1^{I-1}:	$\overline{p_{1,j}}$	$p_{2,j}$	$\cdots$	$p_{I-3,j}$	$p_{I-2,j}$	$p_{I-1,j}$
C_2^{I-1}:	$p_{1,j}$	$\overline{p_{2,j}}$	$\cdots$	$p_{I-3,j}$	$p_{I-2,j}$	$p_{I-1,j}$
$\vdots$	$\vdots$					
C_{I-3}^{I-1}:	$p_{1,j}$	$p_{2,j}$	$\cdots$	$\overline{p_{I-3,j}}$	$p_{I-2,j}$	$p_{I-1,j}$
C_{I-2}^{I-1}:	$p_{1,j}$	$p_{2,j}$	$\cdots$	$p_{I-3,j}$		$p_{I-1,j}$
C_{I-1}^{I-1}:	$p_{1,j}$	$p_{2,j}$	$\cdots$	$p_{I-3,j}$	$p_{I-2,j}$	

Notice that the clauses only differ in the diagonal, where the literals are negated except in the last two clauses where the literal is missing. The pattern is that C_i^{I-1} contains the literals $p_{i',j}$ for $i' < I$, except that $p_{i,j}$ is negated if $i < I-2$ and is missing otherwise.

As we describe below, the first phase of stage I uses only these clauses C_i^{I-1} and the hard clause $\overline{p_{1,j}} \vee \overline{p_{I-1,j}} \vee \overline{p_{I,j}}$. The clauses C_i^{I-1} are used in the reverse order as listed above, with $i = I-1, \ldots, 1$. The first phase produces the soft clauses C_i^I (again, in the order $i = I$ down to $i = 1$) to be used in the next stage. It also produces clauses D_i^I to be used in the second phase (see Figure 2). It is interesting to note that the overall structure of the first phase is a "linear" proof.

The first phase starts by resolving C_{I-1}^{I-1} against the hard clause $\overline{p_{1,j}} \vee \overline{p_{I-1,j}} \vee \overline{p_{I,j}}$, to obtain the soft clauses

$$p_{2,j} \vee \cdots \vee p_{I-2,j} \vee \overline{p_{I-1,j}} \vee \overline{p_{I,j}}$$
$$\boxed{p_{1,j} \vee \cdots \vee p_{I-2,j} \vee p_{I-1,j}}$$
$$p_{1,j} \vee \cdots \vee p_{I-2,j} \vee \overline{p_{I-1,j}} \vee p_{I,j}$$
$$\cdots$$

The first clause is D^I_{I-1} and will be used in the second phase. The second clause is C^I_I and will be carried forward to the next stage. The third clause will be used immediately. The "$\cdots$" indicates other conclusions of the MaxSAT inference which are not used in the refutation. The third clause is resolved against C^{I-1}_{I-2}, which is $p_{1,j} \vee p_{2,j} \vee \cdots \vee p_{I-3,j} \vee p_{I-1,j}$, yielding soft clauses

$$
\begin{gathered}
\boxed{p_{1,j} \vee \cdots \vee p_{I-3,j} \vee p_{I-2,j} \vee p_{I,j}} \\
p_{1,j} \vee \cdots \vee p_{I-3,j} \vee p_{I-1,j} \vee \overline{p_{I,j}} \\
\boxed{p_{1,j} \vee \cdots \vee p_{I-3,j} \vee \overline{p_{I-2,j}} \vee p_{I-1,j} \vee p_{I,j}} \\
\cdots
\end{gathered}
$$

The first and third clauses are C^I_{I-1} and C^I_{I-2} and are carried forward to the next stage. The middle clause will be used immediately by resolving it against C^{I-1}_{I-3}.

The next steps all follow the same pattern: namely, with $2 \leq i \leq I-2$, the clause

$$p_{1,j} \vee \cdots \vee p_{i-1,j} \vee p_{i+1,j} \vee \cdots \vee p_{I-1,j} \vee \overline{p_{I,j}}$$

has just been derived, and it is resolved against C^{I-1}_{i-1}, which is the clause

$$p_{1,j} \vee \cdots \vee p_{i-2,j} \vee \overline{p_{i-1,j}} \vee p_{i,j} \vee \cdots \vee p_{I-1,j},$$

to obtain the soft clauses

$$
\begin{gathered}
p_{1,j} \vee \cdots \vee p_{i-2,j} \vee p_{i,j} \vee \cdots \vee p_{I-1,j} \vee \overline{p_{I,j}} \\
\boxed{p_{1,j} \vee \cdots \vee p_{i-2,j} \vee \overline{p_{i-1,j}} \vee p_{i,j} \vee \cdots \vee p_{I-1,j} \vee p_{I,j}} \\
p_{1,j} \vee \cdots \vee p_{i-1,j} \vee \overline{p_{i,j}} \vee p_{i+1,j} \vee \cdots \vee p_{I-1,j} \vee \overline{p_{I,j}} \\
\cdots
\end{gathered}
$$

The third clause is D^I_i and will be used in phase two; the second clause is C^I_{i-1} and will be carried forward to the next stage. The first clause is used immediately in the next step of the first phase. The only exception is in the final step of the first phase, where $i = 2$: in this case, the first clause is D^I_1 and will be carried forward to the next stage.

The second phase of stage I combines the set of clauses D^I_i (see Figure 2), with the hard initial clauses $\overline{p_{i,j}} \vee \overline{p_{k,j}} \vee \overline{p_{I,j}}$ in a tree-like fashion to eventually obtain the unit clause $\overline{p_{I_j}}$. A final resolution with the initial unit clause $p_{I,j}$ gives the desired empty clause $\perp$ to complete stage I.

Phase two obtains the intermediate clauses $D^I_{k,i}$ defined in Figure 3 for $1 \leq k \leq i < I$. The clauses $D^I_{1,i}$ for $i < I-1$ are the same as the clauses D^I_i derived in first phase. Likewise, the clause $D^I_{2,I-1}$ is the same as D^I_{I-1} derived in the first phase. (Since we have $D^I_{2,I-1}$, we do not need $D^I_{1,I-1}$.) All other clauses $D^I_{k,i}$ with $i > k$ are derived by resolving $D^I_{k-1,i}$ against the initial clause $\overline{p_{k-1,j}} \vee \overline{p_{i,j}} \vee \overline{p_{I,j}}$. And, the clauses $D^I_{k,k}$ are obtained by resolving $D^I_{k-1,k}$ against $\overline{p_{k-1,j}} \vee \overline{p_{i,j}} \vee \overline{p_{I,j}}$ and then against $D^I_{k-1,k-1}$. At the end, the clause $D^I_{I-1,I-1}$

Clause	Literals						
D_1^I		$p_{2,j}$	$p_{3,j}$	$\cdots$	$p_{I-2,j}$	$p_{I-1,j}$	$\overline{p_{I,j}}$
D_2^I	$p_{1,j}$	$\overline{p_{2,j}}$	$p_{3,j}$	$\cdots$	$p_{I-2,j}$	$p_{I-1,j}$	$\overline{p_{I,j}}$
D_3^I	$p_{1,j}$	$p_{2,j}$	$\overline{p_{3,j}}$	$\cdots$	$p_{I-2,j}$	$p_{I-1,j}$	$\overline{p_{I,j}}$
$\vdots$	$\vdots$						
D_{I-2}^I	$p_{1,j}$	$p_{2,j}$	$p_{3,j}$	$\cdots$	$\overline{p_{I-2,j}}$	$p_{I-1,j}$	$\overline{p_{I,j}}$
D_{I-1}^I		$p_{2,j}$	$p_{3,j}$	$\cdots$	$p_{I-2,j}$	$\overline{p_{I-1,j}}$	$\overline{p_{I,j}}$

Figure 2: The clauses D_i^I derived in the first phase and used in the second phase. Notice that this set of clauses also follows a pattern. The last column always contains $\overline{p_{I,j}}$. For the rest of the literals, the only differences are the diagonal, and the last position on the left (where the corresponding literal is missing). The diagonal has the literals negated except in the first clause where it is missing.

Clause	Literals						
$D_{k,k}^I$		$p_{k+1,j}$	$p_{k+2,j}$	$\cdots$	$p_{I-2,j}$	$p_{I-1,j}$	$\overline{p_{I,j}}$
$D_{k,k+1}^I$	$p_{k,j}$	$\overline{p_{k+1,j}}$	$p_{k+2,j}$	$\cdots$	$p_{I-2,j}$	$p_{I-1,j}$	$\overline{p_{I,j}}$
$D_{k,k+2}^I$	$p_{k,j}$	$p_{k+1,j}$	$\overline{p_{k+2,j}}$	$\cdots$	$p_{I-2,j}$	$p_{I-1,j}$	$\overline{p_{I,j}}$
$\vdots$	$\vdots$						
$D_{k,I-2}^I$	$p_{k,j}$	$p_{k+1,j}$	$p_{k+2,j}$	$\cdots$	$\overline{p_{I-2,j}}$	$p_{I-1,j}$	$\overline{p_{I,j}}$
$D_{k,I-1}^I$	$p_{k,j}$	$p_{k+1,j}$	$p_{k+2,j}$	$\cdots$	$p_{I-2,j}$	$\overline{p_{I-1,j}}$	$\overline{p_{I,j}}$

Figure 3: The clauses $D_{k,i}^I$ as for the second phase.

is obtained, and this is the same as the unit clause $\overline{p_{I,j}}$. As mentioned, this is resolved against the initial unit clause $p_{I,j}$ to obtain $\perp$ and complete stage I.

This completes the proof of Theorem 10. $\square$

It is interesting to note that none of the MaxSAT refutations for Theorems 8-10 use the $\mathcal{P}$ clauses. The next sections describe core-guided MaxSAT and MaxHS-like algorithms for these principles: they also do not use any $\mathcal{P}$ clauses.

4.2. Polynomial Bounds with Core-Guided MaxSAT Algorithms

This section develops upper bounds for the propositional encodings of the pigeonhole principle and the doubled pigeonhole principle when using core-guided MaxSAT algorithms. The upper bound for the Mutilated Chessboard problem follows from the one of the pigeonhole principle.

Note that even though we are using a SAT solver inside the core-guided MaxSAT algorithm, we show that there are possible executions of the algorithm that run in polynomial time. Additionally, the results obtained in this section consider inside the Core-Guided MaxSAT Algorithms the SAT solver to be a CDCL SAT solver.

4.2.1. Pigeonhole Principle

This section shows that a core-guided MaxSAT algorithm can conclude in polynomial time that for the dual-rail encoding of the pigeonhole principle (PHP_m^{m+1}), more than $m(m+1)$ soft clauses must be falsified, when the hard clauses are satisfied, thus proving that the original PHP_m^{m+1} is unsatisfiable.

The following observations about the dual-rail encoding of the pigeonhole principle Section 3.2 are essential to prove the bound on the run time. First, the clauses of type $\bigvee_{j=1}^{m} \overline{n_{i,j}}$ do not share variables in common with the clauses of type $\overline{p_{i,j}} \vee \overline{p_{k,j}}$. Second, each clause $\bigvee_{j=1}^{m} \overline{n_{i,j}}$ has its variables completely disjoint from any other clause $\bigvee_{j=1}^{m} \overline{n_{k,j}}$, for any distinct values i and k. Third, for any distinct j and j', the variables of $\overline{p_{i,j}} \vee \overline{p_{k,j}}$ are completely disjoint from the variables of $\overline{p_{i,j'}} \vee \overline{p_{k,j'}}$. Since these groups of clauses use different variables, we will be able to obtain disjoint unsatisfiable cores. We can exploit this partition of the clauses, and compute the MaxSAT solution for each group. A MaxSAT solution can be obtained for the formula based on the MaxSAT solutions for each of the groups. This is completely analogous to the way that the MaxSAT resolution refutations split into independent parts handling positive and negative variables separately. Note that the $\mathcal{P}$ clauses will not be used.

Theorem 11. *The core-guided MSU3 algorithm (Algorithm 1) is able to conclude in polynomial time that the dual-rail encoding of the pigeonhole principle (PHP_m^{m+1}) must falsify more than $m(m+1)$ soft clauses, thus proving that the original PHP_m^{m+1} to be unsatisfiable.*

Proof. In this proof we show that there is a possible sequence of steps for the core-guided MSU3 algorithm that in polynomial time falsifies $m(m+1)+1$ soft clauses.

The first stages of the core-guided MSU3 algorithm identifies $m+1$ disjoint sets of unsatisfiable clauses involving the variables $n_{i,j}$. These $m+1$ unsatisfiable cores are $\{\bigvee_{j=1}^{m} \overline{n_{i,j}}, n_{i,1}, \ldots, n_{i,m}\}$, for each i with $1 \leq i \leq m+1$. The i-th unsatisfiable core includes the m soft unit clauses $n_{i,1}, \ldots, n_{i,m}$. Notice that these cores are disjoint; they produce a modification of their soft clauses; namely, the unit clauses $n_{i,j}$ are substituted by $n_{i,j} \vee r_{i,j}$ along with the the cardinality constraints $\sum_{l=1}^{m} r_{i,l} \leq 1$. These clauses, however, will not be used in the next part of the core-guided MSU3 algorithm.

The next stage will find m unsatisfiable cores for each fixed hole j. For a fixed j, the m unsatisfiable cores will be found by using the set of clauses $\{\overline{p_{i,j}} \vee \overline{p_{k,j}} : \text{for all } i \neq k\} \cup \{p_{1,j}, \ldots, p_{m,j}\}$. Let us fix j and see how to obtain the m unsatisfiable cores. In this case the steps cannot be done in parallel; we instead take into account one new pigeon at a time. (Refer to Table 1.)

In the base step, we work only with pigeons 1 and 2, and fixed hole j. The unsatisfiable core is $\{\overline{p_{1,j}} \vee \overline{p_{2,j}}, p_{1,j}, p_{2,j}\}$. By the algorithm MSU3 (Algorithm 1), we introduce two new variables, $r_{1,j}$ and $r_{2,j}$, we eliminate the soft clauses $p_{1,j}$ and $p_{2,j}$, and introduce the hard clauses $p_{1,j} \vee r_{1,j}$, $p_{2,j} \vee r_{2,j}$ and $\overline{r_{1,j}} \vee \overline{r_{1,j}}$. (The last is the boolean translation of the cardinality constraint $r_{1,j} + r_{2,j} \leq 1$.)

Table 1: Steps to obtain m unsatisfiable cores for each hole j

Pigeons	Hard Clauses	Soft Clauses	Clause Substitution	Count of $\bot$
1 and 2	$\overline{p_{1,j}} \vee \overline{p_{2,j}}$	$p_{1,j}$ and $p_{2,j}$	$r_{1,j} \vee p_{1,j}$ $r_{2,j} \vee p_{2,j}$ $\sum_{l=1}^{2} r_{l,j} \leq 1$	1
3	$\overline{p_{1j}} \vee \overline{p_{3j}}$ $\overline{p_{2j}} \vee \overline{p_{3j}}$ $r_{1j} \vee p_{1j}$ $r_{2j} \vee p_{2j}$ $\sum_{l=1}^{2} r_{lj} \leq 1$	p_{3j}	$r_{3j} \vee p_{3j}$ $\sum_{l=1}^{3} r_{lj} \leq 2$	2
$\dots$	$\dots$	$\dots$	$\dots$	$\dots$
i	$\overline{p_{1j}} \vee \overline{p_{ij}}, \dots,$ $\overline{p_{i-1,j}} \vee \overline{p_{ij}}$ $r_{1j} \vee p_{1j}, \dots,$ $r_{i-1j} \vee p_{i-1j}$ $\sum_{l=1}^{i-1} r_{lj} \leq i-2$	p_{ij}	$r_{ij} \vee p_{ij}$ $\sum_{l=1}^{i} r_{lj} \leq i-1$	$i-1$
$\dots$	$\dots$	$\dots$	$\dots$	$\dots$
$m+1$	$\overline{p_{1j}} \vee \overline{p_{m+1j}}, \dots,$ $\overline{p_{mj}} \vee \overline{p_{m+1j}}$ $r_{1j} \vee p_{1j}, \dots,$ $r_{mj} \vee p_{mj}$ $\sum_{l=1}^{m} r_{lj} \leq m-1$	p_{m+1j}	$r_{m+1j} \vee p_{m+1j}$ $\sum_{l=1}^{m+1} r_{lj} \leq m$	m

Suppose that by the $(i-1)$st step, $i-2$ many unsatisfiable cores have been found, and we have eliminated the soft unit clauses $p_{1,j}, \dots, p_{i-1,j}$ and introduced the clauses $p_{1,j} \vee r_{1,j}, \dots, p_{i-1,j} \vee r_{i-1,j}$. Suppose also that we introduced $\sum_{l=1}^{i-1} r_{l,j} \leq i-2$ in the previous step. We also have the set of hard clauses $\{\overline{p_{1,j}} \vee \overline{p_{i,j}}, \dots, \overline{p_{i-1,j}} \vee \overline{p_{i,j}}\}$, and the soft clause $p_{i,j}$. All of these clauses form an unsatisfiable core. Therefore, by Algorithm 1, the unsatisfiable count due to hole j is now $i-1$, the clause $p_{i,j}$ is substituted by $p_{i,j} \vee r_{i,j}$, and we introduce $\sum_{l=1}^{i} r_{l,j} \leq i-1$. Note this cardinality constraint is expressed by a CNF of total size $O(i^2)$ and thus is polynomial size. At the end of the $(m+1)$st step, m unsatisfiable cores have been introduced. See Table 1 for the details of this proof.

So, for every hole j, we produce m unsatisfiable cores. This makes a total of $m+1+m \cdot m = m(m+1)+1$ iterations of the core-guided procedure. We can conclude that the original pigeonhole formula is unsatisfiable. □

Using the intuition given in Section 4.1.2 and the ideas of the proof of Theorem 11, we also obtain the following.

Theorem 12. *The core-guided MSU3 algorithm (Algorithm 1) is able to conclude in polynomial time that the dual-rail encoding of the mutilated chessboard principle must falsify more than $2n^2-2n-2$ soft clauses, thus proving that the* *original mutilated chessboard principle is unsatisfiable.*

4.2.2. Doubled Pigeonhole Principle

This section shows that the MaxSAT core-guided MSU3 Algorithm 1 can conclude that for the dual-rail encoding of the doubled pigeonhole principle ($2\mathrm{PHP}_m^{2m+1}$, presented in Section 2), more than $(2m+1)m$ soft clauses must be falsified, when the hard clauses are satisfied, thus proving that the original $2\mathrm{PHP}_m^{2m+1}$ is unsatisfiable. The proof follows the same structure as the proof of the PHP_m^{m+1} above.

Theorem 13. *The core-guided MSU3 algorithm (Algorithm 1) is able to conclude in polynomial time that the dual-rail encoding of double pigeonhole principle ($2PHP_m^{2m+1}$) must falsify more than $m(2m+1)$ soft clauses, thus proving that the original $2PHP_m^{2m+1}$ is unsatisfiable.*

Proof. As before we divide the clauses of the dual-rail encoding of $2\mathrm{PHP}_m^{2m+1}$ in two. First we consider the clauses containing the $n_{i,j}$ variables. Then we consider the clauses containing the $p_{i,j}$ variables. In both cases, we disregard the $\mathcal{P}$ clauses of the encoding.

Observe that the clauses $(\bigvee_{j=1}^{m} \overline{n_{i,j}})$ with $i \in [2m+1]$, share no variables between them (due to the different index i), and as such they can be considered separately, in turn. The MaxSAT core-guided MSU3 Algorithm 1 receives the hard clause $(\bigvee_{j=1}^{m} \overline{n_{i,j}})$ together with the unit clauses $(n_{i,j})$, $j \in [m]$, and sends all the clauses to the SAT solver which returns unsatisfiable. The unsatisfiable core corresponds to all the clauses sent to the SAT solver. Then the algorithm relaxes the m soft clauses (the unit clauses $n_{i,j}$ for $j \in [m]$), and adds the cardinality constraint stating the sum of the new relaxation variables is smaller or equal to one. The lower bound is increased in one. The new formula is sent to the SAT solver which returns satisfiable. Thus the MaxSAT core-guided MSU3 Algorithm 1 proves that for each $i \in [2m+1]$, the optimum cost of the hard clause $\bigvee_{j=1}^{m} \overline{n_{i,j}}$ together with the unit soft clauses $n_{i,j}$ with $j \in [m]$, is 1. Since $i \in [2m+1]$, the overall cost of the first part of the formula is $2m+1$, and thus we obtain $2m+1$ disjoint unsatisfiable cores for the original $2\mathrm{PHP}_m^{2m+1}$ formula.

Next, we consider the clauses using the $p_{i,j}$ variables. For each hole j a cost of $2m+1$ is obtained, giving a cost of $(2m+1)m$ for all clauses using the $p_{i,j}$ variables (and only these variables). Similar to Section 4.2.1, for a given hole $j \in [m]$, we present the iterations performed by the MaxSAT core-guided MSU3 Algorithm 1 on the formula with the hard clauses $(\overline{p_{i,j}} \vee \overline{p_{k,j}} \vee \overline{p_{l,j}})$, $1 \leq i < k < l \leq 2m+1$, and the unit soft clauses $(p_{i,j})$, $i \in [2m+1]$. The details of each iteration are shown in Table 2.

In the base case corresponding to the first row of Table 2, we work with pigeons 1, 2 and 3 (for the fixed hole j). The MaxSAT core-guided MSU3 Algorithm 1, will send all the clauses to the SAT solver which will determine the soft clauses (column 3) together with the hard clause in column 2 to be an unsatisfiable core. The soft clauses are relaxed and a new cardinality constraint is created, as in column 4 (Clause Substitution). The cost of the formula is increased to one.

Table 2: Steps to obtain $2m-1$ contradictions for every hole j

Pigeons	Hard Clauses	Soft Clauses	Clause Substitution	Cost
1 2 3	$\overline{p_{1,j}} \vee \overline{p_{2,j}} \vee \overline{p_{3,j}}$	$p_{1,j}$ $p_{2,j}$ $p_{3,j}$	$r_{1,j} \vee p_{1,j}$ $r_{2,j} \vee p_{2,j}$ $r_{3,j} \vee p_{3,j}$ $r_{1,j} + r_{2,j} + r_{3,j} \leq 1$	1
4	$\overline{p_{1,j}} \vee \overline{p_{2,j}} \vee \overline{p_{4,j}}$ $\overline{p_{1,j}} \vee \overline{p_{3,j}} \vee \overline{p_{4,j}}$ $\overline{p_{2,j}} \vee \overline{p_{3,j}} \vee \overline{p_{4,j}}$ $r_{1,j} \vee p_{1,j}$ $r_{2,j} \vee p_{2,j}$ $r_{3,j} \vee p_{3,j}$ $r_{1,j} + r_{2,j} + r_{3,j} \leq 1$	$p_{4,j}$	$r_{4,j} \vee p_{4,j}$ $\sum_{l=1}^{4} r_{l,j} \leq 2$	2
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
i	$\overline{p_{1,j}} \vee \overline{p_{2,j}} \vee \overline{p_{i,j}}$ $\cdots$ $\overline{p_{i-2,j}} \vee \overline{p_{i-1,j}} \vee \overline{p_{i,j}}$ $r_{1,j} \vee p_{1,j}$ $\cdots$ $r_{i-1,j} \vee p_{i-1,j}$ $\sum_{l=1}^{i-1} r_{l,j} \leq i-3$	$p_{i,j}$	$r_{i,j} \vee p_{i,j}$ $\sum_{l=1}^{i} r_{l,j} \leq i-2$	$i-2$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
$2m+1$	$\overline{p_{1,j}} \vee \overline{p_{2,j}} \vee \overline{p_{2m+1,j}}$ $\cdots$ $\overline{p_{2m-1,j}} \vee \overline{p_{2m,j}} \vee \overline{p_{2m+1,j}}$ $r_{1,j} \vee p_{1,j}$ $\cdots$ $r_{2m,j} \vee p_{2m,j}$ $\sum_{l=1}^{2m} r_{l,j} \leq 2m-2$	$p_{2m+1,j}$	$r_{2m+1,j} \vee p_{2m+1,j}$ $\sum_{l=1}^{2m+1} r_{l,j} \leq 2m-1$	$2m-1$

The remaining steps run in a similar way. The MaxSAT core-guided MSU3 Algorithm 1 sends all the hard clauses to the SAT solver, including the so-far relaxed clauses and the current cardinality constraint on the relaxation variables, together with the soft clauses (which have not been relaxed yet). A new pigeon is considered in the current iteration. Assume it to be pigeon i (as in row 3 of Table 2). Then the SAT solver determines a new unsatisfiable core corresponding to the soft clause $p_{i,j}$ (column 3) and the hard clauses in column 2 . The soft clause is relaxed and replaced by the hard clause $(r_{i,j} \vee p_{i,j})$ (column 4, row 3). The cardinality constraint is updated to include the new relaxation variable and increases its right and side in one to $i-2$ (column 4, row 3). Finally the cost of the formula is updated to $i-2$ (column 5, row 3).

After performing the last iteration dealing with pigeon $2m+1$ (as in row 4), the MaxSAT core-guided MSU3 Algorithm 1 will obtain a satisfiable formula (for a fixed j). We thus obtain $2m-1$ unsatisfiable cores for each hole j. Since there are m many holes j, the cost of the sub-formula considering the clauses containing the $p_{i,j}$ variables (disregarding $\mathcal{P}$ clauses) is $(2m-1)m$. Thus the total cost of the $2\mathrm{PHP}_m^{2m+1}$ formula with the dual-rail encoding is computed as $2m+1+(2m-1)m = (2m+1)m+1$, thereby proving the unsatisfiability of the original $2\mathrm{PHP}_m^{2m+1}$ formula.

Given the above, we are guaranteed to find the required number of unsatisfiable cores to determine the original $2\mathrm{PHP}_m^{2m+1}$ formula to be unsatisfiable. Additionally, we can also guarantee that the calls to the SAT solver made by the MaxSAT core-guided MSU3 Algorithm 1 can be made in polynomial time. Namely, for the first part of the formula using the $n_{i,j}$ variables, the unsatisfiable call corresponds to propagating the unit soft clauses to obtain the unsatisfiability of the formula.

For the second part of the formula, we describe a possible sequence of iterations of a SAT solver running in polynomial time. As mentioned earlier, any SAT solver that acts by returning either a satisfying assignment or an unsatisfiable core would be acceptable for proving Theorem 10. However, for concreteness, we describe how a CDCL SAT solver can be used. Consider that we are given the unsatisfiable formula corresponding to pigeon i for hole j as in row 3 of Table 2. The CDCL SAT solver will receive the (hard) clauses of column 2 and the (soft) clause in column 3 (of row 3 in Table 2). Table 3 shows the sequence of iterations made by the SAT solver. The first column and second columns labeled "Dec. Level" and "Decisions" respectively, show the current decision level and the current decision of the SAT solver. The third column ("Clauses") presents the clauses used in the propagation of the assignments of column four ("Propagations"). When a conflict is reached, column four contains the symbol $\perp$ and in column five ("Learn") we present the clause that is learned from that conflict (except at decision level 0, in which case the formula is declared unsatisfiable).

Initially at row 1, the unit clause $p_{i,j}$ is propagated, assigning $p_{i,j} = 1$ at decision level 0. In rows 2 to 6, we deal with pigeon 1, starting by assigning it to hole j, that is, deciding $p_{1,j} = 1$ at decision level 1. This causes the clauses $\overline{p_{1,j}} \vee \overline{p_{x,j}} \vee \overline{p_{i,j}}$, to propagate the assignments $p_{x,j} = 0$, with $x \in [2, i-1]$, in row 3 and in turn, in row 4 the clauses $r_{x,j} \vee p_{x,j}$ propagate the assignments $r_{x,j} = 1$. Due to the clauses of the constraint $\sum_{l=1}^{i-1} r_{l,j} \leq i-3$, then a conflict is reached and the learned clause corresponds to the only decision literal negated, that is $\overline{p_{1,j}}$. Rows 5 and 6, propagate the assignments $p_{1,j} = 0$ and $r_{1,j} = 1$ (respectively), using the learned clause and the clause $r_{1,j} \vee p_{1,j}$ at the backtracked decision level 0.

The following rows of the Table 3 follow a similar structure taking in account each of the pigeons 2 to $i-3$ in turn, until row 16. That is, the SAT solver decides to assign a pigeon s to hole j and then after propagation finds a conflict, learns a clause corresponding to the negation of assigning s to hole j and backtracks to decision level 0. The associated relaxation variable $r_{s,j}$ is assigned 1 at decision level 0.

Table 3: Analysis of SAT solver call for iteration with pigeon i and hole j

Dec. Level	Decisions	Clauses	Propagations	Learned
0		$p_{i,j}$	$p_{i,j} = 1$	
1	$p_{1,j} = 1$	$\overline{p_{1,j}} \vee \overline{p_{2,j}} \vee \overline{p_{i,j}}$ $\dots$ $\overline{p_{1,j}} \vee \overline{p_{i-1,j}} \vee \overline{p_{i,j}}$	$p_{2,j} = 0$ $\dots$ $p_{i-1,j} = 0$	
1		$r_{2,j} \vee p_{2,j}$ $\dots$ $r_{i-1,j} \vee p_{i-1,j}$	$r_{2,j} = 1$ $\dots$ $r_{i-1,j} = 1$	
1		$\sum_{l=1}^{i-1} r_{l,j} \leq i-3$	$(\sum_{l=1}^{i-1} r_{l,j} \leq i-3) \vdash \bot$	$\overline{p_{1,j}}$
0		$\overline{p_{1,j}}$	$p_{1,j} = 0$	
0		$r_{1,j} \vee p_{1,j}$	$r_{1,j} = 1$	
		$\dots$		
1	$p_{s,j} = 1$	$\overline{p_{s,j}} \vee \overline{p_{s+1,j}} \vee \overline{p_{i,j}}$ $\dots$ $\overline{p_{s,j}} \vee \overline{p_{i-1,j}} \vee \overline{p_{i,j}}$	$p_{s+1,j} = 0$ $\dots$ $p_{i-1,j} = 0$	
1		$r_{s+1,j} \vee p_{s+1,j}$ $\dots$ $r_{i-1,j} \vee p_{i-1,j}$	$r_{s+1,j} = 1$ $\dots$ $r_{i-1,j} = 1$	
1		$\sum_{l=1}^{i-1} r_{l,j} \leq i-3$	$(\sum_{l=1}^{i-1} r_{l,j} \leq i-3) \vdash \bot$	$\overline{p_{s,j}}$
0		$\overline{p_{s,j}}$	$p_{s,j} = 0$	
0		$r_{s,j} \vee p_{s,j}$	$r_{s,j} = 1$	
		$\dots$		
1	$p_{i-3,j} = 1$	$\overline{p_{i-3,j}} \vee \overline{p_{i-2,j}} \vee \overline{p_{i,j}}$ $\overline{p_{i-3,j}} \vee \overline{p_{i-1,j}} \vee \overline{p_{i,j}}$	$p_{i-2,j} = 0$ $p_{i-1,j} = 0$	
1		$r_{i-2,j} \vee p_{i-2,j}$ $r_{i-1,j} \vee p_{i-1,j}$	$r_{i-2,j} = 1$ $r_{i-1,j} = 1$	
1		$\sum_{l=1}^{i-1} r_{l,j} \leq i-3$	$(\sum_{l=1}^{i-1} r_{l,j} \leq i-3) \vdash \bot$	$\overline{p_{i-3,j}}$
0		$\overline{p_{i-3,j}}$	$p_{i-3,j} = 0$	
0		$r_{i-3,j} \vee p_{i-3,j}$	$r_{i-3,j} = 1$	
0		$\sum_{l=1}^{i-1} r_{l,j} \leq i-3$	$r_{i-2,j} = 0$ $r_{i-1,j} = 0$	
0		$r_{i-2,j} \vee p_{i-2,j}$ $r_{i-1,j} \vee p_{i-1,j}$	$p_{i-2,j} = 1$ $p_{i-1,j} = 1$	
0		$\overline{p_{i-2,j}} \vee \overline{p_{i-1,j}} \vee \overline{p_{i,j}}$	$(\overline{p_{i-2,j}} \vee \overline{p_{i-1,j}} \vee \overline{p_{i,j}}) \vdash \bot$	

At row 17, because of the constraint $\sum_{l=1}^{i-1} r_{l,j} \leq i-3$, and because previously the $i-3$ variables $r_{x,j}$ ($x \in [i-3]$) were assigned value 1, then the two remaining variables are assigned value 0, that is, $r_{i-2,j} = 0$ and $r_{i-1,j} = 0$ at decision level 0. This causes the assignments $p_{i-2,j} = 1$ and $p_{i-1,j} = 1$ in row 18, and a conflict is reached in row 19 at decision level 0, which determines the formula to be unsatisfiable.

An important observation about the iterations described in Table 3, is that in spite of the fact that there are decisions being made, these are all made at decision level 1, propagating into a conflict, and then backtracking to decision level 0, that is, the search is bounded to at most 1 decision level. The total number of propagations is $4+\sum_{s=1}^{i-3}[2(i-s-1)+2] = i^2-i-2$, that is, $O(i^2)$. Since $i \in [4, 2m+1]$ we obtain $O(m^3)$ propagations which is polynomial in m.[6]

$\square$

4.3. Polynomial Bounds with MaxHS-Like Algorithms

This section develops upper bounds for the dual-rail propositional encodings of the pigeonhole principle, the doubled pigeonhole principle and the parity principle when using MaxHS-like algorithms. The upper bound for the mutilated chessboard problem follows also in the case from the one for the pigeon-hole principle.

Analogously to Section 4.2, note that even though we are using a SAT solver and a minimum hitting set solver inside the MaxHS-like algorithm, we show that there are possible executions of the algorithm that run in polynomial time. Similarly to Section 4.2, the SAT solvers used inside the MaxHS-like algorithm is considered to be a CDCL SAT solver.

4.3.1. Pigeonhole Principle

From Section 3.2, the unsatisfiability of the pigeonhole principle using the dual-rail MaxSAT encoding (PHP_m^{m+1}) implies that $(\mathrm{PHP}_m^{m+1})^{\mathrm{dr}}$ has a MaxSAT cost of at least $m(m+1)+1$. The following shows that a possible execution of the basic MaxHS algorithm can derive this MaxSAT cost for $(\mathrm{PHP}_m^{m+1})^{\mathrm{dr}}$ in polynomial time. Observe that if the $\mathcal{P}$ clauses $\overline{p_{ij}} \vee \overline{n_{ij}}$ from $(\mathrm{PHP}_m^{m+1})^{\mathrm{dr}}$ are ignored, then the formula can be partitioned into disjoint formulas, namely into the formulas $\mathcal{L}_i$, $i \in [m+1]$, representing the encoding of each $\mathsf{AtLeast1}$ constraint, $\mathcal{L}_i = (\neg n_{i1} \vee \cdots \vee \neg n_{im})$; and into the formulas $\mathcal{M}_j$, $j \in [m]$, representing the encoding of each $\mathsf{AtMost1}$ constraint, $\mathcal{M}_j = \bigwedge_{i_1=1}^{m} \bigwedge_{i_2=i_1+1}^{m+1} (\neg p_{i_1 j} \vee \neg p_{i_2 j})$. Thus, one can compute a solution for each of these formulas separately and obtain a lower bound on the MaxSAT solution for the complete formula $(\mathrm{PHP}_m^{m+1})^{\mathrm{dr}}$. We show that the contribution of each $\mathcal{L}_i$ to the total cost is 1. Since there are $m+1$ such formulas, the contribution of all $\mathcal{L}_i$ formulas is $m+1$. Then, we show that each $\mathcal{M}_j$ contributes with a cost of m, and since there are m such formulas, the contribution of all $\mathcal{M}_j$ formulas is m^2. Therefore, we have a lower bound on the total cost of $m(m+1)+1$, proving the original formula to be unsatisfiable. In the remainder of this section we consider $\mathcal{S}$ to be the set of all the soft clauses obtained from the dual-rail encoding $(\mathrm{PHP}_m^{m+1})^{\mathrm{dr}}$.

Theorem 14. *Given a formula $\mathcal{L}_i \cup \mathcal{S}$, where the "at least" clauses $\mathcal{L}_i$ and the soft clauses $\mathcal{S}$ are obtained from $(PHP_m^{m+1})^{\mathrm{dr}}$, there is an execution of the basic MaxHS algorithm that computes a MaxSAT solution of cost 1 in polynomial time.*

[6]The case of the first 3 pigeons can be disregarded since it corresponds to 3 propagations.

Proof. Consider Algorithm 2. In the first iteration, an empty MHS is computed in line 4. The SAT solver (line 5) tests the satisfiability of the hard clause $(\neg n_{i1} \vee \ldots \vee \neg n_{im})$ together with the m soft unit clauses $n_{i1}, \ldots, n_{im}$. Observe that the SAT solver proves the formula to be unsatisfiable by unit propagation. From this unsatisfiable formula a new set to hit is added to K. The new set is the set of unit soft clauses in the unsatisfiable core just obtained, i.e. $\{n_{i1}, \ldots, n_{im}\}$ (line 7).

In the second iteration, K contains only 1 set to hit, and any of its elements can be selected as a minimum hitting set. W.l.o.g. suppose that n_{ij} is selected, and eliminated from the set of soft clauses to use. Then the SAT solver tests for the satisfiability of $\neg n_{i1} \vee \cdots \vee \neg n_{im}$ with the set of soft clauses $\{n_{il} : l \neq j\}$, reporting the formula to be satisfiable. The cost of the solution is 1. □

Before presenting the result for the $\mathcal{M}_j$ formulas, we make a few observations.

Observation 15. *Consider a complete graph G, i.e. a* clique*, of $m+1$ vertices. A vertex cover of a G can be computed in polynomial time and has size m. Simply arbitrarily pick one of the vertices to be out of the cover.*

Observation 16. *Let graph G be composed of a clique of size $m-1$ plus one extra vertex that is connected to at least one of the vertices of the clique. Then a vertex cover of G has size $m-2$, and can be computed in polynomial time by including all vertices, except for two of them that have the smallest degree, i.e. the smallest number of neighbors.*

The hitting set algorithm used below will need to distinguish between the two cases of Observations 15 and 16; clearly this can be done in polynomial time.

Theorem 17. *Given a formula $\langle \mathcal{M}_j, \mathcal{S} \rangle$ s.t. $\mathcal{M}_j$ and $\mathcal{S}$ are from $(PHP_m^{m+1})^{\mathrm{dr}}$, there is an execution of the basic MaxHS algorithm that computes a MaxSAT solution of cost m in polynomial time.*

Proof. The idea of the proof is to show that there is a possible ordering of the set of cores returned by the SAT solver that will cause, at each step, the sets in K to induce a graph that is either a clique or composed of a clique plus one extra vertex connected to some of the other vertices. Then from the previous observations a MHS can be computed in polynomial time. In the final iteration, the graph induced by the sets in K will correspond to a clique of size $m+1$, and therefore the final minimum hitting set will have size m, thus reporting a solution with cost m.

Consider an order of the clauses to consider in $\mathcal{M}_j$, induced by the following choice of variables. First, consider the clauses that involve only p_{1j} and p_{2j} (one hard clause and two soft clauses); then the clauses that involve only p_{1j}, p_{2j}, p_{3j} (three hard clauses and three soft clauses); then clauses that involve only $p_{1j}, p_{2j}, p_{3j}, p_{4j}$ (six hard clauses and four soft clauses); and so on until all variables $p_{i,j}$ are considered. Observe that due to the structure of $\mathcal{M}_j$, every unsatisfiable core returned by the SAT solver has two soft unit clauses. The SAT solver can easily find the unsatisfiable cores by unit propagation. Consequently, the chosen order of variables implies that all pairs of the current set of variables are added to K before considering a new variable. The first set added to K using this order is $\{p_{1j}, p_{2j}\}$, followed by $\{p_{1j}, p_{3j}\}$, $\{p_{2j}, p_{3j}\}$, etc.

Since sets in K are pairs, each set can be regarded as an edge of the induced graph. Given the previous ordering of the variables (and consequently of the sets in K), the induced graph forms a "growing" clique, that is, it is either a clique with all the variables considered so far, or it is a clique with the previous variables plus a new variable connected to some of the previous variables.

Finally, since each clause in $\mathcal{M}_j$ produces an unsatisfiable core returned by the SAT solver (corresponding to a new set to hit in K), the total number of iterations is equal to the number of clauses in $\mathcal{M}_j$ plus 1, which is $C_2^{m+1} + 1 = \frac{(m+1)m}{2} + 1$. On the other hand, the size of the minimum hitting set is m by Observation 15. ☐

Theorem 18. *The basic MaxHS algorithm algorithm (Algorithm 2) is able to conclude in polynomial time that the dual-rail encoding of the pigeonhole principle (PHP_m^{m+1}) must falsify more than $m(m+1)$ soft clauses, thus proving the original PHP_m^{m+1} to be unsatisfiable.*

Proof. Follows from Theorem 14 and Theorem 17. ☐

Using the intuition given in Section 4.1.2 and the ideas of the proofs of Theorems 14, 17 and 18, we obtain also the following.

Theorem 19. *The basic MaxHS algorithm algorithm (Algorithm 2) is able to conclude in polynomial time that the dual-rail encoding of the mutilated chessboard principle must falsify at least $2n^2 - 2n - 2$ soft clauses, thus proving that the original mutilated chessboard principle is unsatisfiable.*

4.3.2. Doubled Pigeonhole Principle

Similar to the pigeonhole principle case, 2PHP_m^{2m+1} is unsatisfiable if and only if the cost of $(2\text{PHP}_m^{2m+1})^{\text{dr}}$ is at least $m(2m+1)+1$ (see Section 3.2). If the $\mathcal{P}$ clauses from $(2\text{PHP}_m^{2m+1})^{\text{dr}}$ are ignored, then the resulting formula can be partitioned into the disjoint formulas $\mathcal{L}_i$ ($i \in [2m+1]$), $\mathcal{L}_i = (\neg n_{i1} \vee \cdots \vee \neg n_{im})$, and the disjoint formulas $\mathcal{M}_j$ (for $j \in [m]$):

$$\mathcal{M}_j = \bigwedge_{i_1=1}^{2m-1} \bigwedge_{i_2=i_1+1}^{2m} \bigwedge_{i_3=i_2+1}^{2m+1} (\neg p_{i_1 j} \vee \neg p_{i_2 j} \vee \neg p_{i_3 j}).$$

One can compute a MaxSAT solution for each of $\mathcal{L}_i$ and $\mathcal{M}_j$ separately and obtain a lower bound on the cost of the MaxSAT solution for the complete formula $(2\text{PHP}_m^{2m+1})^{\text{dr}}$. Processing each formula $\mathcal{L}_i$ can be done as in the PHP case (see Theorem 14); each $\mathcal{L}_i$ contributes 1 to the size of the minimum hitting set.

As shown below, the contribution of each $\mathcal{M}_j$ to the MaxSAT cost is $2m-1$, and since there are m such formulas, the contribution from all $\mathcal{M}_j$ formulas is $m(2m-1)$. As a result, the lower bound on the total cost for $(2\mathrm{PHP}_m^{2m+1})^{\mathrm{dr}}$ is $m(2m-1)+2m+1 = m(2m+1)+1$, thus, proving the formula $2\mathrm{PHP}_m^{2m+1}$ to be unsatisfiable. We also show that the basic MaxHS algorithm is able to derive the MaxSAT cost for each $\mathcal{M}_j$ in polynomial time. To proceed, we first make a couple observations.

Observation 20. *Let X be a set of elements of size $|X| = s+2$. Let K be the set of all possible triples $\{x_i, x_j, x_r\}$ of elements of X, $1 \leq i < j < r \leq s+2$. Then any set of s different elements from X is a minimum hitting set for K.*

Observation 20 is immediately clear by inspection.

Observation 21. *Let X be a set of elements of size $|X| = s+2$, and an additional element p not in X. Let K be the set of all possible triples $\{x_i, x_j, x_r\}$ of elements of X, $1 \leq i < j < r \leq s+2$, together with a strict subset of the triples $\{x_i, x_j, p\}$, $x_i, x_j \in X$, $1 \leq i < j \leq s+2$. A minimum hitting set of K has size s and does not contain p.*

To prove Observation 21, note that any hitting set must contain at least s of the members of X by Observation 20. On the other hand, if, say, the triple $\{x_{s+1}, x_{s+2}, p\}$ is missing from K, then $\{x_1, \ldots, x_s\}$ is a hitting set of size s.

Theorem 22. *Given a formula $\mathcal{M}_j, \mathcal{S}e$ such that the "at most" clauses $\mathcal{M}_j$ and the soft clauses $\mathcal{S}$ are from $(2PHP_m^{2m+1})^{\mathrm{dr}}$, there is an execution of the basic MaxHS algorithm that computes a MaxSAT solution of cost $2m-1$ in polynomial time.*

Proof. The proof illustrates a possible setup of the MHS-algorithm that does a polynomial number of iterations s.t. each minimum hitting set is computed in polynomial time. This setup is achieved by ordering the cores computed by the SAT solver (line 5 of Algorithm 2). Similarly to the PHP case, we order the clauses in the SAT solver, by considering an order on the variables. We consider the clauses that involve only p_{1j}, p_{2j}, p_{3j} (only one hard clause and three soft clauses), then the clauses that involving only $p_{1j}, p_{2j}, p_{3j}, p_{4j}$ (the three hard clauses $\neg p_{1j} \vee \neg p_{2j} \vee \neg p_{4j}$, $\neg p_{1j} \vee \neg p_{3j} \vee \neg p_{4j}$ and $\neg p_{2j} \vee \neg p_{3j} \vee \neg p_{4j}$ and four soft clauses), and so on until all variables/clauses are considered. In contrast to the PHP case, when considering the clauses with a new element, we need to take the clauses in a particular order. For example, after considering all clauses involving only $p_{1j}, p_{2j}, p_{3j}, p_{4j}$, we will consider the clauses that involve p_{5j}. We order these clauses by first considering the clauses that involve only p_{1j}, p_{2j}, p_{5j} (one hard clause), then the clauses that contain $p_{1j}, p_{2j}, p_{3j}, p_{5j}$ (two more hard clauses), and finally the clauses that involve only $p_{1j}, p_{2j}, p_{3j}, p_{4j}, p_{5j}$ (three more hard clauses). Note that, the new sets to hit include the new element being added (in the example above, the element p_{5j}). On the other hand, by Observation 21, the minimum hitting set solution does not include the element being added. As such, if we disregard the new element being added in the new sets to hit, then we have pairs which can be regarded as edges of a graph. The graph induced by the pairs in the hitting sets will be a growing clique, as in the PHP case (Theorem 17). The orderings of the variables guarantee that the sets to hit in K either contain all the possible combinations of size 3 of the variables we are considering, or they induce a "growing" clique. In the first case a minimum hitting set is obtained in polynomial time using the result of Observation 20. For the second case, we obtain a minimum hitting set in polynomial time similarly to Theorem 17, using Observation 21. The process of creating a core (and the corresponding set to hit in K) is repeated for each clause in $\mathcal{M}_j$, thus the total number of iterations is equal to the number of clauses plus 1, which is $\binom{2m+1}{3} + 1 = \frac{(2m+1)(2m)(2m-1)}{6} + 1$. Additionally, the reported cost corresponds to the size of the MHS found in the last iteration, i.e., when all variables are considered. Thus, by Observation 20, the reported cost is $2m - 1$. □

Theorem 23. *The basic MaxHS algorithm algorithm (Algorithm 2) is able to conclude in polynomial time that the dual-rail encoding of the doubled pigeonhole principle ($2PHP_m^{2m+1}$) must falsify more than $m(2m+1)$ soft clauses, thus proving the original $2PHP_m^{2m+1}$ to be unsatisfiable.*

Proof. Follows from Theorem 14 and Theorem 22. □

4.3.3. Parity Principle

This section presents an upper bound for the parity principle using MaxHS-like algorithms and the dual-rail encoding. In Section 6 we will see that the same principle requires exponential size proofs when using MaxSAT resolution (and the dual-rail encoding). It is open whether core-guided MaxSAT has polynomial size refutations of the parity principle.

Recall the definition of the parity principle in Section 2.3.3, and its dual-rail encoding in Section 3.2. The variables of the dual-rail encoding of the parity principle are $\{n_{i,j} : 1 \leq i < j \leq m\}$ and $\{p_{i,j} : 1 \leq i < j \leq m\}$.

As in the case of the pigeonhole principles, we can ignore the $\mathcal{P}$ clauses mixing $n_{i,j}$ and $p_{i,j}$ variables. As before we partition the principle into two disjoint formulas, the one using only $n_{i,j}$ variables, and the one using $p_{i,j}$ variables.

Let $\mathcal{L}_i$, $i \in [m]$, represent the encoding of each $\mathsf{AtLeast1}$ constraint, $\mathcal{L}_i = (\neg n_{i1} \vee \cdots \vee \neg n_{im})$; and the formulas $\mathcal{M}_j$, $j \in [m]$, represent the encoding of each $\mathsf{AtMost1}$ constraint, $\mathcal{M}_j = \bigwedge_{i=1,i\neq j}^{m} \bigwedge_{k=i+1,k\neq j}^{m} (\neg p_{ij} \vee \neg p_{kj})$. We show that the contribution of all the $\mathcal{L}_i$ formulas is $\frac{m+1}{2}$. Then, we show that the contributions of all the $\mathcal{M}_j$ formulas to the minimum hitting set size is $\frac{(m-2)(m-1)}{2} + \frac{m-1}{2}$.

Summing up the contribution of each formula, we obtain $\frac{m+1}{2} + \frac{(m-2)(m-1)}{2} + \frac{m-1}{2} = \frac{m(m-1)}{2} + 1$. Since the number of variables of the normal encoding of the parity principle is $\frac{m(m-1)}{2}$, we get a basic maxHS refutation of the principle, using the dual-rail encoding and the minimum hitting set algorithm.

Theorem 24. *Given the formula $\mathcal{L}_i \cup \mathcal{S}$, where $\mathcal{L}_i$ and $\mathcal{S}$ are the hard and soft dual-rail clauses encoding the "at least" part of the Parity principle, there is an execution of the basic MaxHS algorithm that computes a MaxSAT solution of cost $\frac{m+1}{2}$ in polynomial time.*

Proof. First we will show by induction, how from the soft and hard clauses using variables $n_{i,j}$ (for all $i \leq s$ and all j such that $i < j \leq m$), we obtain a minimum hitting set of size $\frac{s+1}{2}$ if s is odd, and of size $\frac{s}{2}$ if s is even.

Base case, $s = 1$: We assume the SAT algorithm returns the unsatisfiable core:

$$\{\overline{n_{1,2}} \vee \cdots \vee \overline{n_{1,m}}, n_{1,2}, \ldots, n_{1,m}\}$$

At this point, the set K of sets to hit is $\{\{n_{1,2}, \ldots, n_{1,m}\}\}$. Therefore, the minimum hitting set hs could contain any of the elements of the set $\{n_{1,2}, \ldots, n_{1,m}\}$. W.l.o.g. $hs = \{n_{1,m}\}$, and $n_{1,m}$ is eliminated from the set of soft clauses.

Induction step: Suppose now that the algorithm has dealt with s unsatisfiable sets, and suppose s is even. The induction hypothesis is that the minimum size of a hitting set is $s/2$ and that we have the hitting set $hs = \{n_{1,2}, n_{3,4}, \ldots, n_{s-1,s}\}$, and the soft clauses of hs have been eliminated from the set of clauses to work with. At this point that CDCL algorithm (nondeterministically) returns the unsatisfiable core:

$$\{\overline{n_{1,s+1}} \vee \cdots \vee \overline{n_{s,s+1}} \vee \cdots \vee \overline{n_{s+1,m}}, n_{1,s+1}, \ldots, n_{s,s+1}, n_{s+1,s+2}, \ldots, n_{s+1,m}\}$$

The set K of sets to hit is now

$$K = \{\{n_{1,2}, \ldots, n_{1,m}\}, \{n_{1,2}, n_{2,3}, \ldots, n_{2,m}\}, \ldots, \{n_{1,s+1}, \ldots, n_{s,s+1}, n_{s+1,s+2}, \ldots, n_{s+1,m}\}\}.$$

We can take $hs = \{n_{1,2}, \ldots, n_{s-1,s}, n_{s+1,i}\}$, where i is any element smaller than $s + 1$. In this case, the size of hs is $\frac{s}{2} + 1$, and we are finished.

Suppose now s is an odd number. By the induction hypothesis, the minimum hitting set at this point has size $\frac{s+1}{2}$, and is the set $hs = \{n_{1,2}, \ldots, n_{s-2,s-1}, n_{s,1}\}$ Now assume that the SAT algorithm nondeterministically returns the unsatisfiable core:

$$\{\overline{n_{1,s+1}} \vee \cdots \vee \overline{n_{s,s+1}} \vee \cdots \vee \overline{n_{s+1,m}}, n_{1,s+1}, \ldots, n_{s,s+1}, n_{s+1,s+2}, \ldots, n_{s+1,m}\}$$

At this point, the set of sets to hit K is

$$K = \{\{n_{1,2}, \ldots, n_{1,m}\}, \{n_{1,2}, n_{2,3}, \ldots, n_{2,m}\}, \{n_{1,s+1}, \ldots, n_{s,s+1}, n_{s+1,s+2}, \ldots, n_{s+1,m}\}\}.$$

The hitting set still requires size $\frac{s+1}{2}$ and we can use $hs = \{n_{1,2}, \ldots, n_{s-2,s-1}, n_{s,s+1}\}$.

Let us justify now that the sets hs are minimum hitting sets. Any hitting set would have to mention every node of the graph in $\{1, \ldots, s + 1\}$ at least once, since there is a set in K for every such node. Every variable $n_{i,j}$ mentions two nodes. So at least $\frac{s+1}{2}$ elements are needed if s is odd, and $\frac{s}{2}$ if s is even. Any smaller set would fail to mention at least one element.

Thus, the minimum hitting set size is $\frac{m+1}{2}$ at the end of m steps. ◻

Theorem 25. *Given a formula $\bigcup_{j=1}^{m} \mathcal{M}_j \cup \mathcal{S}$ s.t. each $\mathcal{M}_j$ and $\mathcal{S}$ are clauses expressing the "at most" part in the dual-rail encoding of the parity principle, there is an execution of the basic MaxHS algorithm that computes a MaxSAT solution of cost $\frac{(m-2)(m-1)}{2} + \frac{m-1}{2}$ in polynomial time.*

Proof. Now the algorithm works with clauses of type $\overline{p_{i,j}} \vee \overline{p_{k,l}}$. We assume with no loss of generality that $i \leq k$, and also that $i < j$ and $k < l$ since by convention $p_{i,j}$ and $p_{k.l}$ are the same variables as $p_{j,i}$ and $p_{l.k}$. We consider clauses of $\overline{p_{i,j}} \vee \overline{p_{k,l}}$ of three types:

(a) $i = k$ and $1 \leq i < j < l \leq m$. We will deal with these in step 1 below.

(b) $j = k$ and $1 \leq i < j < l \leq m$. We will deal with these in step 2.

(c) $j = l$ and $1 \leq i < k < j \leq m$. When $j \neq m$ we deal with them in step 2, and when $j = m$ we deal with them in step 3.

The variables $p_{i,j}$ will be viewed as the nodes of a graph. Given a clause $\overline{p_{i,j}} \vee \overline{p_{k,l}}$, where $i = k$ or $j = k$ or $j = l$, we can think of it as an edge from node $p_{i,j}$ to node $p_{k,l}$, and also denote it as $\{p_{i,j}, p_{k,l}\}$.

The construction of this part of the proof will consist of three steps. Step 1 will deal with all the unsatisfiable cores of type $\{\overline{p_{i,j}} \vee \overline{p_{i,l}}, p_{i,j}, p_{i,l}\}$, where $1 \leq i < j < l \leq m$. For fixed i, the set

$$K_i = \{\{p_{i,j}, p_{i,l}\} : \text{ for all } j, l \text{ s.t. } 1 \leq i < j < l \leq m\} \tag{6}$$

is a clique. The nodes of the clique consist of all the $p_{i,j}$ variables with $j > i$. Step 1 will generate unsatisfiable cores corresponding to disjoint cliques of decreasing size. For instance, if $m = 5$, the first clique will contain all pairs of elements in $\{p_{1,2}, p_{1,3}, p_{1,4}, p_{1,5}\}$, the second will contain all pairs in $\{p_{2,3}, p_{2,4}, p_{2,5}\}$, the third will contain only the pair $\{p_{3,4}, p_{3,5}\}$, and the fourth will be $\{p_{4,5}\}$.

Step 1 will increase the size of the minimum hitting set to $\frac{(m-2)m-1}{2}$. Step 2 will deal with the rest of unsatisfiable cores except cores of the form $\{\overline{p_{i,m}} \vee \overline{p_{j,m}}, p_{i,m}, p_{j,m}\}$. During this second step the minimum size of the hitting set will not be increased. (The point of this is that the unsatisfiable cores found in step 2 will increase the size of the minimum hitting set during step 3.)

Step 3 will obtain $\frac{(m-1)(m-2)}{2}$ unsatisfiable cores of type $\{\overline{p_{i,m}} \vee \overline{p_{k,m}}, p_{i,m}, p_{k,m}\}$, where $1 \leq i < k \leq m$. This last step will increase the size of the minimum hitting set by $\frac{m-1}{2}$.

Step 1: This step works the same as the argument for the at-most-1 clauses in the pigeonhole principle. First we handle all cores involving node 1, namely $\{\overline{p_{1,j}} \vee \overline{p_{1,l}}, p_{1,j}, p_{1,l}\}$. These cores generate the set of sets to hit: $K_1 = \{\{p_{1,j}, p_{1,l}\} : 1 < j < l \leq m\}$. The minimum hitting set hs for K_1 will contain all but one of the elements in $\{p_{1,2}, \ldots, p_{1,m}\}$. Therefore at this point the size of hs is $m - 2$ (since we are considering only the $p_{1,j}$ variables for now). The justification that hs is a minimum hitting set is as follows: hs is a hitting set for all the cores involving only node 1, because it is only missing one variable $p_{1,j}$. As a consequence, every set in K_1 has an element in hs. It is minimal because if hs lacked two elements, for instance $p_{1,j}$ and $p_{1,k}$, then it would miss the set $\{p_{1,j}, p_{1,k}\}$ in K_1. The sets K_1 and hs get built the same way as it is done in the pigeonhole principle.

Working with node 2 next, we deal with all the unsatisfiable cores using elements $\{p_{2,3}, \ldots, p_{2,m}\}$. As before, we increase the size of hs by $m-3$.

In general, working with the i-th node, we would obtain all unsatisfiable cores using elements $\{p_{i,i+1}, \ldots, p_{i,m}\}$. So the unsatisfiable cores will be of the form $\{\overline{p_{i,j}} \vee \overline{p_{i,l}}, p_{i,j}, p_{i,l}\}$. These form a set of sets to hit that consist of a clique of size $m-i$, and therefore they will increase the size of the minimum hitting set by $m-i-1$ new elements.

The $m-2$ node will generate only one disjoint (from the previous cliques) unsatisfiable core. It will be $\{\overline{p_{m-2,m-1}} \vee \overline{p_{m-2,m}}, p_{m-2,m-1}, p_{m-2,m}\}$, and as before, it will add one element to the hitting set. The node $m-1$ will not generate disjoint unsatisfiable cores. It contains only the element $p_{m-1,m}$.

It is important to notice that the elements of the different cliques are completely disjoint. Therefore the minimum hitting set size so far (using $p_{i,j}$ variables only) is $(m-2) + \cdots + 1 = \frac{(m-1)(m-2)}{2}$. The hitting set is any set of elements that removes one element of every clique. Since each clique i has size $m-i$, it introduces $m-i-1$ new elements in the hitting set.

Step 2: In this step the algorithm will generate the unsatisfiable cores that relate elements of two different disjoint cliques, but it will not increase the size of the minimum hitting set. The unsatisfiable cores will be of type (b),

$$\{\overline{p_{i,j}} \vee \overline{p_{j,l}}, p_{i,j}, p_{j,l}\}, \tag{7}$$

where $1 \leq i < j < l \leq m$, or of type (c),

$$\{\overline{p_{i,j}} \vee \overline{p_{k,j}}, p_{i,j}, p_{k,j}\}, \tag{8}$$

where $1 \leq i < k < j < m$. Note here that $j < m$.

To deal with these two types of unsatisfiable cores, the algorithm will use an ordering and deal with all the unsatisfiable cores involving some $p_{i,j}$ together with each $p_{k,j}$ (for $i < k < j$) or $p_{j,l}$ (for $j < l \leq m$), before dealing with $p_{i,j+1}$ (or $p_{i+1,i+2}$ if $j+1 = m$). The ordering will be the following:

$$p_{1,2}, \ldots, p_{1,m-1}, p_{2,3}, \ldots, p_{2,m-1}, \ldots, p_{m-2,m-1}.$$

We will show the algorithm only for cores of type (b) as shown in (7). The other type is identical. Suppose now that the algorithm is considering the core $\{\overline{p_{i,j}} \vee \overline{p_{j,l}}, p_{i,j}, p_{j,l}\}$, and that at this point

$$\begin{aligned} hs \; = \; & \{p_{t,t+1}, \ldots, p_{t,m-1} : t \neq i, j \text{ and } 1 \leq t \leq m-2\} \\ & \cup \; \{p_{i,i+1}, \ldots, p_{i,j-1}, p_{i,j+1}, \ldots, p_{i,m}\} \\ & \cup \; \{p_{j,j+1}, \ldots, p_{j,l-1}, p_{j,l+1}, \ldots, p_{j,m}\}. \end{aligned} \tag{9}$$

Then if $l < m$, the new minimum hitting set will be

$$\begin{aligned} hs \;=\; & \{p_{t,t+1}, \ldots, p_{t,m-1} : t \neq i, j \text{ and } 1 \leq t \leq m-2\} \\ & \cup\ \{p_{i,i+1}, \ldots, p_{i,j-1}, p_{i,j+1}, \ldots, p_{i,m}\} \\ & \cup\ \{p_{j,j+1}, \ldots, p_{j,l}, p_{j,l+2}, \ldots, p_{j,m}\}, \end{aligned} \tag{10}$$

Note that hs covers all the cliques from Step 1, but excludes $p_{i,j}$ and $p_{j,l+1}$. Now the algorithm is able to find the unsatisfiable core $\{\overline{p_{i,j}} \vee \overline{p_{j,l+1}}, p_{i,j}, p_{j,l+1}\}$.

Step 2 uses this method to deal one-by-one with the unsatisfiable cores joining the cliques created in step 1, except unsatisfiable cores like $\{\overline{p_{i,m}} \vee \overline{p_{j,m}}, p_{i,m}, p_{j,m}\}$. The last unsatisfiable core should be $\{p_{m-2,m-1}, p_{m-1,m}\}$. At the end of step 2, we will have

$$hs = \{p_{i,j} : i \neq j \text{ and } 1 \leq i < j \leq m-1\}$$

as a minimal hitting set of size $(m-2)(m-1)/2$, the same size as in step 1. hs is a hitting set because for every unsatisfiable core of the type shown in (7) or (8), the two elements of the set are in hs if the vertex m is not mentioned, and if m is mentioned, one element of the set is in hs. The set is minimum because no smaller set would suffice to hit the unsatisfiable cores of step 1.

Step 3: In the last step, the algorithm will find unsatisfiable cores of the form

$$\{\overline{p_{i,m}} \vee \overline{p_{j,m}}, p_{i,m}, p_{j,m}\},$$

and the number of new elements that get introduced in hs is $\frac{m-1}{2}$. As in step 1, the algorithm will iteratively build a clique, this time with elements $p_{i,m}$ for every i. As a consequence, we will end up having $m-2$ elements $p_{i,m}$ in the minimum hitting set; but at the same time, some elements $p_{s,t}$ for $s, t < m$, that were in hs, now won't be there.

Suppose now the algorithm non-deterministically finds the unsatisfiable core

$$\{\overline{p_{1,m}} \vee \overline{p_{2,m}}, p_{1,m}, p_{2,m}\}.$$

In this case the algorithm introduces either $p_{1,m}$ or $p_{2,m}$ in hs. Even though we might take out another element from the set, the size of hs will have to increase. Suppose wlog that the algorithm introduces $p_{2,m}$ in hs. Then we might remove some $p_{2,l}$ from hs, given that the clique on the node 2 would continue having $m-2$ elements in hs. But then the algorithm must introduce $p_{l,m}$ in hs to be able to ensure that the unsatisfiable set $\{p_{2,l}, p_{l,m}\}$ has one of its elements in the hitting set. Therefore in either case, the minimum hitting set increases by 1, and we assume the new element is $p_{2,m}$.

Suppose the next unsatisfiable core is

$$\{\overline{p_{1,m}} \vee \overline{p_{3,m}}, p_{1,m}, p_{3,m}\}.$$

Now the minimum hitting set could be $\{p_{1,m}\} \cup \{p_{i,j} : \text{for all } i, j, 1 \leq i < j < m\}$. If the next unsatisfiable core is

$$\{\overline{p_{2,m}} \vee \overline{p_{3,m}}, p_{2,m}, p_{3,m}\},$$

then it will have produced a clique of size 3, but the size of hs will be as in the previous two cases of step 3. In this case, $p_{2,m}$ and $p_{3,m}$ are added to hs, and $p_{1,m}$ and $p_{2,3}$ are eliminated. Notice that by this change, the hitting set hs uses $m-2$ elements to cover the cliques on 1 and 2, and uses $m-1$ elements to cover the clique on 3, compensating for the fact that we have eliminated $p_{2,3}$.

In general, we can assume that when the algorithm builds the clique with the elements $\{p_{1,m}, \dots, p_{s,m}\}$, the minimum hitting set can have the elements

$$\{p_{i,j} : 1 \leq i < j < m\} \cup \{p_{2,m}, \dots, p_{s,m}\} \setminus \{p_{2,3}, p_{4,5} \dots, p_{s-1,s}\} \tag{11}$$

for s odd, and

$$\{p_{i,j} : 1 \leq i < j < m\} \cup \{p_{2,m}, \dots, p_{s,m}\} \setminus \{p_{2,3}, p_{4,5} \dots, p_{s-2,s-1}\} \tag{12}$$

for s even. So if s is odd, we have added $(s-1) - \frac{s-1}{2} = \frac{s-1}{2}$ to the size of hs from what it was at the end of step 2. And if s is even, we have added $(s-1) - \frac{s-2}{2} = \frac{s}{2}$ to the size of hs. Therefore, the size of the minimum hitting set increases by one only when we complete a clique of even size on the elements $p_{i,m}$.

Let us justify the previous statements. First, let us see that (11) and (12) are minimum hitting sets for the corresponding sets of unsatisfiable cores. Notice that for every $i < m$, the set of elements $p_{i,j}$ in hs, contains at least $m-2$ elements, and if s is odd, it contains exactly $m-2$ elements. Also the set of elements $p_{i,m}$ in hs contains exactly $s-1$ elements. It is an easy exercise to check that every unsatisfiable core contains one element in the hitting set. All this shows that the sets are hitting sets. In the case of (11), it also shows minimality, since by Observation 20, the set contains the minimum number of elements to be a minimum hitting set.

In the case of (12), the set of elements containing s has $m-1$ elements, one more than required. But in this case, if we eliminate $p_{s,m}$ from hs, we need to include $p_{1,m}$, and then the set of cliques containing 1 would have one more element than strictly necessary. Another option is to remove another element from the clique of s, say $p_{i,s}$, but then the clique on i would be missing one element. □

Theorem 26. *The basic MaxHS algorithm algorithm (Algorithm 2) is able to conclude in polynomial time that the dual-rail encoding of the parity principle must falsify at least* $\frac{m(m-1)}{2}+1$ *soft clauses, thus proving that the original parity principle is unsatisfiable.*

Proof. This follows from Theorem 24 and Theorem 25.

5. Dual-Rail MaxSAT Simulates Resolution

In this section we will show various simulations of the resolution proof system by different MaxSAT algorithms using the dual-rail encoding. First we show that core-guided MaxSAT (using the dual-rail encoding) simulates resolution. When we use MaxSAT resolution instead of the core-guided algorithm, we need somewhat stronger forms of MaxSAT resolution. If we only want to simulate tree-like resolution, we need multiple dual-rail encodings; but if we want to simulate full resolution, we need weighted dual-rail. On the other hand, we do not know of a simulation of resolution by MaxHS algorithms (using dual-rail).

5.1. Core-Guided MaxSAT Simulates Resolution.

In this section we show how core-guided MaxSAT algorithms with the dual-rail encoding simulate full resolution.

Theorem 27. *Core-guided MaxSAT with the dual-rail encoding p-simulates general resolution.*

Proof. Let $\mathcal{R}$ be a resolution refutation of Γ over the variables $x_1, \ldots, x_n$. We will produce a core-guided MaxSAT refutation of Γ^{dr}.

The first n iterations of the core-guided procedure will be independent of $\mathcal{R}$. From the soft clauses n_i and p_i, and the hard clause $\overline{p_i} \vee \overline{n_i}$ (for $1 \leq i \leq n$), we obtain $\perp$ using the resolution rule. We obtain n empty clauses $\perp$ and the unsatisfiable cores contain the soft clauses $\{n_i, p_i\}$ (for $1 \leq i \leq n$) which are disjoint. The soft clauses in the core are substituted by a new set of hard clauses $\{p_i \vee a_i, n_i \vee a'_i, \neg a_1 \vee \neg a'_i\}$ using new variables a_i and a'_i. The last clause is equivalent to $a_i + a'_i \leq 1$. So far, we have obtained n empty clauses, and we haven't yet used the resolution refutation of Γ.

The resulting set of clauses so far is unsatisfiable. In the next iteration of the core-guided MaxSAT algorithm, the SAT solver is called, and a possible execution of the SAT solver simulates the following resolution refutation. First, the hard clauses corresponding to the clauses of Γ are transformed to have only p_i variables. For every $i, 1 \leq i \leq n$, resolving $p_i \vee a_i$ against $\neg a_i \vee \neg a'_i$, we obtain $p_i \vee \neg a'_i$. Resolving this last clause with $n_i \vee a'_i$, we obtain $n_i \vee p_i$. At this point we can eliminate all the occurrences of $\neg n_i$ from the set of hard clauses, by resolving each clause $C \vee \neg n_i$ with $n_i \vee p_i$, and obtaining $C \vee p_i$. Now we have an unsatisfiable set of (soft) clauses on variables $p_1, \ldots, p_n$, equivalent to Γ. We get one final $\perp$ by replicating the resolution refutation $\mathcal{R}$ using p_i variables instead of x_i variables. □

5.2. Multiple Dual-Rail MaxSAT Simulates Tree-like Resolution.

We start with an observation that will be useful for the simulations in the following subsections. The definition of multiple dual-rail MaxSAT can be found at the end of Section 3.1.

Observation 28. *The dual-rail encodings include soft unit clauses p_i and n_i and hard clauses $\overline{p_i} \vee \overline{n_i}$. Applying a MaxSAT inference to p_i and $\overline{p_i} \vee \overline{n_i}$ yields the two soft clauses $\overline{n_i}$ and $p_i \vee n_i$. Combining $\overline{n_i}$ and n_i with a MaxSAT inference yields the clause $\perp$. Thus, we have used up the soft clauses p_i and n_i and obtained one instance of $\perp$ plus the soft clause $p_i \vee n_i$.*

Theorem 29. *Multiple dual-rail MaxSAT resolution simulates tree-like resolution.*

Proof. Let $\mathcal{R}$ be a tree-like resolution refutation of Γ over the variables $x_1, \ldots, x_n$. Let k_i be the number of times that x_i is resolved on in $\mathcal{R}$. We form Γ^{mdr} by adding the soft clauses p_i and n_i with multiplicity k_i, and the hard clauses $\overline{p_i} \vee \overline{n_i}$. (This is permitted as the values k_i correspond to the weights w_i of a weighted dual-rail MaxSAT refutation.) Set $K = \sum_i k_i$. By the above Observation 28, from these clauses there is a MaxSAT derivation of K many instances of $\perp$, plus the soft clauses $p_i \vee n_i$ with multiplicity k_i.

We modify the refutation $\mathcal{R}$. For each clause A in Γ, let A^{dr} be the result of replacing members x_i with $\overline{n_i}$, and members $\overline{x_i}$ with $\overline{p_i}$. An inference in Γ resolving $x_i \vee A$ and $\overline{x_i} \vee B$ to obtain $A \vee B$ becomes

$$\frac{\overline{n_i} \vee A^{\mathrm{dr}} \qquad \overline{p_i} \vee B^{\mathrm{dr}}}{A^{\mathrm{dr}} \vee B^{\mathrm{dr}}}$$

To make this a valid MaxSAT inference, first resolve $\overline{n_i} \vee A^{\mathrm{dr}}$ against an available soft clause $p_i \vee n_i$ to obtain the soft clause $p_i \vee A$ plus some additional clauses. A further MaxSAT inference resolves this against $\overline{p_i} \vee B^{\mathrm{dr}}$ to obtain $A^{\mathrm{dr}} \vee B^{\mathrm{dr}}$ plus some additional clauses. Continuing this process yields a valid MaxSAT refutation of $\perp^{\mathrm{dr}}$, i.e. of $\perp$. This gives a total of $K + 1$ clauses $\perp$ as desired. ☐

Note the proof works as long as k_i is greater than or equal to the number of times x_i is resolved on. For applications, this means it is only needed to have an upper bound on the number of resolutions on x_i; for instance, taking k_i equal to the total number of inferences in $\mathcal{R}$ certainly works.

5.3. Weighted Dual-Rail MaxSAT Simulates Resolution.

The definition of weighted dual-rail MaxSAT can be found at the end of Section 3.1.

Theorem 30. *Weighted dual-rail MaxSAT resolution simulates general resolution.*

Proof. Let $\mathcal{R}$ be a resolution refutation of Γ containing clauses $C_1, \ldots, C_m$. Each C_i is either an initial clause from Γ or is derived from two clauses C_{j_1} and C_{j_2}, where $j_1 < j_2 < i$. We define a directed graph $G = ([m], E)$ encoding the dependencies in the derivation. The set of vertices of G is $\{1, \ldots, m\}$, corresponding to the m clauses of $\mathcal{R}$. The edges are based on inference rules; E is the set of directed edges (j, i) such that C_j is a hypothesis of the resolution inference introducing C_i. Thus, the vertex m (corresponding to the empty clause C_m) is a sink of G. The sources in G correspond to initial clauses in Γ. All other vertices in G have in-degree two. Since $\mathcal{R}$ is not assumed to be tree-like, the out-degrees can be greater than one.

We must assign to each clause $C_i \in \mathcal{R}$ a weight $w_i \in \mathbb{N}$. These weights give the weights k_i needed for the soft clauses n_i and p_i when we construct a weighted dual-rail MaxSAT refutation of Γ. The last clause C_m is the final™¡ $\perp$ derived for the MaxSAT refutation: this clause has weight one, $w_m = 1$. For all $j < m$, define

$$w_j \;=\; \textstyle\sum_{(j,i)\in E} w_i.$$

This is the same as defining w_j to be the sum of the weights of the clauses which are inferred directly from C_j.

Recall the Fibonacci numbers $F_1 = F_2 = 1$ and $F_i = F_{i-1} + F_{i-2}$ for $i > 2$. The next lemma depends only on the fact that $G = ([m], E)$ has in-degree 0 or 2 at every node, and that the directed edges respect the usual ordering of $[m]$.

Lemma 31. *$w_i \le F_{m+1-i}$. Thus $w_i < \phi^m/\sqrt{5}$ where ϕ is the golden ratio.*

Proof. Since every clause has in-degree two in G, this lemma is intuitively obvious; we sketch a proof nonetheless for completeness. For this, it will suffice to prove the weights w_j are collectively maximized provided that for every $i > 2$, the two edges $(i-1, i)$ and $(i-2, i)$ are in E. Define $i \in [m-2]$ to be *out-good* if its only outgoing edges are $(i, i+1)$ and $(i, i+2)$; and define $m-1$ to be *out-good* if its only out-going edge is $(m-1, m)$. Clearly if all $i \in [m-1]$ are out-good then each $w_i = F_{m+1-i}$; this is proved using induction on $m + 1 - i$ (that is, by induction with i ranging from m to 1).

Without loss of generality, every $j > 2$ has in-degree 2, not 0, since otherwise, we could add two incoming edges to j and this will increase the weights.

Suppose i_0 is the least i which is not out-good. Since i_0+1 and i_0+2 have in-degree two, and by choice of i_0, the edges (i_0, i_0+1) and (i_0, i_0+2) are both present. Suppose there is an edge (i_0, j) with $j > i_0+2$. Since the in-degree of j is 2, there is at least one of i_0+1 and i_0+2 which is not an immediate predecessor of j; denote this non-predecessor i_1. We create a set of edges E' from E by removing the edge (i_0, j) and adding the edge (i_1, j). This modifies the weights w_i to new values w'_i. Clearly $w'_i = w_i$ for $i > i_1$. And, $w'_{i_1} = w_{i_1} + w_j > w_{i_1}$. Thus, for $i_0 < i \le i_1$, we have $w'_i \ge w_i$. It follows that $w'_{i_0} \ge w_{i_0}$. Once all such edges j are handled, i_0 is out-good. Therefore, we have created one new out-good vertex, increased at least one weight, and did not decrease any weights. Proceeding inductively proves the lemma. ◻

To finish the proof of Theorem 30, we also need to fix weights k_i for the variables x_i. Set k_i to be equal (or be greater than) the sum of the weights w_j of clauses C_j which are introduced by a resolution on x_i. By Lemma 31, $k_i \le \sum_{i=1}^{m-2} \phi^i < \phi^{m-1}$, so $k_i = 2^m$ is always sufficient. Now Theorem 30 can be proved with the essentially the same construction as Theorem 29. A clause C_ℓ in $\mathcal{R}$ becomes the weighted clause $(C_\ell^{\rm dr}, w_\ell)$ in $\mathcal{R}^{\rm wdr}$. If C_ℓ is equal to $A \vee B$ and is derived from $x_i \vee A$ and $\overline{x_i} \vee B$, then in $\mathcal{R}^{\rm wdr}$, it becomes the (not-yet-valid) inference

$$\frac{(\overline{n_i} \vee A^{\rm dr}, w_\ell) \qquad (\overline{p_i} \vee B^{\rm dr}, w_\ell)}{(A^{\rm dr} \vee B^{\rm dr}, w_\ell)} \tag{13}$$

Note the weights of all three clauses are equal to w_ℓ. As described below, this is arranged for the two hypotheses by earlier extraction inferences. In $\mathcal{R}^{\text{wdr}}$, the "inference" (13) is replaced by two MaxSAT resolution inferences which resolve against the weighted soft clauses (n_i, w_ℓ) and (p_i, w_ℓ) and the hard clauses $(\overline{n_i} \vee \overline{p_i}, \top)$.

$\mathcal{R}^{\text{wdr}}$ needs inferences to fix up the weights. For $i \leq n$, let $C_{\ell_1}, \ldots, C_{\ell_s}$ be the clauses which are inferred by resolving on x_i, so $k_i \geq \sum_j w_{\ell_j}$. At the start of $\mathcal{R}^{\text{wdr}}$, from the initial soft clauses (n_i, k_i) and (p_i, k_i), extraction rules are used to derive all the clauses (n_i, w_{ℓ_j}) and (p_i, w_{ℓ_j}). Similarly, let $C_{\ell_1}, \ldots, C_{\ell_s}$ now denote clauses which are derived by resolution using C_ℓ, so $w_\ell = \sum_j w_{\ell_j}$. Extraction inferences are used to derive all of the clauses (C_ℓ, w_{ℓ_j}) from (C_ℓ, w_ℓ). These clauses are used as hypotheses of later inferences similarly as was done for (13). $\square$

Note that k_i can be upper bounded by $\sum_{i=1}^{m-2} \phi^i < \phi^{m-1}$. As before, the proof of Theorem 30 works as long as the k_i's are sufficiently large.

6. Dual-Rail MaxSAT does not Simulate Cutting Planes

The primary result of the present section (Theorem 32) is that the dual-rail MaxSAT resolution proof system can be polynomially simulated by the constant depth Frege proof system augmented with the schematic pigeonhole principle.

It is known that the proof system of constant depth Frege augmented with the schematic pigeonhole principle, denoted AC^0-Frege+PHP, requires exponential size to prove the parity principles [1, 11]. Therefore, we obtain as an immediate corollary that MaxSAT resolution refutations of the dual-rail encoded parity principle require exponential size. Additionally, the dual-rail MaxSAT resolution proof system does not polynomially simulate the Cutting Planes proof system.

Theorem 32. AC^0-*Frege+PHP p-simulates the dual-rail MaxSAT resolution system. More precisely, there is a constant d_0 and a polynomial $p(s)$ so that the following holds. If Γ is a set of clauses and Γ^{dr} has a MaxSAT resolution refutation of size s, then Γ has a depth d_0 Frege refutation from instances of the PHP of size $p(s)$.*

The value of d_0 depends on the exact definitions of the Frege system (e.g., with modus ponens, or with the sequent calculus, etc.) and of depth; however, d_0 is small, approximately equal to 3. In particular, the Frege proof uses instances of PHP which are obtained by substituting depth one formulas (either conjunctions or disjunctions of literals) for the variables $z_{i,j}$ of a pigeonhole formula.

It is open whether the theorem holds for the dual-rail MaxSAT system generalized to allow arbitrary (binary-encoded) weights.

Proof. We prove Theorem 32. Let Γ be an unsatisfiable set of clauses in the variables $x_1, \ldots, x_N$. Its dual-rail encoding Γ^{dr} uses the variables n_i and

p_i for $i \in [N]$. By hypothesis, there is a MaxSAT resolution derivation $\mathcal{D}$ of $N+1$ many empty clauses $\bot$ from Γ^{dr}. Our goal is to give a AC^0-Frege+PHP refutation of Γ; this refutation involves only the variables x_i. The intuition for forming the AC^0-Frege proof is that we assume that Γ is satisfied by (the assignment of truth values to) the variables $x_1, \ldots, x_N$, and use the refutation $\mathcal{D}$ to define a contradiction to the pigeonhole principle. In other words, we argue that a polynomial size AC^0-Frege can use the formulas in Γ as hypotheses to derive a contradiction to the pigeonhole principle. This contradiction will be defined using clauses $P_{\alpha,\beta}$ (defined below) involving the variables x_i, where α, β will range over vertices of a bipartite graph; the AC^0-Frege proof will argue that these clauses define a contradiction to the pigeonhole principle.

The MaxSAT refutation $\mathcal{D}$ has size s and contains $m < s$ inferences. The j-th inference of $\mathcal{D}$ has the form

$$\frac{l \vee A \qquad \bar{l} \vee B}{A \vee B \qquad l \vee A \vee \overline{B} \qquad \bar{l} \vee \overline{A} \vee B} \tag{14}$$

for l a literal. Here, $l \vee A \vee \overline{B}$ and $\bar{l} \vee \overline{A} \vee B$ denote sets of zero or more clauses, which depend on orderings of the literals in A and in B. W.l.o.g., $\mathcal{D}$ is annotated with information about which clauses are used in the j-th inference including the orderings on the literals of A and B.

Let $\mathcal{D}_j$ be the multiset of clauses which are available for use in $\mathcal{D}$ after the j-th inference. Thus, $\mathcal{D}_0$ is the same as Γ^{dr}. The multiset D_{j+1} is obtained from $\mathcal{D}_j$ by removing the hypotheses of the j-th inference (14) and adding its conclusions. Since $\mathcal{D}$ is a valid MaxSAT refutation, the final set $\mathcal{D}_m$ contains $N+1$ many empty clauses $\bot$. Two extra sets $\mathcal{D}_{-1}$ and $\mathcal{D}_{m+1}$ are defined by letting $\mathcal{D}_{-1}$ contain the N unit clauses $x_1, \ldots, x_N$ and letting $\mathcal{D}_{m+1}$ be the multiset containing $N+1$ copies of the empty clause $\bot$.

Let $\mathcal{D}_*$ denote the disjoint union of the multisets D_j for $-1 \leq j \leq m{+}1$. Members of the multiset $\mathcal{D}_*$ are denoted (C, j) indicating that C is a member of $\mathcal{D}_j$. If there are multiple occurrences of C in $\mathcal{D}_j$, then there are multiple occurrences of (C, j) in $\mathcal{D}_*$. We will assume that multiple occurrences are correctly tracked with each "C" labeled as to which occurrence it is, but suppress this in the notation. In other words, C is a particular occurrence of a clause in D_j. (Strictly speaking, we should write something like (C, i, j) to indicate that C is the i-th occurrence of C in $\mathcal{D}_j$, but we prefer to keep the notation simple and do not do this.)

Let S be the cardinality of $\mathcal{D}_*$, so $S = s^{O(1)}$. Define

$$T = \textstyle\bigcup_{0 \leq i \leq m+1} \mathcal{D}_i \qquad \text{and} \qquad U = \textstyle\bigcup_{-1 \leq i \leq m} \mathcal{D}_i.$$

We have $|T| = S - N$ and $|U| = S - N - 1$, so $|T| = |U| + 1$. We wish to define a total and injective function $f : T \to U$, based on the assumption that $x_1, \ldots, x_N$ specify a satisfying assignment for Γ: this will contradict the pigeonhole principle. The function f will be defined by giving formulas $P_{\alpha,\beta}$ (involving the variables $x_1, \ldots, x_n$) that define the graph of f. For this, we define formulas $P_{\alpha,\beta}$ for each $\alpha = (C, j) \in T$ and each $\beta = (C', j') \in U$ which define the condition that $f(\alpha) = \beta$. Again, these formulas $P_{\alpha,\beta}$ will involve the variables $x_1, \ldots, x_N$.

If $(C, j) \in U$, then C is a clause (possibly empty) involving only the variables n_i and p_i. We wish to identify p_i and n_i with x_i and $\overline{x_i}$ to evaluate the truth of C. Accordingly, define $X(C)$ to the the formula obtained by replacing the literals p_i and $\overline{n_i}$ with x_i, and the literals $\overline{p_i}$ and n_i with $\overline{x_i}$. If C contains both p_i and n_i (or both $\overline{p_i}$ and $\overline{n_1}$) for some i, then $X(C)$ becomes a tautologous clause and can be treated as the constant $\top$.

We next give the definition of the function f and define the formulas $P_{\alpha,\beta}$. Let α be (C, j) and β be (C', j'). The intuition is that if $X(C)$ is true, then $f(\alpha) = \alpha$; and if $X(C)$ is false then $f(\alpha) = \beta$ exactly when $j' = j - 1$ and C' is the formula in $\mathcal{D}_j$ which corresponds to C under the application of the j-th inference of $\mathcal{D}$ and thus has $X(C')$ false. More formally:

1. Suppose $j = m + 1$, so C is an empty clause $\perp$ in the "extra" set $\mathcal{D}_{m+1}$. We arbitrarily order the members $\perp$ of $\mathcal{D}_{m+1}$ and $\mathcal{D}_m$. Suppose C is the ℓ-th member of $\mathcal{D}_{m+1}$. We wish to assign $f(\alpha)$ to equal the ℓ-th $\perp$ in $\mathcal{D}_m$. Accordingly, $P_{\alpha,\beta}$ is the constant $\top$ (true) if and only if $j' = j-1 = m$ and C' is the ℓ-th $\perp$ in $\mathcal{D}_m$. Otherwise, $P_{\alpha,\beta}$ is the constant $\perp$ (false).

2. Suppose $j \geq 1$, and that C, as a member of $\mathcal{D}_j$, is not a clause in the conclusion of the j-th inference (14). The idea is that if C is true, then $f(\alpha) = \alpha$, and if C is false, then $f(\alpha) = \beta$ provided $j' = j - 1$ and C' is the same formula as C, namely the occurrence of the clause in $\mathcal{D}_{j-1}$ which corresponds to C. More formally, $P_{\alpha,\alpha}$ is the formula $X(C)$. And, if $j' = j-1$ and $C' \in \mathcal{D}_{j-1}$ is the corresponding occurrence of the clause C in $\mathcal{D}_{j-1}$, then $P_{\alpha,\beta}$ is the formula $\neg X(C)$. In all other cases, $P_{\alpha,\beta}$ is $\perp$.

3. Suppose $j \geq 1$, and C is one of the conclusions of the j-th inference (14). The idea is that if C is true, then $f(\alpha) = \alpha$, and if C is false, then $f(\alpha) = \beta$ provided $j' = j - 1$ and C' is the false hypothesis of (14). More formally, $P_{\alpha,\alpha}$ is the formula $X(C)$. And, if $j' = j-1$ and $C' \in \mathcal{D}_{j-1}$ is one of the hypotheses of (14), then $P_{\alpha,\beta}$ is the formula $\neg X(C) \wedge \neg X(C')$, which is a conjunction of literals. (This can make $P_{\alpha,\beta}$ false by virtue of containing both ℓ and $\overline{\ell}$.) In all other cases, $P_{\alpha,\beta}$ is $\perp$.

4. Suppose $j = 0$ and C is a hard clause of Γ^{dr} in $\mathcal{D}_0$. Assuming Γ is satisfied by $x_1, \ldots, x_N$, C is true; the idea is that $f(\alpha) = \alpha$. Accordingly, $P_{\alpha,\alpha}$ is the clause $X(C)$. For all other β, $P_{\alpha,\beta}$ is $\perp$.

5. Finally suppose $j = 0$ and C is a soft unit clause in Γ^{dr}, i.e. either p_i or n_i. The intuition is again that $f(\alpha) = \alpha$ if C is true. Otherwise $f(\alpha) = (x_i, -1)$. Formally, $P_{\alpha,\alpha}$ is $X(C)$. And, for $\beta = (x_i, -1)$, $P_{\alpha,\beta}$ is $\neg X(C)$. For all other β, $P_{\alpha,\beta}$ is $\perp$.

The formulas $P_{\alpha,\beta}$ are linear size and depth one, either conjunctions or disjunctions of literals. We must argue there are constant depth Frege proofs of the injectivity conditions

$$\neg P_{\alpha,\beta} \vee \neg P_{\alpha',\beta} \qquad \text{for all } \alpha \neq \alpha' \in T \text{ and all } \beta$$

and of the totality conditions

$$\bigvee_{\beta \in U} P_{\alpha,\beta} \qquad \text{for all } \alpha \in T.$$

The injectivity conditions are easy to check since so many $P_{\alpha,\beta}$'s are the constant $\perp$. First, suppose that $\alpha = (C, j)$ and $\alpha' = (C', j)$ where C and C' are two of the conclusions of the j-th inference (14). By inspection, C and C' contain a clashing literal; thus they cannot both be false. It follows that at least one of $P_{\alpha,\beta}$ or $P_{\alpha',\beta}$ is false. Obviously this fact, $\neg P_{\alpha,\beta} \vee \neg P_{\alpha',\beta}$, is easily provable in AC^0-Frege in this case. A similar, even simpler, argument works when $\alpha = (p_i, 0)$ and $\alpha' = (n_i, 0)$. The injectivity conditions for all other α, α', β are trivial.

There are only a couple non-trivial cases to check for the provability of the totality conditions. The first case is when $\alpha = (C, j)$ is the conclusion of the j-th inference (14). For this, we must argue that if $X(C)$ is false, then (14) has a hypothesis C' that has $X(C')$ false. This is completely trivial to prove with a constant depth Frege proof, since either (a) one of the hypotheses is a sub-clause C' of C so $X(C')$ is a sub-clause of $X(C)$ and thus $X(C')$ is false, or (b) C is $A \vee B$ in (14) and since $X(\ell)$ is either false or true and C' is either the first or second hypothesis (respectively, based on the truth value of ℓ). The second non-trivial case to check for totality is the case where $\alpha = (C, 0)$ with C one of the hard clauses in Γ^{dr}. In this case, $P_{\alpha,\alpha}$ holds only if $X(C)$ is true. However, $X(C)$ is a member of Γ, and hence $X(C)$ holds under the assumption that $x_1, \ldots, x_N$ satisfy the clauses of Γ.

The above obtained a contradiction to the pigeonhole principle from the assumption that the clauses of Γ are true. The argument can be formalized in constant depth Frege; hence AC^0-Frege+PHP refutes Γ. By construction, the AC^0-Frege+PHP refutation is polynomial size in s. $\square$

Notice that in the proof of Theorem 32, every pigeon can only go to at most three holes. The following corollary answers the question of whether this restricted principle is as hard as the usual pigeonhole principle for AC^0-Frege.

Corollary 33. *The pigeonhole principle where every pigeon can only go to a constant number $d \geq 3$ of holes, requires exponential size proofs in AC^0-Frege.*

Proof. By the proof of Theorem 32, if AC^0-Frege has sub-exponential size proofs of PHP when every pigeon can only go to a three holes, then AC^0-Frege simulates dual-rail MaxSAT. Since dual-rail MaxSAT has polynomial size refutations of the pigeonhole principle, AC^0-Frege has them too. This is impossible by the known lower bounds of PHP in AC^0-Frege [1, 11]. Therefore the corollary follows. $\square$

Corollary 34. *MaxSAT resolution refutations of the dual-rail encoded parity principle require exponential size 2^{n^ϵ} for some $\epsilon > 0$.*

Proof. Corollary 34 follows from Theorem 32 since [11] and [65], building on [1], showed that AC^0-Frege+PHP refutations of Parity_n require size 2^{n^ϵ} for some $\epsilon > 0$. □

Corollary 35. *The dual-rail MaxSAT resolution proof system does not polynomially simulate* CP *or even* CP*.

Proof. Corollary 35 follows from Corollary 34 since it is easy to give polynomial size CP* (Cutting Planes proof system with polynomially bounded coefficients) proofs of the parity principle [39]. □

7. Experiments

This section evaluates the power of the dual-rail based MaxSAT solving and aims at confirming the theoretical claims of the paper. A thorough experimentation is presented testing modern SAT and MaxSAT solvers, as well as solutions based on mixed integer programming (MIP). We consider several benchmark sets encoding hard combinatorial principles: pigeonhole principle formulas, doubled pigeonhole principle, mutilated chessboard formulas [52, 42, 58, 59], parity principle, Urquhart formulas [72] and their combination. The evaluation comprises extensions of the results presented in earlier works [38, 17], as well as novel contributions. The evaluation shows clear performance gains provided by the dual-rail problem transformation and the follow-up MaxSAT solving.

7.1. Experimental Setup

In the evaluation, a large number of solvers were tested. However, the discussion below 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. The families of the evaluated solvers as well as the chosen representatives for the families are listed in Table 4. 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[7], e.g. MSCG (*mscg*), OpenWBO16 (*wbo*) and WPM3 (*wpm3*), as well as the recent MaxSAT solver Eva500a (*eva*) based on MaxSAT resolution.

The other competitor considered is CPLEX (*lp*). Three configurations of CPLEX were tested: (1) the default configuration and the configurations used in (2) MaxHS (*maxhs*) and (3) LMHS (*lmhs*). Given the overall performance, we decided to present the results for one best performing configuration, which

[7] https://maxsat-evaluations.github.io/

Table 4: Families of solvers considered in the evaluation (their best performing representatives are written in *italics*). *SAT+* stands for SAT strengthened with additional 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.

SAT		**SAT+**		**IHS MaxSAT**		**CG MaxSAT**			**MRes**	**MIP**
minisat	*glucose*	*lgl*	crypto	*maxhs*	*lmhs*	*mscg*	*wbo*	wpm3	*eva*	*lp*
[30]	[8]	[13, 15]	[70, 69]	[27, 28, 29]	[67]	[56]	[51]	[5]	[57]	[37]

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 dual-rail encoded formulas (*lp-wcnf*).

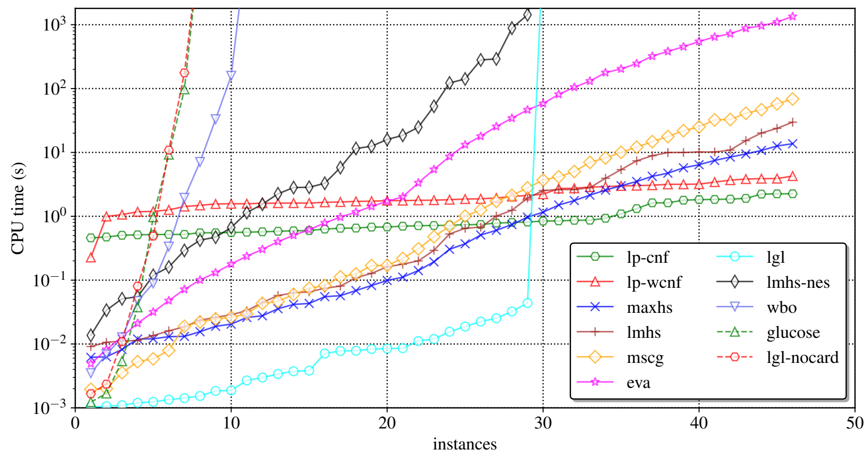
Regarding the IHS-based MaxSAT solvers, both MaxHS and LMHS implement the *Eq-Seeding* constraints [28]. Given that all soft clauses constructed by the proposed HornMaxSAT transformation are *unit* and that the set of all variables of HornMaxSAT formulas is *covered* by the soft clauses, these eq-seeding 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.

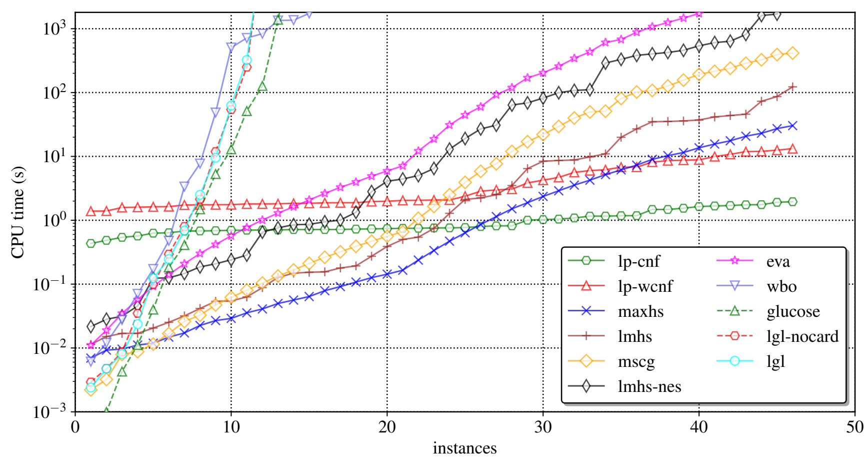
7.2. Pigeonhole Principle benchmarks

This first set of experiments is supposed to assess thoroughly the performance among all the considered families of solvers with the pigeonhole formulas (PHP) [26]. 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 [68] (*PHP-sc*). Each of the families contains 46 CNF formulas encoding the pigeonhole principle with the number of pigeons varying from 5 to 100. Figure 4[9] shows the performance of the solvers 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 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.

[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 scale the Y axis logarithmically.



(a) PHP-pw (pairwise)



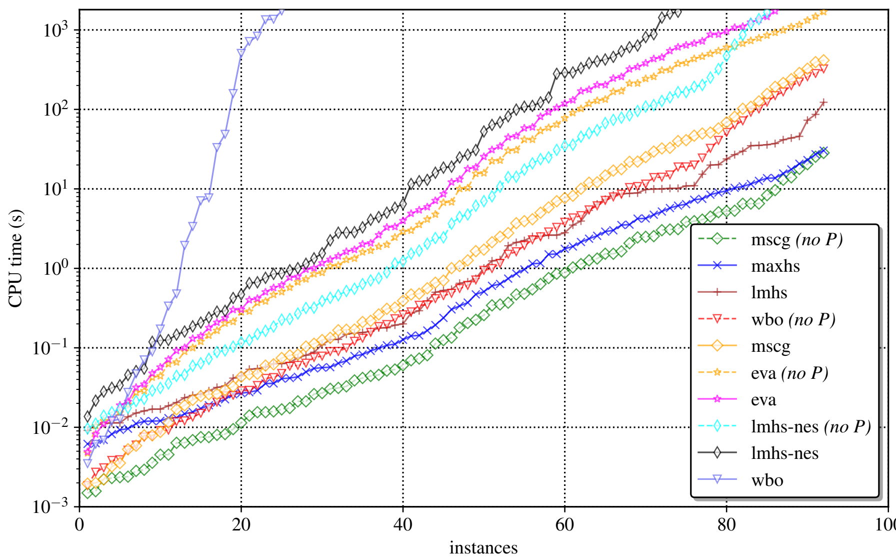
(b) PHP-sc (sequential counter)

Figure 4: Performance of the considered solvers on pigeonhole formulas.

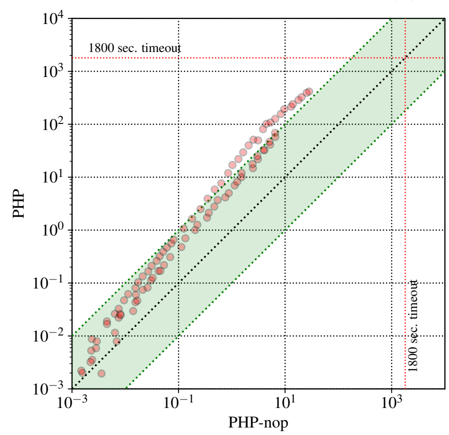
7.2.1. On discarding $\mathcal{P}$ clauses

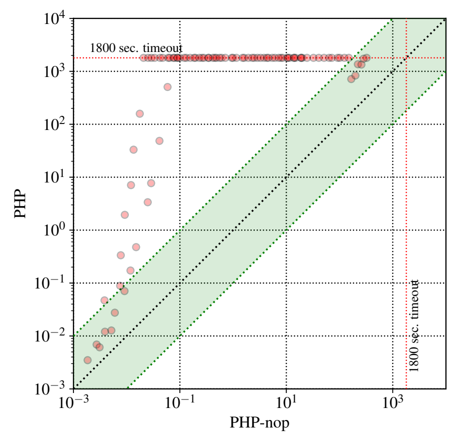
As described above, given a CNF formula over variables X, its dual-rail MaxSAT encoding contains hard $\mathcal{P}$ clauses, namely clauses of the form $(\overline{p_i} \vee \overline{n_i})$ for each variable $x_i \in X$, among other clauses. Observe that the polynomial upper bounds for PHP formulas in Theorems 8, 11 and 18 for all three of the dual-rail systems core-guided MaxSAT, MaxHS-like MaxSAT, and MaxSAT resolution are obtained without using the $\mathcal{P}$ clauses. Also, note that other ways of refuting PHP in dual-rail MaxSAT exist, e.g. those replicating non-polynomial resolution refutations. Such refutations involve getting trivial unsatisfiable cores comprising triples of clauses of the form $(\overline{p_i} \vee \overline{n_i}, \top)$, $(p_i, 1)$, and $(n_i, 1)$.

Since there is generally no control of what unsatisfiable cores a MaxSAT



(a) Cactus plot





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

Figure 5: Performance of MaxSAT solvers on PHP-pw $\cup$ PHP-sc w/ and w/o $\mathcal{P}$ clauses.

solver computes, our conjecture is that in some cases the presence of the $\mathcal{P}$ clauses in the formula can be harmful for a MaxSAT solver as they may confuse it to go in a "wrong direction", i.e. by computing these trivial unsatisfiable cores. This may result in hampering the overall performance of a MaxSAT solver in some cases. To confirm this conjecture, we also considered both PHP-pw and PHP-sc instances *without* the $\mathcal{P}$ clauses. Figure 5 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 $\mathcal{P}$ clauses (except

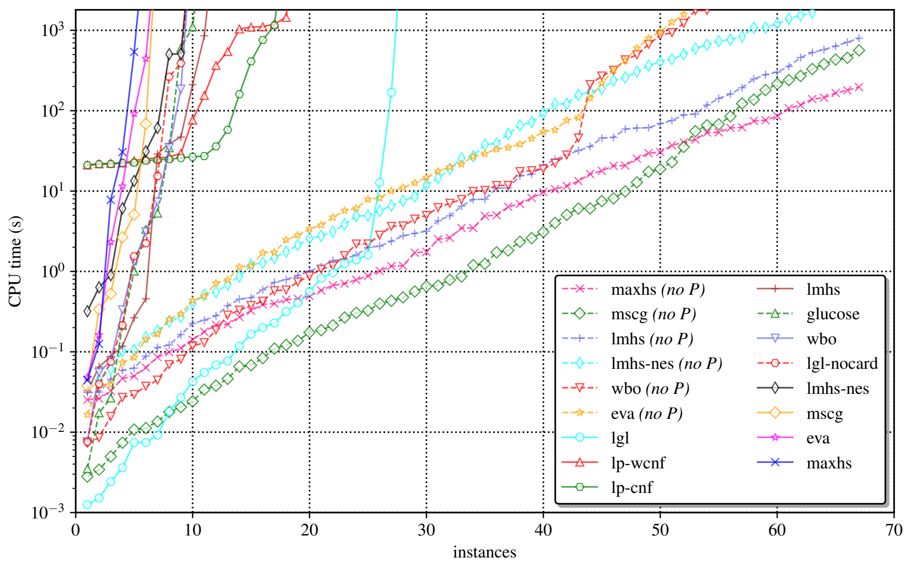


Figure 6: Performance of SAT and MaxSAT solvers on "doubled" pigeonhole formulas.

maxhs and *lmhs* whose performance is not affected by removal of $\mathcal{P}$). As can be observed, the $\mathcal{P}$ clauses can indeed hamper a solver's ability to get a sequence of *good* unsatisfiable cores, which affects its performance. For instance, as detailed in Figure 5c, the efficiency of *wbo* is improved by a few orders of magnitude if the $\mathcal{P}$ clauses are discarded. Also, as shown in Figure 5b, *mscg* gets about an order of magnitude performance improvement outperforming all the other solvers.

Note that although discarding the $\mathcal{P}$ clauses can be done for the PHP formulas due to the existence of a correct refutation ignoring them, in general discarding them completely can lead to incorrect MaxSAT solutions. The reason is that some formulas have to be refuted necessarily by exploiting (a subset of) the $\mathcal{P}$ clauses. Given the above, one can envision a possible strategy to solve problems (in general) without considering the $\mathcal{P}$ clauses at the beginning, and then adding them on demand, as deemed necessary to block non-solutions. The operation is similar to the well-known counterexample-guided abstraction refinement paradigm (CEGAR) [23].

7.3. Doubled Pigeonhole Principle

This section aims at comparing the performance of the state-of-the-art SAT and MaxSAT solvers with respect to the "doubled" pigeonhole formulas. More concretely, two sets of $2\mathrm{PHP}_m^{2m+1}$ formulas were considered encoding $AtMost2$ constraints by (1) *triplewise* encoding (*PHP-tw*) as studied earlier in this paper and (2) sequential counters [68] (*PHP-sc*), i.e. with the use of auxiliary variables [71]. The former set contains $2\mathrm{PHP}_m^{2m+1}$ formulas for $m \in \{5, \ldots, 25\}$[10] while the latter one contains instances for $m \in \{5, \ldots, 100\}$. The total number of instances in both 2PHP benchmark sets is 67.

Figure 6 depicts the performance of the considered competitors on the total set of 2PHP benchmarks consisting of both 2PHP-tw and 2PHP-sc instances. As expected, SAT solvers with no additional reasoning can only deal with $2\mathrm{PHP}_m^{2m+1}$ for $m \leq 7$ given 1800s timeout (*lgl* performs better and solves 27 instances in total). Surprisingly, MaxSAT solvers do not perform *much* better being able to deal with $m \leq 15$ if the $\mathcal{P}$ clauses are present in the formula. Given the fact of existence of a short dual-rail based MaxSAT proof for 2PHP, this comes as another evidence that the $\mathcal{P}$ clauses can prevent MaxSAT solvers to compute *good* unsatisfiable cores, which affects their overall performance. Indeed, our results confirm this as the performance of all MaxSAT solvers gets tremendously increased when clauses $(\overline{p_i} \vee \overline{n_i}, \top)$ are discarded[11]. In particular, *maxhs*, *lmhs*, as well as *mscg* can solve all the considered instances (for m up to 100) with the "harmful" clauses being discarded while *lmhs-nes*, *eva* and *wbo* are a few instances behind. Observe that the performance of *maxhs* and *lmhs* is affected by the presence of the $\mathcal{P}$ clauses, which was not the case for PHP formulas studied in Section 7.2. This seems a little bit surprising provided that PHP and 2PHP formulas share a common structure. We believe that a deeper understanding of the principles underlying the IHS-based MaxSAT solving could shed light on this phenomenon. Finally, another surprise is that *lp-cnf* and *lp-wcnf* have a hard time dealing with 2PHP formulas, which is in clear contrast to the case of PHP.

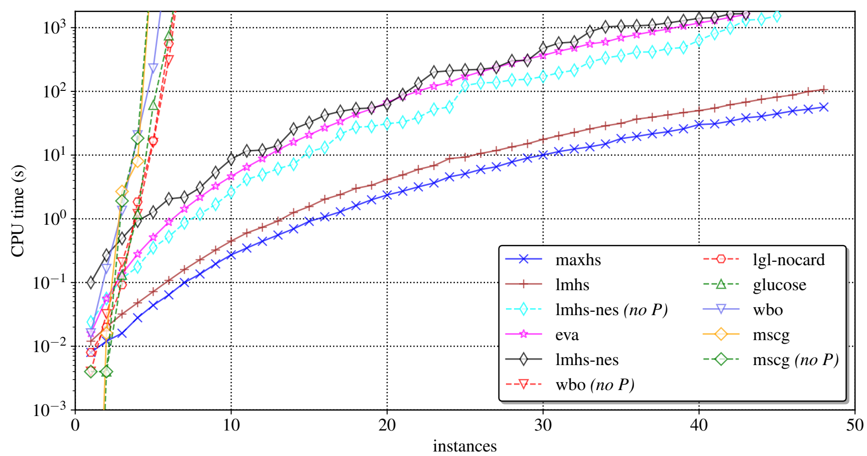
7.4. Mutilated Chessboard

This section targets assessing the practical efficiency of dual-rail MaxSAT compared to modern SAT solvers for the mutilated chessboard principle formulas (*CB*) [52, 42], which is known to be hard for resolution [3].

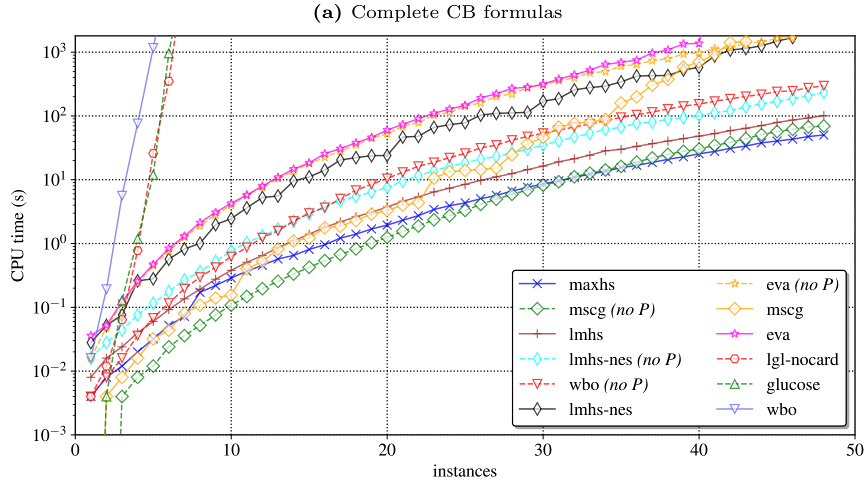
The benchmark formulas considered here encode the mutilated chessboard principle for chessboard of size $2n \times 2n$ with n being varied from 3 to 50 inclusively. The encoding is standard and follows [3]. Thus, the total number of formulas is 48. Recall that the standard encoding of CB is "redundant" in the following sense: after removing two corner squares of the chessboard (let us assume those are white), the principle forces mappings between adjacent black and white squares and vice versa, which is clearly impossible given that the number of black and white squares is n^2 and $n^2 - 2$, respectively. As shown above (see Section 4), for refuting CB formulas it is enough to use only a half of the complete CB formula. This half can be seen as forcing a mapping from the set of black squares into the set of white squares, which is similar to the

[10] Larger values of m were not considered for triplewise-encoded $2\mathrm{PHP}_m^{2m+1}$ formulas because the size of the formulas grows as m^3.

[11] Note that discarding the $\mathcal{P}$ clauses can be done for 2PHP because the short proofs for $2\mathrm{PHP}_m^{2m+1}$ provided in this paper do not use clauses $(\overline{p_i} \vee \overline{n_i}, \top)$.



(a) Complete CB formulas



(b) Partial CB formulas

Figure 7: Performance of SAT and MaxSAT solvers on CB formulas.

pigeonhole principle. Based on this observation, we additionally created a set of benchmarks encoding this half of the formula. In contrast to the *complete CB* instances, this additional set of CB benchmarks is referred to as *partial CB* formulas. The partial CB benchmark set is constructed with the same values of parameter n, and thus its size is also 48. Finally, both complete and partial formulas were considered with and without the $\mathcal{P}$ clauses.

Figure 7 show cactus plots detailing the performance of the considered solvers. Similar to the PHP and 2PHP case, IHS-based MaxSAT solvers *maxhs* and *lmhs* demonstrate the best performance. This is the case for both complete and partial CB benchmarks. Observe that their configurations dealing with the CB formulas without the $\mathcal{P}$ clauses are not shown in the plots because the per-

Table 5: Performance of considered solvers on formulas encoding Parity Principle.

n	crypto	glucose	lgl	lgl-nocard	minisat	lp-cnf	mscg	wbo	lmhs	lmhs-neqs	maxhs	lp-wcnf
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.01	0.01	0.02	1.65
2	0.0	0.0	0.0	0.0	0.0	1.08	0.0	0.0	0.03	0.01	0.02	23.25
3	0.0	0.0	0.0	0.0	0.0	19.42	0.0	0.0	0.02	0.08	0.03	26.63
4	0.0	0.00	0.0	0.0	0.0	21.87	0.0	0.01	0.03	0.2	0.03	23.88
5	0.04	0.05	0.04	0.04	0.03	20.59	0.13	0.13	0.04	1.4	0.07	2.09
6	1.09	1.08	0.67	0.62	0.69	24.04	4.73	2.16	0.05	1.45	0.14	26.0
7	23.35	43.02	18.94	19.6	38.74	22.46	139.82	24.84	0.13	10.7	0.07	24.9
8	733.35	1506.72	876.98	980.47	1703.33	22.30	—	1070.46	0.05	6.46	0.28	26.36
9	—	—	—	—	—	22.08	—	—	0.12	6.43	0.57	39.92
10	—	—	—	—	—	21.06	—	—	0.13	45.03	0.54	39.71
11	—	—	—	—	—	23.0	—	—	0.18	36.15	0.37	46.88
12	—	—	—	—	—	21.68	—	—	0.84	37.04	0.86	44.11
13	—	—	—	—	—	23.91	—	—	0.83	92.47	0.91	31.65
14	—	—	—	—	—	18.8	—	—	0.23	24.16	1.02	86.26
15	—	—	—	—	—	15.64	—	—	1.04	287.28	1.15	36.31
16	—	—	—	—	—	16.45	—	—	—	365.06	1.17	23.23
17	—	—	—	—	—	12.84	—	—	0.42	531.51	0.31	115.03
18	—	—	—	—	—	16.49	—	—	0.52	234.13	1.32	23.27
19	—	—	—	—	—	18.26	—	—	1.87	457.45	1.68	25.34
20	—	—	—	—	—	17.86	—	—	1.76	112.82	1.69	22.93

formance of *maxhs* and *lmhs* is not affected by them. As expected, SAT solvers have hard time refuting the CB formulas being able to deal with only relatively small values of n, e.g. when $n \leq 8$. Surprisingly, core-guided MaxSAT solvers *mscg* and *wbo* dot not succeed either if the $\mathcal{P}$ clauses are present. This is the case for both complete and partial CB formulas, which is in contrast with the results for PHP shown earlier (*mscg* was able to solve all the PHP instances, even with the $\mathcal{P}$ clauses enabled). Moreover, *eva*, which is based on MaxSAT resolution, significantly outperforms its core-guided rivals. In fact, *eva* is able to refute the complete CB formulas with or without the $\mathcal{P}$ clauses showing the same performance (that is why only one configuration of *eva* is shown in Figure 7a). Although we do not have a clear understanding of this phenomenon at this point, a hypothesis is that *eva* has some pattern matching heuristics working effectively in this concrete case of the complete CB formulas. Observe that the core-guided MaxSAT solver *mscg* is able to find a way to refute the partial CB formulas efficiently, when the $\mathcal{P}$ clauses are enabled or disabled. In contrast to these results, inefficiency of both core-guided MaxSAT solvers on the complete CB formulas can be attributed to the additional clauses of the formulas that bring a number of unsatisfiable cores and, thus, ways to refute the formula taken by the solvers.

7.5. Parity Principle

Similar to the previous sections, this section targets assessing the practical efficiency of dual-rail MaxSAT for the Parity Principle as expressed in Section 4.3.3. In this set of experiments, consider n that varies between 1 and 20. The size of the graph of the Parity Principle encoded is m, corresponding to $m = 2n + 1$. Recall that the Parity Principle expresses a kind of mod 2 counting, which states that no graph on m (odd) nodes consists of a complete perfect matching [1, 9, 11].

Table 6: Performance of considered solvers on dual-rail MaxSAT formulas encoding Parity Principle w/o $\mathcal{P}$ clauses.

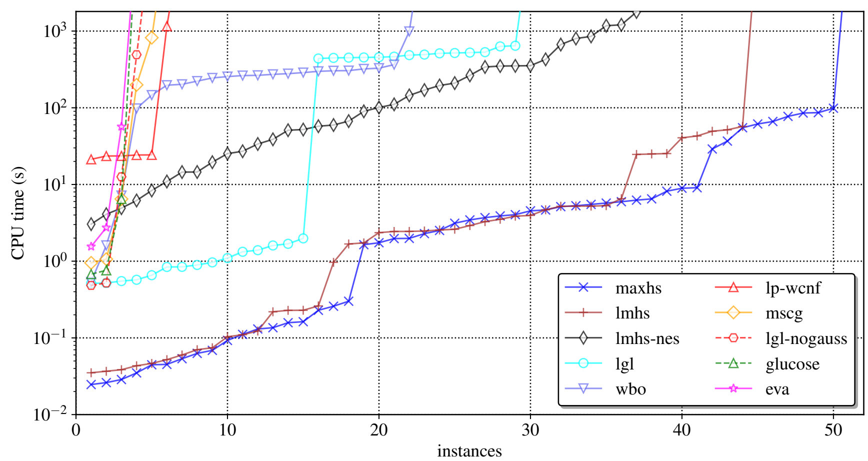
n	mscg	wbo	lmhs	lmhs-neqs	maxhs	lp-wcnf
1	0.0	0.0	0.01	0.01	0.0	0.01
2	0.0	0.0	0.01	0.01	0.0	3.77
3	0.0	0.0	0.01	0.02	0.0	3.32
4	0.01	0.01	0.01	0.06	0.01	3.47
5	0.1	0.08	0.02	0.07	0.01	3.55
6	1.41	0.82	0.02	0.20	0.01	3.54
7	23.38	5.66	0.02	0.26	0.01	3.74
8	427.46	113.15	0.04	0.39	0.02	32.98
9	—	—	0.04	0.82	0.02	35.41
10	—	—	0.06	1.09	0.03	32.58
11	—	—	0.07	1.80	0.04	33.08
12	—	—	0.09	3.07	0.05	3.76
13	—	—	0.11	4.82	0.06	3.71
14	—	—	0.14	6.77	0.07	3.76
15	—	—	0.17	7.73	0.09	36.09
16	—	—	0.8	14.99	0.57	6.0
17	—	—	0.76	26.82	0.19	14.32
18	—	—	1.04	68.49	0.76	12.56
19	—	—	1.50	—	0.84	44.54
20	—	—	5.6	100.9	0.21	24.92

Table 5 shows the results of running the considered solvers on the encoded Parity Principle formulas, while Table 6 shows the results of running the dual-rail MaxSAT encoding of the encoded Parity Principle formulas, but disregarding the $\mathcal{P}$ clauses. As before, IHS-based MaxSAT solvers maxhs and lmhs demonstrate the best performance. In this case, there is one outlier where lmhs was unable to compute the solution, which does happen when disregarding the $\mathcal{P}$ clauses. Interestingly, CPLEX works well on this set on benchmarks, both with lp-cnf and lp-wncf, which solve all the 20 parity formulas, on average one order of magnitude slower than maxhs. On the other hand, the core-guided approaches using dual-rail MaxSAT perform equally to the SAT based approaches, both of approaches not solving more than 8 instances.

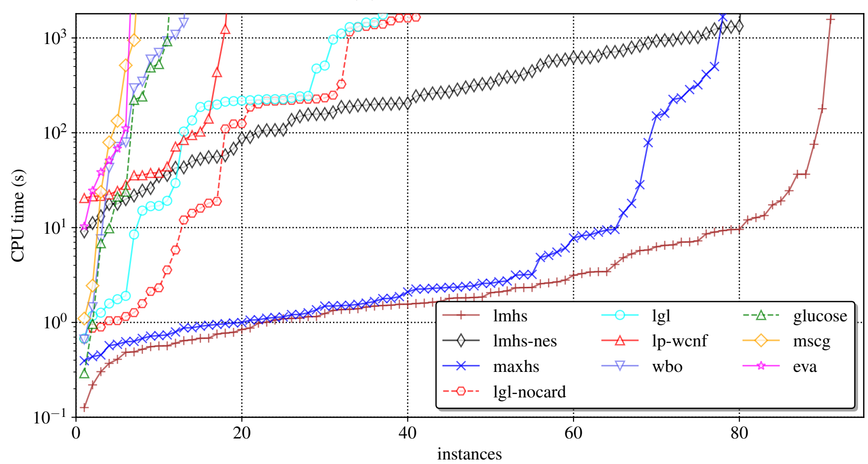
Finally, note that for this set of benchmarks, ignoring the $\mathcal{P}$ clauses does not produce gains in the performances of the dual-rail MaxSAT approaches.

7.6. Urquhart benchmarks and combined instances

The Urquhart (URQ) instances (of the Tseitin tautologies) are based on linear equations mod 2 which are known to be hard for resolution [72], 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 84 instances with n ranging from 3 to 30. The best performance is demonstrated by both *maxhs* and *lmhs*. 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 8a) performs reasonably well being able to solve 29 instances. However, as the result for *lgl-nogauss* suggests,



(a) URQ instances



(b) COMB instances

Figure 8: Performance of the considered solvers on URQ and combined formulas.

GA is crucial for *lgl* to efficiently decide URQ. Note that *lp-cnf* is not shown in Figure 8a due to its inability to solve any instance.

The COMB benchmark set inherits the complexity of both PHP and URQ instances and contains formulas $\mathrm{PHP}_m^{m+1} \vee \mathrm{URQ}_{n,i}$ with the PHP part being pairwise-encoded, where $m \in \{7, 9, 11, 13\}$, $n \in \{3, \ldots, 10\}$, and $i \in \{1, 2, 3\}$, i.e. $|\mathrm{COMB}| = 96$. By construction, in order to refute the COMB formulas, one has to refute both PHP and URQ subformulas. This makes the COMB formulas at least as hard as the subformulas involved. As one can observe in Figure 8b, even the small values of m and n used result in instances that are hard for most of the competitors. All IHS-based MaxSAT solvers (*maxhs*, *lmhs*, and *lmhs-nes*)

Table 7: Number of solved instances per solver.

		glucose	lgl	lgl-no[12]	maxhs	lmhs	lmhs-nes	mscg	wbo	eva	lp-cnf	lp-wcnf
PHP-pw	(46)	7	29	7	**46**	**46**	29	**46**	10	**46**	**46**	**46**
PHP-sc	(46)	13	11	11	**46**	**46**	45	**46**	15	40	**46**	**46**
2PHP	(67)	10	**27**	9	5	11	9	6	9	6	17	18
CB	(96)	12	23	12	**96**	**96**	89	47	10	83	—	—
Parity	(20)	8	8	8	**20**	19	**20**	7	8	—	**20**	**20**
URQ	(84)	3	29	4	**50**	44	37	5	22	3	0	6
COMB	(96)	11	37	41	78	**91**	80	7	13	6	0	18
Total	(455)	64	164	92	341	**353**	309	164	87	184 [13]	*129*[14]	*154*

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). 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.

7.7. Summary of Experimental Results

This section presents an overall summary of the experiments in this work. Table 7 shows that, given all the previously considered benchmarks sets, the dual-rail 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 perform well on the pigeonhole formulas (except *wbo*, and this has to be investigated further), which supports the theoretical claims of the paper. However, using them does not help solving the other considered benchmarks families. As expected, SAT solvers cannot deal with most of the considered formulas as long as they do not utilize additional powerful reasoning techniques (e.g. GA or CBR). However, and as the COMB instances demonstrate, it is easy to construct formulas that are hard for the state-of-the-art SAT solvers, even if 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*.

In the experiments, we have also considered the possibility of discarding $\mathcal{P}$ clauses. Discarding $\mathcal{P}$ clauses in general does not lead to correct results, but depending on the benchmark family, and as demonstrated in each of the sections associated to the specific benchmark families, the inclusion of $\mathcal{P}$ clauses can be harmful for the performance of the MaxSAT solvers, (leading in some cases to orders of magnitude improvements).

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

[13] As Parity results are unavailable for *eva*, the total number of instances solved by *eva* should be seen an under-approximation.

[14] As CB results are unavailable for the two configurations of CPLEX used, the total number of instances solved by *lp-cnf* and *lp-wcnf* should be seen an under-approximation.

To conclude, the experimental results confirm the practical efficiency of the dual-rail based MaxSAT solving compared to the CDCL SAT approach, which is known to be equivalent to resolution. A number of families of formulas encoding hard combinatorial principles were considered for showing this. Moreover, in some situations dual-rail based MaxSAT solving was shown to outperform solutions based on mixed integer programming and cutting planes. We deem these results encouraging in light of the success of MaxSAT solvers in the recent years.

8. Conclusions & Research Directions

This paper contributes to the ongoing quest for a practically effective SAT algorithm that exploits a proof system stronger than resolution. The paper's main contribution is to aggregate and extend earlier results on the dual-rail MaxSAT proof system [38, 17, 55].

Although the paper offers a characterization of upper bounds and some initial simulation results, additional work remains for establishing the comparative strength of the different approaches using the dual-rail encoding. This is the subject of future work. Another line of work is to assess whether practical implementations of the dual-rail MaxSAT proof system are effective as an alternative to CDCL SAT solvers.

Acknowledgments.
This work was supported by FCT grants ABSOLV (PTDC/CCI-COM/28986/2017), FaultLocker (PTDC/CCI-COM/29300/2017), SAFETY (SFRH/BPD/120315/2016), SAMPLE (CEECIND/04549/2017), and INFOCOS (PTDC/CCI-COM/32378/2017); grant TIN2016-76573-C2-2-P (TASSAT 3); grant PID2019-109137GB-C21 (PROOFS); and Simons Foundation grant 578919. The work was also supported by the AI Interdisciplinary Institute ANITI, funded by the French program "Investing for the Future - PIA3" under Grant agreement n^o ANR-19-PI3A-0004.

We thank the referees for many substantial and useful comments improving the paper.

References

[1] Ajtai, M., 1990. Parity and the pigeonhole principle, in: Feasible Mathematics, Birkhäuser. pp. 1–24.

[2] Alekhnovich, M., 2004a. Mutilated chessboard problem is exponentially hard for resolution. Theoretical Computer Science 310, 513–525.

[3] Alekhnovich, M., 2004b. Mutilated chessboard problem is exponentially hard for resolution. Theor. Comput. Sci. 310, 513–525. doi:10.1016/S0304-3975(03)00395-5.

[4] Ansótegui, C., Bonet, M.L., Levy, J., 2013. SAT-based MaxSAT algorithms. Artif. Intell. 196, 77–105. doi:10.1016/j.artint.2013.01.002.

[5] Ansótegui, C., Didier, F., Gabàs, J., 2015. Exploiting the structure of unsatisfiable cores in maxsat, in: IJCAI, pp. 283–289. URL: http://ijcai.org/Abstract/15/046.

[6] Atserias, A., Fichte, J.K., Thurley, M., 2011. Clause-learning algorithms with many restarts and bounded-width resolution. J. Artif. Intell. Res. 40, 353–373. URL: https://doi.org/10.1613/jair.3152, doi:10.1613/jair.3152.

[7] Audemard, G., Katsirelos, G., Simon, L., 2010. A restriction of extended resolution for clause learning SAT solvers, in: Fox, M., Poole, D. (Eds.), Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010, AAAI Press. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/view/1811.

[8] Audemard, G., Lagniez, J., Simon, L., 2013. Improving Glucose for incremental SAT solving with assumptions: Application to MUS extraction, in: SAT, pp. 309–317. doi:10.1007/978-3-642-39071-5_23.

[9] Beame, P., Impagliazzo, R., Krajíček, J., Pitassi, T., Pudlák, P., 1996. Lower bounds on Hilbert's Nullstellensatz and propositional proofs. Proceedings of the London Mathematical Society 73, 1–26.

[10] Beame, P., Kautz, H.A., Sabharwal, A., 2004. Towards understanding and harnessing the potential of clause learning. J. Artif. Intell. Res. 22, 319–351. URL: https://doi.org/10.1613/jair.1410, doi:10.1613/jair.1410.

[11] Beame, P., Pitassi, T., 1996. An exponential separation between the parity principle and the pigeonhole principle. Annals of Pure and Applied Logic 80, 195–228.

[12] Beame, P., Sabharwal, A., 2014. Non-restarting SAT solvers with simple preprocessing can efficiently simulate resolution, in: AAAI, AAAI Press. pp. 2608–2615.

[13] Biere, A., 2013a. Lingeling, plingeling and treengeling entering the SAT competition 2013, in: Balint, A., Belov, A., Heule, M., Järvisalo, M. (Eds.), Proceedings of SAT Competition 2013. University of Helsinki. volume B-2013-1 of *Department of Computer Science Series of Publications B*, pp. 51–52.

[14] Biere, A., 2013b. Two pigeons per hole problem, in: Proceedings of SAT Competition 2013, pp. 103–103.

[15] Biere, A., 2014. Lingeling essentials, A tutorial on design and implementation aspects of the SAT solver lingeling, in: Pragmatics of SAT workshop, p. 88. URL: http://www.easychair.org/publications/?page=1039022100.

[16] Biere, A., Heule, M., van Maaren, H., Walsh, T. (Eds.), 2009. Handbook of Satisfiability. volume 185 of *Frontiers in Artificial Intelligence and Applications*, IOS Press.

[17] Bonet, M.L., Buss, S., Ignatiev, A., Marques-Silva, J., Morgado, A., 2018. MaxSAT resolution with the dual rail encoding, in: AAAI. URL: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16782.

[18] Bonet, M.L., Levy, J., Manyà, F., 2007. Resolution for Max-SAT. Artif. Intell. 171, 606–618. doi:10.1016/j.artint.2007.03.001.

[19] Bryant, R.E., Beatty, D.L., Brace, K.S., Cho, K., Sheffler, T.J., 1987. COSMOS: A compiled simulator for MOS circuits, in: DAC, pp. 9–16.

[20] Buss, S.R., 2012. Towards NP-P via proof complexity and proof search. Annals of Pure and Applied Logic 163, 1163–1182.

[21] Buss, S.R., Clote, P., 1996. Cutting planes, connectivity and threshold logic. Archive for Mathematical Logic 35, 33–62.

[22] Chatalic, P., Simon, L., 2001. Multiresolution for SAT checking. International Journal on Artificial Intelligence Tools 10, 451–481. URL: http://dx.doi.org/10.1142/S0218213001000611, doi:10.1142/S0218213001000611.

[23] Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H., 2003. Counterexample-guided abstraction refinement for symbolic model checking. J. ACM 50, 752–794. URL: http://doi.acm.org/10.1145/876638.876643, doi:10.1145/876638.876643.

[24] Cook, S.A., 1971. The complexity of theorem-proving procedures, in: STOC, ACM. pp. 151–158.

[25] Cook, S.A., Reckhow, R.A., 1979a. The relative efficiency of propositional proof systems. Journal of Symbolic Logic 44, 36–50.

[26] Cook, S.A., Reckhow, R.A., 1979b. The relative efficiency of propositional proof systems. J. Symb. Log. 44, 36–50. doi:10.2307/2273702.

[27] Davies, J., Bacchus, F., 2011. Solving MAXSAT by solving a sequence of simpler SAT instances, in: CP, pp. 225–239.

[28] Davies, J., Bacchus, F., 2013a. Exploiting the power of mip solvers in maxsat, in: SAT, pp. 166–181. URL: http://dx.doi.org/10.1007/978-3-642-39071-5_13, doi:10.1007/978-3-642-39071-5_13.

[29] Davies, J., Bacchus, F., 2013b. Postponing optimization to speed up MAXSAT solving, in: CP, pp. 247–262. URL: http://dx.doi.org/10.1007/978-3-642-40627-0_21, doi:10.1007/978-3-642-40627-0_21.

[30] Eén, N., Sörensson, N., 2003. An extensible SAT-solver, in: SAT, pp. 502–518. doi:10.1007/978-3-540-24605-3_37.

[31] Elffers, J., Nordström, J., 2018. Divide and conquer: Towards faster pseudo-boolean solving, in: Lang, J. (Ed.), Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden., ijcai.org. pp. 1291–1299. URL: https://doi.org/10.24963/ijcai.2018/180, doi:10.24963/ijcai.2018/180.

[32] Fu, Z., Malik, S., 2006. On solving the partial MAX-SAT problem, in: SAT, pp. 252–265.

[33] Heras, F., Morgado, A., Marques-Silva, J., 2011. Core-guided binary search algorithms for maximum satisfiability, in: Burgard, W., Roth, D. (Eds.), Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011, AAAI Press. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI11/paper/view/3713.

[34] Hertel, P., Bacchus, F., Pitassi, T., Gelder, A.V., 2008. Clause learning can effectively p-simulate general propositional resolution, in: Fox, D., Gomes, C.P. (Eds.), Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008, AAAI Press. pp. 283–290. URL: http://www.aaai.org/Library/AAAI/2008/aaai08-045.php.

[35] Heule, M.J.H., Kiesl, B., Biere, A., 2019. Encoding redundancy for satisfaction-driven clause learning, in: Vojnar, T., Zhang, L. (Eds.), Tools and Algorithms for the Construction and Analysis of Systems - 25th International Conference, TACAS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part I, Springer. pp. 41–58. URL: https://doi.org/10.1007/978-3-030-17462-0_3, doi:10.1007/978-3-030-17462-0_3.

[36] Huang, J., 2010. Extended clause learning. Artif. Intell. 174, 1277–1284. URL: https://doi.org/10.1016/j.artint.2010.07.008, doi:10.1016/j.artint.2010.07.008.

[37] IBM ILOG, 2016. IBM ILOG: CPLEX optimizer 12.7.0. http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer.

[38] Ignatiev, A., Morgado, A., Marques-Silva, J., 2017. On tackling the limits of resolution in SAT solving, in: SAT, pp. 164–183. doi:10.1007/978-3-319-66263-3_11.

[39] Impagliazzo, R., Pitassi, T., Urquhart, A., 1994. Upper and lower bounds for tree-like cutting planes proofs, in: Proceedings of the Ninth Annual

Symposium on Logic in Computer Science (LICS '94), Paris, France, July 4-7, 1994, IEEE Computer Society. pp. 220–228. doi:10.1109/LICS.1994.316069.

[40] Jabbour, S., Marques-Silva, J., Sais, L., Salhi, Y., 2014. Enumerating prime implicants of propositional formulae in conjunctive normal form, in: JELIA, pp. 152–165.

[41] Jaumard, B., Simeone, B., 1987. On the complexity of the maximum satisfiability problem for horn formulas. Inf. Process. Lett. 26, 1–4. URL: https://doi.org/10.1016/0020-0190(87)90028-7, doi:10.1016/0020-0190(87)90028-7.

[42] Krishnamurthy, B., 1985. Short proofs for tricky formulas. Acta Inf. 22, 253–275. URL: https://doi.org/10.1007/BF00265682, doi:10.1007/BF00265682.

[43] Larrosa, J., Heras, F., 2005. Resolution in Max-SAT and its relation to local consistency in weighted CSPs, in: IJCAI, pp. 193–198.

[44] Manquinho, V.M., Flores, P.F., Marques-Silva, J., Oliveira, A.L., 1997. Prime implicant computation using satisfiability algorithms, in: ICTAI, pp. 232–239. doi:10.1109/TAI.1997.632261.

[45] Marques-Silva, J., Ignatiev, A., Morgado, A., 2017. Horn maximum satisfiability: Reductions, algorithms and applications, in: Oliveira, E.C., Gama, J., Vale, Z.A., Cardoso, H.L. (Eds.), Progress in Artificial Intelligence - 18th EPIA Conference on Artificial Intelligence, EPIA 2017, Porto, Portugal, September 5-8, 2017, Proceedings, Springer. pp. 681–694. URL: https://doi.org/10.1007/978-3-319-65340-2_56, doi:10.1007/978-3-319-65340-2_56.

[46] Marques-Silva, J., Malik, S., 2018. Propositional SAT solving, in: Handbook of Model Checking. Springer, pp. 247–275.

[47] Marques-Silva, J., Planes, J., 2007. On using unsatisfiability for solving maximum satisfiability. CoRR abs/0712.1097. URL: http://arxiv.org/abs/0712.1097.

[48] Marques-Silva, J., Sakallah, K.A., 1996. GRASP - a new search algorithm for satisfiability, in: ICCAD, pp. 220–227.

[49] Marques-Silva, J., Sakallah, K.A., 1999. GRASP: A search algorithm for propositional satisfiability. IEEE Trans. Computers 48, 506–521.

[50] Martins, R., Joshi, S., Manquinho, V.M., Lynce, I., 2014a. Incremental cardinality constraints for MaxSAT, in: CP, pp. 531–548.

[51] Martins, R., Manquinho, V.M., Lynce, I., 2014b. Open-WBO: A modular MaxSAT solver,, in: SAT, pp. 438–445. URL: http://dx.doi.org/10.1007/978-3-319-09284-3_33, doi:10.1007/978-3-319-09284-3_33.

[52] McCarthy, J., 1964. A tough nut for proof procedures. Standord Artificial Intelligence Project, Memo No. 16.

[53] Morgado, A., Dodaro, C., Marques-Silva, J., 2014. Core-guided MaxSAT with soft cardinality constraints, in: CP, pp. 564–573. URL: http://dx.doi.org/10.1007/978-3-319-10428-7_41, doi:10.1007/978-3-319-10428-7_41.

[54] Morgado, A., Heras, F., Liffiton, M.H., Planes, J., Marques-Silva, J., 2013. Iterative and core-guided MaxSAT solving: A survey and assessment. Constraints 18, 478–534. doi:10.1007/s10601-013-9146-2.

[55] Morgado, A., Ignatiev, A., Bonet, M.L., Marques-Silva, J., Buss, S., 2019. Drmaxsat with maxhs: First contact, in: Janota, M., Lynce, I. (Eds.), Theory and Applications of Satisfiability Testing - SAT 2019 - 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9-12, 2019, Proceedings, Springer. pp. 239–249. URL: https://doi.org/10.1007/978-3-030-24258-9_17, doi:10.1007/978-3-030-24258-9_17.

[56] Morgado, A., Ignatiev, A., Marques-Silva, J., 2015. MSCG: Robust core-guided MaxSAT solving. JSAT 9, 129–134.

[57] Narodytska, N., Bacchus, F., 2014. Maximum satisfiability using core-guided MaxSAT resolution, in: AAAI, pp. 2717–2723. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8513.

[58] Pan, G., Vardi, M.Y., 2004. Search vs. symbolic techniques in satisfiability solving, in: SAT, pp. 235–250.

[59] Pan, G., Vardi, M.Y., 2005. Symbolic techniques in satisfiability solving. J. Autom. Reasoning 35, 25–50. URL: https://doi.org/10.1007/s10817-005-9009-7, doi:10.1007/s10817-005-9009-7.

[60] Pipatsrisawat, K., Darwiche, A., 2011. On the power of clause-learning SAT solvers as resolution engines. Artif. Intell. 175, 512–525. URL: https://doi.org/10.1016/j.artint.2010.10.002, doi:10.1016/j.artint.2010.10.002.

[61] Previti, A., Ignatiev, A., Morgado, A., Marques-Silva, J., 2015. Prime compilation of non-clausal formulae, in: IJCAI, pp. 1980–1988.

[62] Pudlák, P., 1997. Lower bounds for resolution and cutting planes proofs and monotone computations. Journal of Symbolic Logic 62, 981–998.

[63] Pudlák, P., 1999. On the complexity of propositional calculus, Sets and proofs, in: Logic Colloquium '97, Cambridge University Press. pp. 197–218.

[64] Reiter, R., 1987. A theory of diagnosis from first principles. Artif. Intell. 32, 57–95.

[65] Riis, S., 1993. Independence in Bounded Arithmetic. Ph.D. thesis. Oxford University.

[66] Roorda, J., Claessen, K., 2005. A new SAT-based algorithm for symbolic trajectory evaluation, in: CHARME, pp. 238–253.

[67] Saikko, P., Berg, J., Järvisalo, M., 2016. LMHS: A SAT-IP hybrid MaxSAT solver, in: SAT, pp. 539–546.

[68] Sinz, C., 2005. Towards an optimal CNF encoding of Boolean cardinality constraints, in: CP, pp. 827–831.

[69] Soos, M., 2010. Enhanced gaussian elimination in dpll-based SAT solvers, in: POS@SAT, pp. 2–14.

[70] Soos, M., Nohl, K., Castelluccia, C., 2009. Extending SAT solvers to cryptographic problems, in: SAT, pp. 244–257.

[71] Tseitin, G.S., 1968. On the complexity of derivation in propositional calculus. Studies in constructive mathematics and mathematical logic 2, 10–13.

[72] Urquhart, A., 1987. Hard examples for resolution. J. ACM 34, 209–219. URL: http://doi.acm.org/10.1145/7531.8928, doi:10.1145/7531.8928.

[73] Vinyals, M., Elffers, J., Giráldez-Cru, J., Gocht, S., Nordström, J., 2018. In between resolution and cutting planes: A study of proof systems for pseudo-boolean SAT solving, in: Beyersdorff, O., Wintersteiger, C.M. (Eds.), Theory and Applications of Satisfiability Testing - SAT 2018 - 21st International Conference, SAT 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9-12, 2018, Proceedings, Springer. pp. 292–310. URL: https://doi.org/10.1007/978-3-319-94144-8_18, doi:10.1007/978-3-319-94144-8_18.