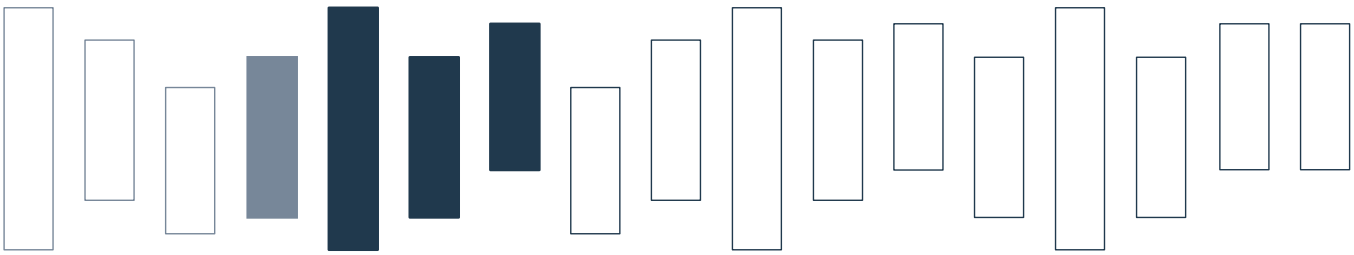
Technical Report AC-TR-20-008

September 2020

Finding the Hardest Formulas for Resolution

Tomáš Peitl and Stefan Szeider



This is the authors' copy of a paper that will appear in the proceedings of CP'20, the 26th International Conference on Principles and Practice of Constraint Programming, and received the Best Paper Award for the CP'20 Technical Track.
DOI 10.1007/978-3-030-58475-7_30
www.ac.tuwien.ac.at/tr

Finding the Hardest Formulas for Resolution*

Tomáš Peitl[1] and Stefan Szeider[2]

[1] Friedrich Schiller University Jena, Germany, tomas.peitl@uni-jena.de
[2] TU Wien, Austria, sz@tuwien.ac.at

Abstract. A CNF formula is harder than another CNF formula with the same number of clauses if it requires a longer resolution proof. The *resolution hardness numbers* give for $m = 1, 2, \ldots$ the length of a shortest proof of a hardest formula on m clauses. We compute the first ten resolution hardness numbers, along with the corresponding hardest formulas. We achieve this by a candidate filtering and symmetry breaking search scheme for limiting the number of potential candidates for formulas and an efficient SAT encoding for computing a shortest resolution proof of a given candidate formula.

Keywords: Resolution proofs · minimal unsatisfiability · SAT encodings · graph symmetries · computational experiments

1 Introduction

Resolution is a fundamental proof system that can be used to certify the unsatisfiability of a propositional formula in conjunctive normal form (CNF). What makes resolution particularly interesting is that the length of a shortest resolution proof of a given CNF formula (called the *resolution complexity* of the formula) provides an unconditional lower bound on the running time of modern SAT solvers [17]. Since we know that there are classes of unsatisfiable CNF formulas (such as the formulas based on the Pigeon Hole Principle) with exponential resolution complexity [6], we have an exponential lower bound on the runtime. It is a natural question to ask: *which formulas are the hardest for resolution?* i.e., which formulas have the highest resolution complexity? This is a quite intriguing and hard question, which has been approached mainly in an asymptotic way by propositional proof complexity [22].

We address this question by following a recent trend in tackling combinatorial problems using SAT and CSP methods [2, 7, 8]. For small values of n and m, we compute all the formulas (modulo isomorphisms) with n variables and m clauses that are the hardest formulas for resolution. With these results, we can compute the first few *resolution hardness numbers* $(h_m)_{m\geq 1}$, where h_m gives the highest resolution complexity of a CNF formula with m clauses.

We obtain our results by the combination of two techniques:

* The authors acknowledge the support by the FWF (projects P32441 and J-4361) and by the WWTF (project ICT19-065).

1. A *candidate filtering and symmetry breaking search scheme* for limiting the number of potential candidate formulas with m variables whose resolution complexity is h_m.
2. An *efficient SAT encoding* for computing the resolution complexity of a given candidate formula.

In our search scheme, we reduce the candidate formulas to a certain class of minimally unsatisfiable (MU) formulas that obey additional degree constraints. We model these formulas by graphs of a particular kind. We generate these graphs modulo symmetries by a special adaptation of the Nauty graph symmetry package.

This still leaves us with a large number of formulas whose resolution complexity we must determine algorithmically. For this task, we devised an efficient SAT encoding that produces for a given candidate formula F and an integer s, a CNF formula $\texttt{short}_s(F)$, which is satisfiable if and only if F admits a resolution proof of length $\leq s$. We determine the resolution complexity of F by feeding $\texttt{short}_s(F)$ to a SAT solver with various choices of s. While a SAT encoding for this problem has been proposed before [14], and we do take some inspiration from it, we make crucial adaptations tailored towards minimally unsatisfiable formulas. Furthermore, we introduce a symmetry-breaking scheme that fully breaks all symmetries resulting from permutations of the sequence of clauses.

In addition to the values of the resolution hardness numbers, we can draw a more detailed map of the hardest formulas with a particular number n of variables and a particular number m of clauses.

Our theoretical results reveal the significance of *regular saturated minimally unsatisfiable (RSMU)* formulas, which are unsatisfiable formulas that (i) become satisfiable by adding any further literal to any clause, and (ii) where each literal appears in at least two clauses. As a by-product of our computations, we obtain a catalog of RSMU formulas with a small number of variables and clauses, which may be of independent interest in the research on minimal unsatisfiability. For instance, the computed formulas' structure can possibly be used to come up with infinite sequences of hard formulas, which can lead to tighter general bounds.

An alternative but not very interesting object of study would be v_n, the highest resolution complexity of formulas with n *variables*. It is not hard to see that every unsatisfiable formula on n variables has a resolution refutation of length $\leq 2^{n+1} - 1$ and that indeed $v_n = 2^{n+1} - 1$, witnessed by the formula which contains all possible clauses of width n.

2 Preliminaries

Formulas. We consider propositional formulas in conjunctive normal form (CNF) represented as sets of clauses. We assume an infinite set *var* of (propositional) *variables*. A *literal* ℓ is a variable x or a negated variable $\neg x$; we write $\mathit{lit} := \{ x, \neg x \mid x \in \mathit{var} \}$. For a literal ℓ we put $\overline{\ell} := \neg x$ if $\ell = x$, and $\overline{\ell} := x$ if $\ell = \neg x$. For a set of C literals we put $\overline{C} := \{ \overline{\ell} \mid \ell \in C \}$. C is *tautological* if $C \cap \overline{C} \neq \emptyset$. A finite non-tautological set of literals is a *clause*; a finite set of clauses is a (CNF)

formula. The empty clause is denoted by $\square$. We write $\mathrm{CNF}(n,m)$ for the class of all CNF formulas on n variables and m clauses, and $\mathrm{CNF}(m) = \bigcup_{n=0}^{\infty} \mathrm{CNF}(n,m)$. For a clause C, we put $\mathit{var}(C) = \{\, \mathit{var}(\ell) \mid \ell \in C \,\}$, and for a formula F, $\mathit{var}(F) = \bigcup_{C \in F} \mathit{var}(C)$. Similarly, we put $\mathit{lit}(F) := \mathit{var}(F) \cup \overline{\mathit{var}(F)}$. A formula F is *satisfiable* if there is a mapping $\tau : \mathit{var}(F) \to \{0,1\}$ such that every clause of F contains either a literal x with $\tau(x) = 1$ or a literal $\neg x$ with $\tau(x) = 0$, and *unsatisfiable* otherwise. A formula is *minimally unsatisfiable (MU)* if it is unsatisfiable, but every proper subset is satisfiable.

Resolution Proofs. If $C_1 \cap \overline{C_2} = \{\ell\}$ for clauses C_1, C_2 and a literal ℓ, then the *resolution rule* allows the derivation of the clause $D = (C_1 \cup C_2) \setminus \{\ell, \overline{\ell}\}$; D is the *resolvent* of the *premises* C_1 and C_2, and we say that D is obtained by *resolving on* ℓ. Let F be a formula and C a clause. A sequence $P = L_1, \ldots, L_s$ of clauses (proof *lines*) is a *resolution derivation of* L_s *from* F if for each $i \in \{1, \ldots, s\}$ at least one of the following holds.

1. $L_i \in F$ ("L_i is an axiom");
2. L_i is the resolvent of L_j and $L_{j'}$ for some $1 \leq j < j' < i$ ("L_i is obtained by resolution").

We write $|P| := s$ and call s the *length* of P. If L_s is the empty clause, then P is a *resolution refutation* or *resolution proof* of F. A line L_i in a resolution derivation may have different possible "histories;" i.e., L_i may be the resolvent of more than one pair of clauses preceding L_i, or L_i may be both an axiom and obtained from preceding clauses by resolution, etc. In the sequel, however, we assume that an arbitrary but fixed history is associated with each considered resolution derivation.

It is well known that resolution is a complete proof system for unsatisfiable formulas; i.e., a formula F is unsatisfiable if and only if there exists a resolution refutation of it. The *resolution complexity* or *resolution hardness* $h(F)$ of an unsatisfiable formula F is the length of a shortest resolution refutation of F (for satisfiable formulas we put $h(F) := -\infty$). For a nonempty set $\mathcal{C}$ of formulas, we put $h(\mathcal{C}) = \max_{F \in \mathcal{C}} h(F)$.

Isomorphisms of Formulas. Two formulas F and F' are *isomorphic* if there exists a bijection $\varphi : \mathit{lit}(F) \to \mathit{lit}(F')$ such that for each literal $\ell \in \mathit{lit}(F)$ we have $\overline{\varphi(\ell)} = \varphi(\overline{\ell})$ and for each $C \subseteq \mathit{lit}(F)$ we have $C \in F$ if and only if $\varphi(C) \in F'$. For instance the formulas $F = \{\{x, \overline{y}\}, \{\overline{x}, y\}, \{\overline{y}\}\}$, and $F' = \{\{\overline{z}, w\}, \{z, w\}, \{\overline{w}\}\}$ are isomorphic.

Obviously, two isomorphic formulas have the same properties concerning satisfiability, minimal unsatisfiability, and resolution proof length. For a set $\mathcal{C}$ of formulas, we define $\mathrm{Iso}(\mathcal{C})$ to be an inclusion-maximal subset of $\mathcal{C}$ such that no two elements of $\mathrm{Iso}(\mathcal{C})$ are isomorphic. In other words, $\mathrm{Iso}(\mathcal{C})$ contains exactly one representative from each isomorphism class.

A *2-graph* is an undirected graph $G = (V, E)$ together with a partition of its vertex set into two subsets $V = V_1 \uplus V_2$. Two 2-graphs $G = (V_1 \uplus V_2, E)$ and $G' = (V_1' \uplus V_2', E')$ are *isomorphic* if there exists a bijection $\varphi : V_1 \uplus V_2 \to V_1' \uplus V_2'$

such that $v \in V_i$ if and only if $\varphi(v) \in V'_i$, $i = 1, 2$, and $\{u, v\} \in E$ if and only if $\{\varphi(u), \varphi(v)\} \in E'$.

The *clause-literal graph* of a formula F is the 2-graph $G(F) = (V_1 \uplus V_2, E)$ with $V_1 = \mathit{lit}(F)$, $V_2 = F$, and $E = \{\, \{x, \overline{x}\} \mid x \in \mathit{var}(F) \,\} \cup \{\, \{C, \ell\} \mid C \in F, \ell \in C \,\}$.

The following statement is easy to verify.

Proposition 1. *Two formulas are isomorphic if and only if their clause-literal graphs are isomorphic.*

3 Theoretical Framework

We define the *m-th resolution hardness number* as the highest resolution complexity among formulas with m clauses:

$$h_m = \max_{F \in \mathrm{CNF}(m)} h(F) = h(\mathrm{CNF}(m)).$$

In this section, we discuss various properties such that it suffices to consider only formulas with these properties for computing h_m.

Let $H(n, m) = \{\, F \in \mathrm{CNF}(n, m) \mid h(F) = h_m \,\}$ and $H(m) = \bigcup_{n=0}^{\infty} H(n, m)$; thus $h(\mathrm{CNF}(n, m)) = h(H(n, m))$, and $h(\mathrm{CNF}(m)) = h(H(m))$.

Lemma 1. *All formulas in $H(m)$ are minimally unsatisfiable.*

Proof. Suppose to the contrary, that there exists some $F \in H(m)$ which is not minimally unsatisfiable and choose any minimally unsatisfiable subset $F' \subsetneq F$ and let $d = m - |F'| \geq 1$. Pick a clause $C \in F'$ and take new variables $x_1, \ldots, x_d \notin \mathit{var}(F)$. We obtain a minimally unsatisfiable formula F'' from F' by replacing C by the clauses $C \cup \{x_1, \ldots, x_d\}$, $\{\overline{x_1}\}, \ldots, \{\overline{x_d}\}$. From a shortest resolution proof P'' of F'' we obtain a resolution proof P' of F'. By construction, $|P'| + d = |P''|$, hence $h(F) \leq h(F') + d \leq h(F'')$, which is a contradiction to $h(F'') \leq h_m = h(F)$, a contradiction. □

A formula is *saturated minimally unsatisfiable* if it is unsatisfiable and adding a literal to any of its clauses makes it satisfiable. Every saturated minimally unsatisfiable formula is minimally unsatisfiable, since adding a pure literal to a clause has the same effect as deleting the clause.

Lemma 2. *$H(m)$ contains a saturated minimally unsatisfiable formula.*

Proof. Let F be an arbitrary formula in $H(m)$. By Lemma 1, F is minimally unsatisfiable. Assume now that F is not saturated, and we can add to some clause C of F a literal ℓ, obtaining a minimally unsatisfiable formula F'. We claim that $h(F') \geq h(F) = h_m$. Take a shortest proof P of F'. Delete ℓ from the axiom $C \cup \{\ell\}$ in P and propagate this deletion through P to other clauses. This way, we obtain a sequence P' of clauses, which contains as a subsequence a resolution proof of F. Hence indeed $h(F') \geq h(F) = h_m$, and so $F' \in H(m)$. □

A literal ℓ is called *r-singular* in a formula F if there is exactly one clause in F that contains ℓ, and there are exactly r clauses in F that contain $\overline{\ell}$. A literal

is *singular* in F if it is r-singular for some $r \geq 0$ [21]. We also say a literal is $\geq r$-singular if it is r'-singular for some $r' \geq r$.

We denote by $\mathrm{MU}(n, m)$ the class of minimally unsatisfiable formulas with n variables and m clauses, and by $\mathrm{SMU}(n, m) \subseteq \mathrm{MU}(n, m)$ the subclass consisting of saturated formulas. $\mathrm{RSMU}(n, m)$ denotes the subclass of $\mathrm{SMU}(n, m)$ containing only formulas without singular variables. We call such formulas *regular*. We also use the shorthand $\mathrm{SSMU}(n, m) = \mathrm{SMU}(n, m) \setminus \mathrm{RSMU}(n, m)$.

Consider a formula F and a variable x of F. Let $\mathrm{DP}_x(F)$ denote the formula obtained from F after adding all possible resolvents that can be obtained from clauses in F by resolving on x and removing all clauses in which x occurs [21]. We say that $\mathrm{DP}_x(F)$ is obtained from F by *Davis-Putnam reduction* or short *DP-reduction* on x [3]. We will mainly use DP-reduction in the opposite direction, starting with a formula F and generating a formula F' such that $F = \mathrm{DP}_x(F')$. We then say that F' has been obtained from F by *DP-lifting*.

The following result by Kullmann and Zhao [12, Lemma 12] establishes an important link between DP-reduction on a singular variable and saturated minimal unsatisfiability.

Lemma 3 (Kullmann and Zhao [12]). *Let F be a formula and x an r-singular literal of F such that C_0 is the only clause of F containing x and $C_1, \ldots, C_r$ are the only clauses of F containing $\overline{x}$. Then $F \in \mathrm{SMU}(n, m)$ if and only if the following three conditions hold: (i) $\mathrm{DP}_x(F) \in \mathrm{SMU}(n-1, m-1)$, (ii) $C_0 \setminus \{x\} = \bigcap_{i=1}^{r} C_i \setminus \{\overline{x}\}$, and (iii) for every $C' \in F \setminus \{C_0, \ldots, C_r\}$ there is some literal $\ell \in C_0 \setminus \{x\}$ which does not belong to C'.*

The next lemma, a direct consequence of the preceding one, states that in the context of saturated minimally unsatisfiable formulas, DP-lifting is uniquely determined by a subset of the lifted formula.

Lemma 4. *Let $n \geq 2$ and let $F' \in \mathrm{SMU}(n-1, m-1)$. Then each formula $F \in \mathrm{SSMU}(n, m)$ which can be obtained from F' by DP-lifting on a singular literal x of F, can be generated by selecting r clauses $C'_1, \ldots, C'_r \in F'$ such that $\bigcap_{i=1}^{r} C'_i \not\subseteq C$ for any $C \in F \setminus \{C'_1, \ldots, C'_r\}$, and replacing them by the $r+1$ clauses $C_0, \ldots, C_r$ where $C_0 = \bigcap_{i=1}^{r} C'_i \cup \{x\}$ and $C_i = C'_i \cup \{\overline{x}\}$.*

The next lemma is useful when we know h_{m-1}, have a lower bound on h_m, and want to show that a formula F containing singular literals does not require longer proofs than our current bound on h_m, without laboriously computing a shortest proof of F.

Lemma 5. *Let $F \in \mathrm{MU}(n, m)$ with an r-singular variable. Then $h(F) \leq h_{m-1} + r + 1$.*

Proof. We perform DP-reduction on the r-singular variable using $r+1$ axioms, then refute the resulting formula on $m-1$ remaining clauses. □

The *deficiency* $\delta(F)$ of a formula F is defined as $|F| - |var(F)|$. By a lemma attributed to Tarsi [1], all minimally unsatisfiable formulas have a positive

deficiency. That means that a minimally unsatisfiable formula with a fixed number of clauses cannot have *too many* variables. It is easy to see that it cannot have *too few* variables either: each clause must be falsified by an assignment that satisfies every other clause, whence we infer that the number of assignments bounds the number of clauses. Putting the two inequalities together yields Lemma 6.

Lemma 6 (Aharoni and Linial [1]). *Let F be a minimally unsatisfiable formula. Then* $\log_2 |F| \leq |var(F)| < |F|$.

The structure of saturated minimally unsatisfiable formulas of deficiencies 1 and 2 is well understood [11]. In particular, it is known that for $m > 1$, each $F \in \mathrm{SMU}(m-1, m)$ has a 1-singular literal. It is also known that $|\mathrm{Iso}(\mathrm{RSMU}(m-2, m))| = 1$ for $m \geq 4$ [10] (otherwise, there are no minimally unsatisfiable formulas of deficiency 2). We pick the unique representative $\mathcal{F}^2_m$ for $\mathrm{Iso}(\mathrm{RSMU}(m-2, m))$, which consists of the clauses $\{\overline{x_1}, x_2\}, \ldots, \{\overline{x_{n-1}}, x_n\}, \{\overline{x_n}, x_1\}$, $\{x_1, \ldots, x_n\}$, and $\{\overline{x_1}, \ldots, \overline{x_n}\}$, $n = m - 2$.

Due to their simple structure, we can determine the resolution hardness of $\mathrm{SMU}(m-1, m)$ and $\mathrm{RSMU}(m-2, m)$ formulas without any computation.

Proposition 2. *For every $m \geq 1$, $h(\mathrm{SMU}(m-1, m)) = 2m - 1$.*

Proof. Apart from the formula $\{\square\}$, every formula from $\mathrm{SMU}(m-1, m)$ contains a 1-singular variable [4, Theorem 12], so the statement follows by induction from Lemma 5 and the fact that $2m - 1$ is the shortest possible proof length. □

Proposition 3. *For every $m \geq 4$, $h(\mathrm{RSMU}(m-2, m)) = h(\mathcal{F}^2_m) = 3m - 5$.*

Proof. $\mathcal{F}^2_m$ consists of binary strict Horn clauses (*BSH*—one negative and one positive literal) and the full positive and full negative clause. Resolving any pair of BSH clauses produces a BSH clause again. Resolving a BSH clause with a positive (negative) clause produces a positive (negative) clause, which is at most one shorter. Hence, to get to a positive (negative) unit clause, one must shorten the full positive (negative) clause at least $n - 1 = m - 3$ times. In total, we have m axioms plus $2(m-3)$ shortening steps plus a final resolution step, altogether $3m - 5$ proof lines. It is easy to see that such proof exists for every m. □

Propositions 2 and 3, together with Lemma 6, give us a lower bound for h_m.

Corollary 1. *For $m \leq 3$, $h_m = 2m - 1$. For $m \geq 4$, $h_m \geq 3m - 5$.*

Proof. For $m \leq 3$ Lemma 6 rules out formulas with deficiency higher than 1, showing $h_1 = 1$, $h_2 = 3$, and $h_3 = 5$. The rest is a direct consequence of Proposition 3. □

For other formulas, we will need to generate the formulas and compute their shortest proofs. Our general approach for computing $\mathrm{Iso}(H(m))$ and in turn h_m is to compute the sets $\mathrm{Iso}(\mathrm{SMU}(n, m))$ for $n = \lceil \log_2(m) \rceil, \ldots, m-1$, and test for each $F \in \mathrm{Iso}(\mathrm{SMU}(n, m))$ its resolution hardness $h(F)$ using the SAT encoding, which we describe in Sections 4 and 5.

We split the computation of $\mathrm{Iso}(\mathrm{SMU}(n, m))$ into two parts. We first generate $\mathrm{Iso}(\mathrm{RSMU}(n, m))$ for $\lceil \log_2(m) \rceil \leq n < m$. Due to Proposition 1, we can do this by

enumerating non-isomorphic 2-graphs, which correspond to clause-literal graphs of formulas in RSMU(n, m). We can limit ourselves to 2-graphs $G = (V_1 \uplus V_2, E)$ where $|V_1| = 2n$ and $|V_1| = m$, and where every vertex in V_1 has exactly one neighbor in V_1 and at least two neighbors in V_2. We use a tailor-made adaptation of the graph symmetry package Nauty [13] to enumerate such graphs; further details can be found in Section 6.

If n and m are such that $2^{n-1} < m-1$, we know there cannot be any formulas in SSMU(n, m) because singular DP-reduction would turn them into minimally unsatisfiable formulas on $n-1$ variables with $2^{n-1} < m-1$ clauses, but no such formulas exist. Hence, in those cases RSMU$(n, m) =$ SMU(n, m), and we already have Iso(SMU(n, m)). From these starting points, we repeatedly apply Lemma 4 to every formula in Iso(SMU(n, m)) to obtain Iso(SSMU$(n+1, m+1)$). Together with Iso(RSMU$(n+1, m+1)$) we then obtain Iso(SMU$(n+1, m+1)$).

The rationale for splitting the computation of SMU(n, m) into two pieces is the following. Enumerating non-isomorphic clause-literal graphs by Nauty for given parameters n and m is the hardest part of our computation. We often need to enumerate a significantly larger set than SMU(n, m). Therefore, we need to prune the enumeration phase as much as possible. When focusing on regular formulas, we can introduce additional bounds for Nauty, which significantly speed up the search. Applying then Lemma 4 inductively to the rather small set SMU(n, m) is computationally affordable (as long as the set SMU(n, m) remains reasonably small).

Since we are interested only in saturated minimally unsatisfiable formulas, we need to filter the graphs that we generate, which requires multiple calls to a SAT solver for every graph generated. Re-initializing the SAT solver with different formulas for different tests is expensive. Therefore it is desirable to bundle as many SAT calls together either by adding clauses incrementally or by using assumptions. While incrementally testing minimal unsatisfiability without solving multiple different formulas is relatively straightforward (via clause selector variables), it is not immediately clear how to do the same for saturation. We devised an algorithm that decides saturated minimal unsatisfiability using assumption-based calls to a SAT solver without the need to solve multiple different formulas. As a bonus, the formula for the saturation test contains all the clauses of the formula used for the minimality test, so both tests can proceed incrementally. The following lemma is the basis for our algorithm (a satisfiability test precedes the saturation test in our implementation, so it is safe to assume that the formula tested is unsatisfiable).

Lemma 7. *Let F be an unsatisfiable formula, C a clause of F, and $x \notin C$ a literal. The formula $E = F \cup (C \cup \{x\}) \setminus C$, where the clause C was extended with the literal x, is unsatisfiable if and only if the formula $G = F \cup \{\{x\}\} \setminus C$ is unsatisfiable.*

Proof. If E is unsatisfiable, so is G because every clause in G is a subset of some clause in E. Conversely, assume that E is satisfiable with the assignment τ. Because τ satisfies $F \setminus C$, and F is unsatisfiable, τ must falsify C, and so it must satisfy x. Hence, it satisfies G. □

Lemma 7 gives rise to the following algorithm: for all $C \in F$ and every literal $x \notin C$, check whether $F \cup \{x\} \setminus C$ is unsatisfiable. If so, the clause C can be extended with x preserving unsatisfiability, meaning that F is not saturated. This can be implemented in an assumption-based fashion with a single formula by augmenting F with all possible unit clauses, adding selector variables to every clause, and turning clauses on and off as necessary using assumptions.

4 Encoding for Shortest Resolution Proofs

This section gives the details of our SAT encoding computing the shortest resolution proof of an input formula. We aim to encode the following question.

Given a formula F with the clauses (*axioms*) $A_1, \ldots, A_m$ and $\mathit{var}(F) = V = \{x_1, \ldots, x_n\}$, does there exist a resolution refutation of F of length at most s, i.e., does there exist a sequence $P = L_1, \ldots, L_s$ of s lines (clauses), such that each L_i is either some axiom A_j or a resolvent of two previous $L_{i'}, L_{i''}$, $i', i'' < i$, and L_s is empty. We denote this problem by $\texttt{SHORT}(F, s)$.

It is easy to see that $\texttt{SHORT}(F, s)$ is coNP-hard (s given in binary): since each unsatisfiable formula F with n variables has a resolution refutation of length at most $2^{n+1} - 1$, we have $\texttt{UNSAT}(F) = \texttt{SHORT}(F, 2^{n+1} - 1)$. Therefore, using a SAT-based approach is indeed justified. On the other hand, membership in NEXPTIME can easily be seen as well: guess a refutation of length s and verify that it is correct. The precise complexity of $\texttt{SHORT}(F, s)$ is an open problem—our intuition, based on our inability to construct a deterministic single-exponential-time algorithm for $\texttt{SHORT}(F, s)$, is that it might be NEXPTIME-complete.

The basic idea of our encoding is to have variables $\mathrm{pos}[i, v]$ and $\mathrm{neg}[i, v]$ that determine whether v and $\overline{v}$ occur in L_i, and variables $\mathrm{arc}[i, j]$, which hold the information about the structure of the resolution steps in the proof. Together, these variables fully determine a candidate resolution proof sequence P. We additionally use auxiliary variables to express certain constraints more succinctly. Table 1 lists the core variables used by the encoding.

We drew inspiration from a similar encoding proposed by Marques-Silva and Mencía [14] (henceforth referred to as MSM), but took several departures, afforded by the fact that we focus on minimally unsatisfiable formulas. One of the strongest points of MSM, enumerating *minimal correction subsets* (MCSes, i.e., inclusion-minimal sets of clauses whose deletion renders the formula satisfiable) in a preprocessing step, becomes trivial for minimally unsatisfiable formulas: the MCSes are precisely all singletons by definition of minimal unsatisfiability. Instead, we require that every axiom of the input formula is used in the proof.

On the other hand, we extend the encoding with powerful symmetry breaking predicates. These predicates, explained in detail in Section 5, completely break all symmetries resulting from different permutations of the same sequence of clauses, and as such, they constitute a valuable theoretical contribution. Moreover, thanks to this additional symmetry breaking, we were able to compute shortest proofs of several formulas, for which MSM failed to produce an answer in hours of

running time. This symmetry breaking uses further auxiliary variables, which are introduced in Section 5.

Another novelty of our encoding is the capacity to reject a partially constructed proof early based on a counting argument involving the number of times a clause is used in resolution steps. We give the details at the end of this section.

variable	meaning	how many
pos$[i,v]$	$v \in L_i$	$O(ns)$
neg$[i,v]$	$\overline{v} \in L_i$	$O(ns)$
piv$[i,v]$	v is the pivot variable for the resolvent L_i	$O(ns)$
ax$[i,j]$	$L_i = A_j$	$O(ms)$
isax$[i]$	$\exists j : L_i = A_j$	$O(s)$
arc$[i,j]$	L_i is a premise of L_j	$O(s^2)$
upos$[i,v]$	v occurs in at least one premise of L_i	$O(ns)$
uneg$[i,v]$	$\overline{v}$ occurs in at least one premise of L_i	$O(ns)$
poscarry$[i,j,v]$	$v \in L_i$ and L_i is a premise of L_j	$O(ns^2)^*$
negcarry$[i,j,v]$	$\overline{v} \in L_i$ and L_i is a premise of L_j	$O(ns^2)^*$

Table 1. Variables used by the shortest-proof encoding. The symbol v is understood to range over V, while the symbols i, j range over the set $\{1, \ldots, s\}$ with $i < j$, except for ax$[i,j]$, where j ranges over $\{1, \ldots, m\}$ instead. The *-marked terms are asymptotically dominating ($m \leq s$).

In the following subsections, we list the clauses of the encoding, using complex Boolean expressions where convenient, and implicitly assuming that those are translated into a logically equivalent CNF in the natural way. Sometimes we write pos|neg to save space, meaning that the surrounding expression should be interpreted twice, with `pos` and `neg` substituted. We will also use cardinality constraints of the form $\sum_{x \in X} x \leq k$, which can be encoded using an arbitrary CNF cardinality-constraint encoding. We use the sequential counter [19], which seemed to perform best in our tests.

Definitions. First, we list all the clauses that provide definitions for the variables ax, isax, {pos|neg}carry, the union of premises via u{pos|neg}, and piv.

$$\bigwedge_{\substack{1 \leq i \leq s \\ 1 \leq j \leq m}} \text{ax}[i,j] \rightarrow \left(\bigwedge_{v \in A_j} \text{pos}[v,i] \bigwedge_{\overline{v} \in A_j} \text{neg}[v,i] \bigwedge_{v \notin var(A_j)} \overline{\text{pos}[v,i]} \wedge \overline{\text{neg}[v,i]} \right),$$

$$\bigwedge_{1\le i\le s}\left(\mathrm{isax}[i]=\bigvee_{1\le j\le m}\mathrm{ax}[i,j]\right),$$

$$\bigwedge_{1\le i,j\le s;v\in V}\{\mathrm{pos}|\mathrm{neg}\}\mathrm{carry}[i,j,v]=\{\mathrm{pos}|\mathrm{neg}\}[i,v]\wedge\mathrm{arc}[i,j],$$

$$\bigwedge_{1\le j\le s;v\in V}\left(\mathrm{u}\{\mathrm{pos}|\mathrm{neg}\}[j,v]=\bigvee_{1\le i<j}\{\mathrm{pos}|\mathrm{neg}\}\mathrm{carry}[i,j,v]\right),$$

$$\bigwedge_{1\le i\le s;v\in V}\mathrm{piv}[i,v]=\mathrm{upos}[i,v]\wedge\mathrm{uneg}[i,v].$$

Essential Constraints. The final clause is empty: $\bigwedge_{v\in V}\overline{\mathrm{pos}[i,v]}\wedge\overline{\mathrm{neg}[i,v]}$.

Axioms have no incoming arcs: $\bigwedge_{1\le i<j\le s}\mathrm{isax}[j]\rightarrow\overline{\mathrm{arc}[i,j]}$.

Clauses are non-tautological: $\bigwedge_{\substack{1\le i\le s\\ v\in V}}\overline{\mathrm{pos}[i,v]}\vee\overline{\mathrm{neg}[i,v]}$.

Non-pivot literals are retained after resolution.

$$\bigwedge_{1\le i\le s;v\in V}\overline{\mathrm{piv}[i,v]}\wedge\mathrm{u}\{\mathrm{pos}|\mathrm{neg}\}[i,v]\rightarrow\{\mathrm{pos}|\mathrm{neg}\}[i,v]$$

No new literals are introduced into resolvents.

$$\bigwedge_{1\le i\le s;v\in V}\overline{\mathrm{isax}[i]}\wedge\{\mathrm{pos}|\mathrm{neg}\}[i,v]\rightarrow\mathrm{u}\{\mathrm{pos}|\mathrm{neg}\}[i,v]$$

Every resolvent has a pivot: $\bigwedge_{1\le i\le s}\left(\overline{\mathrm{isax}[i]}\rightarrow\bigvee_{v\in V}\mathrm{piv}[i,v]\right)$, and the pivot is unique: $\bigwedge_{\substack{1\le i\le s\\ v\neq v'\in V}}\overline{\mathrm{piv}[i,v]}\vee\overline{\mathrm{piv}[i,v']}$. Every clause has exactly two premises[3].

$$\bigwedge_{3\le j\le s}\sum_{1\le i<j}\mathrm{arc}[i,j]=2$$

Redundant Constraints. If we search for the proof by iteratively incrementing the bound s, we know that every clause must be used: $\bigwedge_{1\le i<s}\bigvee_{i<j\le s}\mathrm{arc}[i,j]$. Axioms, do not have pivots: $\bigwedge_{\substack{1<i\le s\\ v\in V}}\mathrm{isax}[i]\rightarrow\overline{\mathrm{piv}[i,v]}$. We require that the axioms are placed at the beginning of the proof $\bigwedge_{1\le i<s}\mathrm{isax}[i+1]\rightarrow\mathrm{isax}[i]$, and in the same order as they appear in the original formula $\bigwedge_{1\le i\le s;1\le j_1\le j_2\le m}\overline{\mathrm{ax}[i,j_2]}\vee\overline{\mathrm{ax}[i+1,j_1]}$. Hence, A_j can appear no later than as L_j, expressed by the unit clauses $\overline{\mathrm{ax}[i,j]}$ for $1<i\le s$ and $1\le j\le\min(i-1,m)$. When considering only MU formulas, we can omit the above and directly place all axioms at the start in a fixed order: $\bigwedge_{i=1}^{m}\mathrm{ax}[i,i]\bigwedge_{i=m+1}^{s}\overline{\mathrm{isax}[i]}$.

[3] It would be enough to specify the at-most-two constraint here: the presence of at least two premises for resolvents is already enforced by the existence of a pivot and because the clauses are non-tautological: the pivot appears in both polarities in the union of premises (upos and uneg), which could not happen with one premise. Nevertheless, including both constraints appears to improve performance.

Counting the In- and Out-Degrees. For every sequence of clauses $P = L_1, \ldots, L_s$ that constitutes a resolution proof we can define a *directed acyclic graph (DAG)* $G(P)$ whose vertices are the clauses of P and which has the arcs (L_i, L_k) and (L_j, L_k) if L_k is a resolvent of L_i and L_j. Using the redundant constraints from above and assuming minimal unsatisfiability of F, we will show how one can place an additional redundant constraint on the proof DAG structure. This feature is based on the simple identity $\sum_{L \in G} d_{\text{out}}(L) = \sum_{L \in G} d_{\text{in}}(L)$, which holds in every directed graph G. In a proof DAG $G(P)$ of size s, axioms have in-degree 0 and resolvents have in-degree 2, so $\sum_{L \in G(P)} d_{\text{in}}(L) = 2(s - m)$. At the same time, every clause except for the last has out-degree at least 1. Therefore, at any time of the search, with $d'_{\text{out}}(L_i) \geq 0$ outgoing arcs already added to L_i in the partial DAG, it must hold that

$$\sum_{i=1}^{s-1} \max(d'_{\text{out}}(L_i), 1) \leq 2(s - m) \iff \sum_{i=1}^{s-1} \max(d'_{\text{out}}(L_i) - 1, 0) \leq s - 2m + 1.$$

We encode the latter inequality as a cardinality constraint. To capture the value of $\max(d'_{\text{out}}(L_i) - 1, 0)$, we introduce the notion of an *extra* arc: for a clause $L_i \in P$ with multiple outgoing arcs to clauses $L_{j_1}, \ldots, L_{j_k}$, $j_1 < \cdots < j_k$, we say that the arcs to $L_{j_2}, \ldots, L_{j_k}$ are extra. [4] Hence, $\max(d_{\text{out}}(L_i) - 1, 0)$ is precisely the number of extra outgoing arcs from L_i. We define the variables $\text{exarc}[i, j]$ whose meaning is that $\text{arc}[i, j]$ is an extra arc and enforce the cardinality constraint on them.

$$\bigwedge_{1 \leq i < j < k \leq s} \text{arc}[i, j] \wedge \text{arc}[i, k] \rightarrow \text{exarc}[i, k]; \qquad \sum_{1 \leq i < j \leq s} \text{exarc}[i, j] \leq s - 2m + 1.$$

Since the cardinality constraint is unsatisfiable if the right-hand side is negative, we additionally get that $s \geq 2m - 1$; indeed, a shorter proof could not use all m axioms. In other words, any proof of an MU formula must be "read-at-least-once."

5 Symmetry Breaking

Consider the proof DAG $G(P)$ of a resolution proof P. Any proof P is simply a topological sort of its DAG $G(P)$. If two sequences P_1 and P_2 share the same DAG $G(P_1) = G(P_2) = G$, then P_1 and P_2 are essentially the same proof. Our aim now is to make sure that for each candidate proof DAG G, exactly one topological sort is accepted by the encoding.

A directed acyclic graph can be topologically sorted by repeatedly picking and deleting from G a source vertex, i.e., one with no incoming arcs, as the next vertex in the resulting topologically sorted sequence. In the event that several sources are available, any one can be picked, which is why a given DAG, in general, has many topological sorts. We define a *canonical* topological sort of a

[4] This includes symmetry breaking: the single non-extra arc is the first one.

given DAG G in the following way. Let $\leq^*$ be an arbitrary total order on the vertices of G. The *canonical topological sort of* G is the topological sort that results from always picking the greatest source vertex under $\leq^*$. The idea for this symmetry breaking is due to Schidler and Szeider [18] who introduced it in a different context; Fichte et al. [5] further studied this technique under the name *LexTopSort*.

To verify that a given sequence P is the canonical topological sort of $G(P)$, we need to check that for every pair of vertices L_i, L_j, $i < j$, if L_j was a source already at the time when L_i was inserted, then $L_j \leq^* L_i$. We can check whether L_j was a source *simultaneously* with L_i by checking that there is no arc (L_k, L_j) with $i \leq k$. This is the role of the variables $\mathrm{sim}[i, j]$.

We also need to reason about the order $\leq^*$ on clauses. We define the following order on the literals $x_1 < \overline{x_1} < \cdots < x_n < \overline{x_n}$, and order clauses of the proof lexicographically based on this order: $L_i <^* L_j$ if there is a literal $l \in L_j$ such that $l \notin L_i$ and $\{l'\} \cap L_i = \{l'\} \cap L_j$ for all $l' < l$. We represent $\leq^*$ using the variables $\mathrm{equal}[i, j, l]$, which say that the clauses L_i and L_j are equal up to position l in the ordering of the literals, and the corresponding constraints below.

$$\bigwedge_{1\leq i<j\leq s} \mathrm{equal}[i,j,x_1] = (\mathrm{pos}[i,x_1] \iff \mathrm{pos}[j,x_1])$$

$$\bigwedge_{1\leq i<j\leq s;\ 1<k\leq n} \mathrm{equal}[i,j,x_k] = \mathrm{equal}[i,j,\overline{x_{k-1}}] \wedge (\mathrm{pos}[i,x_k] \iff \mathrm{pos}[j,x_k])$$

$$\bigwedge_{1\leq i<j\leq s;\ 1\leq k\leq n} \mathrm{equal}[i,j,\overline{x_k}] = \quad \mathrm{equal}[i,j,x_k] \wedge (\mathrm{neg}[i,x_k] \iff \mathrm{neg}[j,x_k])$$

Definition of sim: for $1 \leq i < s$, $\mathrm{sim}[i, i+1] = \overline{\mathrm{arc}[i, i+1]}$, and

$$\bigwedge_{1\leq i<j-1\leq s} \mathrm{sim}[i,j] = \mathrm{sim}[i+1,j] \wedge \overline{\mathrm{arc}[i,j]}.$$

The following constraint enforces that the sequence is the canonical topological sort (for resolvents only, the order of axioms is handled differently—see Section 4).

$$\bigwedge_{1\leq i<j\leq s} \left(\mathrm{sim}[i,j] \wedge \overline{\mathrm{ax}[i]}\right) \to (\mathrm{pos}[i,x_1] \geq \mathrm{pos}[j,x_1])$$

$$\bigwedge_{\substack{1\leq i<j\leq s\\ 1\leq k\leq n}} \left(\mathrm{sim}[i,j] \wedge \overline{\mathrm{ax}[i]} \wedge \mathrm{equal}[i,j,x_k]\right) \to (\mathrm{neg}[i,x_k] \geq \mathrm{neg}[j,x_k])$$

$$\bigwedge_{\substack{1\leq i<j\leq s\\ 1\leq k<n}} \left(\mathrm{sim}[i,j] \wedge \overline{\mathrm{ax}[i]} \wedge \mathrm{equal}[i,j,\overline{x_k}]\right) \to (\mathrm{pos}[i,x_{k+1}] \geq \mathrm{pos}[j,x_{k+1}])$$

$$\bigwedge_{1\leq i<j\leq s} \overline{\mathrm{equal}[i,j,\overline{x_n}]}$$

The following theorem summarizes the properties of our encoding.

Theorem 1. *Let F be a propositional formula on n variables and m clauses and let $\texttt{short}_s(F)$ be the formula defined above. Then the following statements hold:*

1. *the size of $\texttt{short}_s(F)$ is polynomial in $\max(n, m, s)$ (s can be exponential in the input length);*
2. *$\texttt{short}_s(F)$ is satisfiable if and only if F has a resolution refutation of length s in which every clause is used to derive the empty clause;*
3. *any model of $\texttt{short}_s(F)$ can naturally be interpreted as a sequence of clauses P that constitutes a valid resolution proof of F;*
4. *P is the canonical topological sort of $G(P)$.*

Theorem 1 gives rise to a simple algorithm. Start with $s = 1$, and increment s by one while $\texttt{short}_s(F)$ is unsatisfiable. As soon as $\texttt{short}_s(F)$ becomes satisfiable, s is the length of a shortest resolution refutation of F, and the refutation itself can be extracted from a model of $\texttt{short}_s(F)$. An improvement is possible for MU formulas, by starting not at $s = 1$, but $s = 2m - 1$, as described in Section 4.

6 Experiments

In this section, we describe how we performed our computations. We will refer to formulas and graphs interchangeably throughout this section, saying for instance that a graph is minimally unsatisfiable. In such cases, it is understood that we are using the correspondence between formulas and graphs sketched in Section 3, and implicitly mean the corresponding object.

To generate Iso(RSMU(n, m)), we run a modified version of the `genbg` utility[5] from the graph automorphism package `Nauty` [13], which enumerates isomorph-free 2-graphs. The modification is that the graphs generated are not bipartite as in `genbg`, but V_1 induces a *matching*, i.e., the graph is a clause-literal graph as defined in Section 2. We run `genbg` with the parameters `-cAtd3:2 2n m`, meaning we are interested in connected (`-c`) triangle-free (`-t`) 2-graphs $G = (V_1 \uplus V_2, E)$, such that V_1 has $2n$ vertices (the literals) whose minimum degree is 3 (every literal should occur twice, plus the edge between the two literals of a variable), and V_2 has m vertices (the clauses) with minimum degree 2 (a unit clause would imply a singular literal, so we can skip such graphs), and such that no two vertices V_2 have neighborhoods that are subsets of one another (`-A`). This gives us a set S of graphs that contains Iso(RSMU(n, m)), and such that all graphs in S represent formulas without tautological clauses (triangle-freeness), without singular literals (degree bounds), and without subsumed clauses (`-A`). Hence it remains to filter the output of `genbg` for saturated minimal unsatisfiability. For that purpose we use CryptoMiniSAT [20] via its `C` API, in essentially the natural way of testing for saturated minimal unsatisfiability, however, with some technical optimizations worth mentioning.

[5] We gratefully acknowledge the help of Brendan McKay, author of Nauty, who provided a modification of `genbg` for our purpose.

It turns out that for small values of n and m, the vast majority of graphs generated are satisfiable[6]. This, in turn, means that the vast majority of time is spent checking satisfiability, which happens to be relatively expensive due to the need to compute an explicit representation of the clauses and to re-initialize the solver for every graph from scratch. Upon realizing this limitation, we implemented a carefully tuned and highly cache-performant brute-force satisfiability test that uses Nauty's data structures directly and simply checks every assignment, and observed an orders-of-magnitude speed-up.

To check minimal unsatisfiability of a formula F, we construct $F' = \{ C \cup \{\overline{x_C}\} \mid C \in F \}$ with the fresh selector variables x_C, and solve all formulas $F'[\tau_C]$, where τ_C sets x_C to 0 and all other $x_{C'}$ to 1, via assumptions.

If F passes the minimal unsatisfiability test, we build $F'' = F' \cup \{ \{\ell, x_\ell\} \mid \ell \in \mathit{lit}(F) \}$ by adding the extra clauses to F', which is already loaded in the SAT solver, and test for saturation by solving the formulas $F''[\tau_{C,\ell}]$, where $\tau_{C,\ell}$ sets x_C and x_ℓ to 0 (turns off the clause C and turns on the unit clause $\{\ell\}$) and all other auxiliary variables to 1, again via assumptions.

Once we have generated $\mathrm{Iso}(\mathrm{RSMU}(n,m))$, which is equal to $\mathrm{Iso}(\mathrm{SMU}(n,m))$ for values of n, m where $\mathrm{MU}(n-1,m-1)$ is empty, we use Lemma 4 to compute $\mathrm{SMU}(n+1,m+1)$. Whenever we have computed $\mathrm{SMU}(n+1,m+1)$, we simply run our encoding on every formula, incrementally increasing the proof length bound s, and compute all shortest proofs.

We implemented the encoding and the iterative search for a shortest proof in Python using the PySAT framework [9]. We concluded from our initial tests that among the SAT solvers available in PySAT, CaDiCaL [7] performed best, and we decided to stick with it.

Table 2 lists the length of the longest shortest proof required by an $\mathrm{SMU}(n,m)$ formula, and, by taking the maximum in each column, also values of h_m. In particular, we obtain the first ten resolution hardness numbers:

$$(h_m)_{m\geq 1} = 1, 3, 5, 7, 10, 13, 16, 19, 22, 26, \ldots$$

We observe the interesting phenomenon that all computed formulas attaining the maximum hardness h_m are regular.

It is known that every $\mathrm{MU}(n,m)$ formula has a proof of length at most $2^{m-n-1}n + m$ [11, Section 11.3], and, along with the existence of formulas which require exponentially long proofs, this implies that maximum hardness cannot forever be attained by formulas of any bounded deficiency $m - n$. Our computations reveal that $m = 10$ is the tipping point where formulas of deficiency 2 "drop out of the race," as there is no longer a hardest formula of deficiency 2, see Table 2. Up to isomorphism, there are exactly three hardest formulas for $m = 10$, all of which are of deficiency 4. Figure 1 shows the clause-literal graphs of these three formulas.

Our encoding [16] and our catalog of SMU formulas [15] are publicly available.

[6] As an example, for $n = 5$ and $m = 9$, a total of 9356316116 out of the 9360503942 generated graphs were satisfiable. The proportion was similar for other parameters.

[7] https://fmv.jku.at/cadical

$n\backslash m$	1	2	3	4	5	6	7	8	9	10
0	**1**	-	-	-	-	-	-	-	-	-
1	-	**3**	-	-	-	-	-	-	-	-
2	-	-	**5**	**7**(1)	-	-	-	-	-	-
3	-	-	-	**7**	**10**(1)	11(3)	13(1)	15(1)	-	-
4	-	-	-	-	9	**13**(1)	15(1)	**19**(1)	20(1)	21(5)
5	-	-	-	-	-	11	**16**(1)	18(3)	**22**(1)	25(1)
6	-	-	-	-	-	-	13	**19**(1)	**22**(3)	**26**(3)
7	-	-	-	-	-	-	-	15	**22**(1)	25(5)
8	-	-	-	-	-	-	-	-	17	25(1)
9	-	-	-	-	-	-	-	-	-	19

Table 2. Values of $h(H(n,m))$, and in parenthesis the number of formulas in RSMU(n,m), up to isomorphism, that require resolution proofs of length $h(H(n,m))$. For all $3 \leq n \leq 9$ and $n+2 \leq m \leq 10$, we found $H(n,m) \subseteq \text{RSMU}(n,m)$, except for $H(7,10)$ which also contains 19 singular formulas up to isomorphism. By Proposition 2, $h(H(m-1,m)) = 2m-1$ and so, no computation is necessary. By Lemma 6, there are no minimally unsatisfiable formulas in the areas marked by a hyphen.

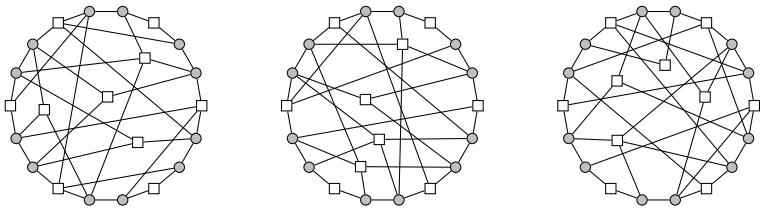


Fig. 1. Clause-literal graphs of the three hardest formulas with 10 clauses.

7 Conclusion

We conducted an extensive computational investigation into resolution hardness. First, we developed theoretical foundations that allowed us to pinpoint classes of formulas of maximum resolution hardness. Then, using a tight graph representation of formulas and carefully tuned generation procedures, we computed all candidates for hardest formulas for up to ten clauses. With this information, and using a SAT encoding for the computation of shortest resolution proofs targeted towards minimally unsatisfiable formulas and with powerful novel symmetry breaking, we calculated the first ten resolution hardness numbers. Our results indicate that regular saturated minimally unsatisfiable formulas achieve the highest hardness. It remains as an interesting theoretical question whether the hardest formulas are always regular.

References

1. Aharoni, R., Linial, N.: Minimal non-two-colorable hypergraphs and minimal unsatisfiable formulas. J. Combin. Theory Ser. A **43**, 196–204 (1986)
2. Codish, M., Miller, A., Prosser, P., Stuckey, P.J.: Constraints for symmetry breaking in graph representation. Constraints An Int. J. **24**(1), 1–24 (2019). https://doi.org/10.1007/s10601-018-9294-5, https://doi.org/10.1007/s10601-018-9294-5
3. Davis, M., Putnam, H.: A computing procedure for quantification theory. J. of the ACM **7**(3), 201–215 (1960)
4. Davydov, G., Davydova, I., Kleine Büning, H.: An efficient algorithm for the minimal unsatisfiability problem for a subclass of CNF. Ann. Math. Artif. Intell. **23**, 229–245 (1998)
5. Fichte, J.K., Hecher, M., , Szeider, S.: Breaking symmetries with RootClique and LexTopSort. In: Simonis, H. (ed.) Proceedings of CP 2020, the 26rd International Conference on Principles and Practice of Constraint Programming. Lecture Notes in Computer Science, Springer Verlag (2020), this volume
6. Haken, A.: The intractability of resolution. Theoretical Computer Science **39**, 297–308 (1985)
7. Heule, M.J.H.: Schur number five. In: Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018. pp. 6598–6606 (2018)
8. Heule, M.J.H., Kullmann, O., Marek, V.W.: Solving and verifying the Boolean Pythagorean Triples problem via cube-and-conquer. In: Creignou, N., Berre, D.L. (eds.) Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings. Lecture Notes in Computer Science, vol. 9710, pp. 228–245. Springer Verlag (2016)
9. Ignatiev, A., Morgado, A., Marques-Silva, J.: PySAT: A Python toolkit for prototyping with SAT oracles. In: SAT. pp. 428–437 (2018). https://doi.org/10.1007/978-3-319-94144-8_26, https://doi.org/10.1007/978-3-319-94144-8_26
10. Kleine Büning, H.: On subclasses of minimal unsatisfiable formulas. Discr. Appl. Math. **107**(1–3), 83–98 (2000)
11. Kleine Büning, H., Kullmann, O.: Minimal unsatisfiability and autarkies. In: Biere, A., Heule, M.J.H., van Maaren, H., Walsh, T. (eds.) Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185, chap. 11, pp. 339–401. IOS Press (2009)
12. Kullmann, O., Zhao, X.: On Davis-Putnam reductions for minimally unsatisfiable clause-sets. Theoretical Computer Science **492**, 70–87 (2013)
13. McKay, B.D., Piperno, A.: Practical graph isomorphism, {II}. Journal of Symbolic Computation **60**(0), 94 – 112 (2014). https://doi.org/http://doi.org/10.1016/j.jsc.2013.09.003, http://www.sciencedirect.com/science/article/pii/S0747717113001193
14. Mencía, C., Marques-Silva, J.: Computing shortest resolution proofs. In: Oliveira, P.M., Novais, P., Reis, L.P. (eds.) Progress in Artificial Intelligence, 19th EPIA Conference on Artificial Intelligence, EPIA 2019, Vila Real, Portugal, September 3-6, 2019, Proceedings, Part II. Lecture Notes in Computer Science, vol. 11805, pp. 539–551. Springer (2019). https://doi.org/10.1007/978-3-030-30244-3_45, https://doi.org/10.1007/978-3-030-30244-3_45

15. Peitl, T., Szeider, S.: Saturated minimally unsatisfiable formulas on up to ten clauses (Jul 2020). https://doi.org/10.5281/zenodo.3951546
16. Peitl, T., Szeider, S.: short.py: Encoding for the shortest resolution proof (Jul 2020). https://doi.org/10.5281/zenodo.3952168
17. Pipatsrisawat, K., Darwiche, A.: On the power of clause-learning SAT solvers with restarts. In: Gent, I.P. (ed.) Principles and Practice of Constraint Programming - CP 2009, 15th International Conference, CP 2009, Lisbon, Portugal, September 20-24, 2009, Proceedings. Lecture Notes in Computer Science, vol. 5732, pp. 654–668. Springer Verlag (2009)
18. Schidler, A., Szeider, S.: Computing optimal hypertree decompositions. In: Blelloch, G., Finocchi, I. (eds.) Proceedings of ALENEX 2020, the 22nd Workshop on Algorithm Engineering and Experiments. pp. 1–11. SIAM (2020)
19. Sinz, C.: Towards an optimal cnf encoding of boolean cardinality constraints. In: van Beek, P. (ed.) Principles and Practice of Constraint Programming - CP 2005, 11th International Conference, CP 2005, Sitges, Spain, October 1-5, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3709, pp. 827–831. Springer Verlag (2005)
20. Soos, M., Nohl, K., Castelluccia, C.: Extending SAT solvers to cryptographic problems. In: Kullmann, O. (ed.) Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5584, pp. 244–257. Springer (2009). https://doi.org/10.1007/978-3-642-02777-2_24, https://doi.org/10.1007/978-3-642-02777-2_24
21. Szeider, S.: Minimal unsatisfiable formulas with bounded clause-variable difference are fixed-parameter tractable. J. of Computer and System Sciences **69**(4), 656–674 (2004)
22. Urquhart, A.: The complexity of propositional proofs. Bull. of Symbolic Logic **1**(4), 425–467 (Dec 1995)