Journal of Artificial Intelligence Research 72 (2021) 69-97 Submitted 01/2021; published 09/2021

Finding the Hardest Formulas for Resolution

Tomáš Peitl TOMAS.PEITL@UNI-JENA.DE
Institute of Computer Science, Friedrich Schiller University Jena
Ernst-Abbe-Platz 2,
07743 Jena, Germany

Stefan Szeider SZ@AC.TUWIEN.AC.AT
Institute of Logic and Computation, TU Wien
Favoritenstraße 9-11,
1040 Wien, Austria

Abstract

A CNF formula is harder than another CNF formula with the same number of clauses if it requires a longer resolution proof. In this paper, we introduce *resolution hardness numbers*; they 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. To achieve this, we devise a candidate filtering and symmetry breaking search scheme for limiting the number of potential candidates for hardest formulas, and an efficient SAT encoding for computing a shortest resolution proof of a given candidate formula.

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 (Pipatsrisawat & Darwiche, 2009).[1] 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 (Haken, 1985), we have an exponential lower bound on worst-case running time. Propositional proof complexity (Urquhart, 1995) studies resolution complexity, focussing mainly on asymptotic analysis. In this paper, we investigate resolution complexity in *exact*, rather than asymptotic terms. As the central object of our study, we define the *resolution hardness numbers* $(h_m)_{m \geq 1}$ to denote the highest resolution complexity of a formula with m clauses, taking some inspiration from the famous Busy-Beaver numbers.[2] In contrast to the Busy-Beaver numbers, the resolution hardness numbers are in principle computable, however, the search space grows rapidly with m, and so a practical computation is challenging already for relatively small values of m.

1. More precisely, resolution complexity provides a lower bound on the running time of *conflict-driven clause learning (CDCL)* (Silva & Sakallah, 1996), on which modern SAT solvers are based. Supplemental solving techniques may prevail against the resolution lower bound.
2. The n-th Busy-Beaver number gives the longest run of a terminating Turing machine with two symbols and n states on an empty tape (Rado, 1962; Michel, 2019).

As our main contribution, we compute the first ten resolution hardness numbers. In order to do so, we have to efficiently generate sets of formulas $\chi(m)$, for $m = 1, \ldots, 10$, which contain at least one hardest formula, and compute their resolution complexity. We obtain our results by the combination of two techniques:

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 structural constraints. We model these formulas by suitable graphs, and generate these graphs modulo isomorphisms by a special adaptation of the Nauty graph symmetry package (McKay & Piperno, 2014).

This still leaves us with a large number of formulas whose resolution complexity we must determine algorithmically. For this task, following a recent trend in tackling combinatorial problems using SAT and CSP methods (Heule, Kullmann, & Marek, 2016; Heule, 2018; Codish, Miller, Prosser, & Stuckey, 2019), we devised an efficient SAT encoding that produces a CNF formula $\texttt{short}_s(F)$, for a given candidate formula F and integer s, 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. Encoding resolution proofs is a common theme in propositional proof complexity (Pudlák, 2003; Atserias & Bonet, 2004), was used with the goal of deploying local search for unsatisfiability (Prestwich & Lynce, 2006), and recently, an encoding has been proposed specifically for the computation of shortest proofs (Mencía & Marques-Silva, 2019). We do take some inspiration from these previous works, but make crucial novel adaptations tailored towards minimally unsatisfiable formulas. Furthermore, we introduce a symmetry-breaking scheme that fully breaks all symmetries resulting from permuting the clauses in a proof.

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 of variables and a particular number 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 ν_n, the highest resolution complexity of formulas with n *variables*. It is not hard to see by induction on n that every unsatisfiable formula on n variables has a resolution refutation of length $\leq 2^{n+1} - 1$ and that indeed $\nu_n = 2^{n+1} - 1$, witnessed by the formula which contains all possible clauses of width n.

After some preliminaries, we discuss the theory that allows us to reduce the search space in Section 3. Sections 4 and 5 are devoted to our SAT encoding of the shortest resolution proof problem and the novel symmetry breaking techniques. We discuss implementation details in Section 6, followed by a detailed analysis of our results in Section 7, and conclude with a summary and some directions and open questions in Section 8. The paper is an extended version of our CP paper (Peitl & Szeider, 2020), augmented with the following additions:

- a discussion of the existence of hardness numbers at the beginning of Section 3;
- Lemma 2, and thanks to it a simplified proof of Lemma 3;
- Algorithm 1 for brute-force testing of saturated minimal unsatisfiability;
- a more detailed exposition of the encoding and symmetry breaking;
- and a significantly extended analysis of the results including several new figures.

2. Preliminaries

Graphs. We assume familiarity with standard notions of graph theory, including those of *(un)directed graphs*, *acyclicity*, and *in-* and *out-degree* of a vertex—we refer to the standard graph theory handbook (Diestel, 2012). All graphs considered, directed or undirected, do not contain any self-loops or parallel edges.

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 write $\overline{\ell} := \neg x$ if $\ell = x$, and $\overline{\ell} := x$ if $\ell = \neg x$. For a set C of literals, we define $\overline{C} := \{\, \overline{\ell} \mid \ell \in C \,\}$; we say 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 $\mathit{lit}(F) := \mathit{var}(F) \cup \overline{\mathit{var}(F)}$. An *assignment* is a mapping $\tau : \mathit{var}(F) \to \{0, 1\}$. A formula F is *satisfiable* if there is a *satisfying assignment*, i.e., 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 $\tau(x) = 0$; otherwise it is *unsatisfiable*. 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* (also *resolution proof*; we use these terms interchangably) of F. A line L_i in a resolution derivation may have different possible "histories." 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 single history is associated with each considered resolution derivation. Thus, with a proof P we can associate the directed acyclic graph (*proof DAG*) $G(P)$ whose vertices are the proof lines, and which has an arc from L_i to L_j if there is L_k, $i, k < j$, such that L_j is the resolvent of L_i and L_k. For a DAG G and vertex v, we write $d_{\text{in}}(v)$ and $d_{\text{out}}(v)$ for the *in-* and *out-degree* of v in G. In any proof DAG $G(P)$, the in-degree of each axiom vertex is 0, while each resolvent has in-degree 2.

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 (Davis & Putnam, 1960). 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 define $h(F) := -\infty$). For a nonempty set $\mathcal{C}$ of formulas, we define $h(\mathcal{C}) = \sup_{F \in \mathcal{C}} h(F)$.

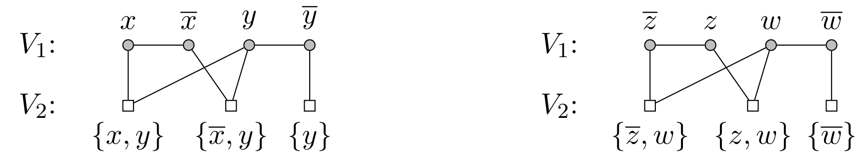
Isomorphisms of Formulas. Two formulas F and F' are *isomorphic* if there exists a bijection $\varphi : \mathit{lit}(F) \to \mathit{lit}(F')$ such that for each literal $\ell \in \mathit{lit}(F)$ we have $\overline{\varphi(\ell)} = \varphi(\overline{\ell})$ and for each $C \subseteq \mathit{lit}(F)$ we have $C \in F$ if and only if $\varphi(C) \in F'$. For instance the formulas $F = \{\{x, y\}, \{\overline{x}, y\}, \{\overline{y}\}\}$, 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 $\text{Iso}(\mathcal{C})$ to be an inclusion-maximal subset of $\mathcal{C}$ such that no two elements of $\text{Iso}(\mathcal{C})$ are isomorphic. In other words, $\text{Iso}(\mathcal{C})$ contains exactly one representative from each isomorphism class.

For sets S, S', T, we write $T = S \uplus S'$ if $T = S \cup S'$ and $S \cap S' = \emptyset$. A *2-graph* is an undirected graph $G = (V, E)$ together with a partition of its vertex set into two disjoint subsets $V_1 \uplus V_2 = V$. 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 $E = \{\, \{x, \overline{x}\} \mid x \in \mathit{var}(F)\,\} \cup \{\, \{C, \ell\} \mid C \in F, \ell \in C\,\}$. We refer to the edges $\{\, x, \overline{x}\,\}$ as *variable* edges.

For instance, the formulas F and F' mentioned above give rise to the following two isomorphic clause-literal graphs.



The following statement is easy to verify.

Proposition 1. *Two formulas are isomorphic if and only if their clause-literal graphs are isomorphic (as 2-graphs).*

3. Theoretical Framework

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

$$h_m = \max_{F \in \mathrm{CNF}(m)} h(F) = h(\mathrm{CNF}(m)). \tag{1}$$

We will also write $H(m) = \{\, F \in \mathrm{CNF}(m) \mid h(F) = h_m \,\}$ for the set of hardest formulas with m clauses.

Our general approach to computing h_m is to generate algorithmically a subset $\chi(m) \subseteq \mathrm{CNF}(m)$ of candidate hardest formulas of which at least one is guaranteed to have hardness h_m (i.e., $\chi(m) \cap H(m) \neq \emptyset$), and then compute shortest proof lengths of every formula in $\chi(m)$ using the encoding described in Sections 4 and 5; the maximum hardness found is h_m. Naturally, we could just look at all formulas with m clauses, i.e. $\chi(m) = \mathrm{CNF}(m)$, but that would be prohibitive, and so we want to restrict the size of $\chi(m)$ as much as possible. Accordingly, in this section we discuss various formula properties that allow us to restrict the candidate set $\chi(m)$.

Before we proceed with that, we should spare a word on whether h_m is even well-defined, i.e., whether the maximum taken in Equation 1 is finite. This is not a priori clear, since a formula with m clauses can have an unbounded number of variables, and so $\mathrm{CNF}(m)$ is, in fact, infinite. The infinitude, however, comes only from the ability to cram an unbounded number of variables into a single clause, which, as we shall show, cannot result in infinitely increasing hardness. The notion that is useful here is that of the *lean kernel* of a formula. Sparing ourselves a formal definition, all we need is that every unsatisfiable formula has a unique lean kernel, and it has the following key properties.

Lemma 1 (Corollary 3.8 (Kullmann, 2000c), Lemma 3.7 (Kullmann, 2000a))**.** *Let F be an unsatisfiable formula. Then, there is a subset $\mathcal{L}(F) \subseteq F$ (namely the lean kernel) with the following properties:*

- *every resolution refutation of F uses only clauses from $\mathcal{L}(F)$; and*
- *$\mathcal{L}(F)$ has more clauses than variables.*

Lemma 1 in particular implies that $h(F) = h(\mathcal{L}(F))$, because F and $\mathcal{L}(F)$ have the same refutations. Hence, the maximum in Equation 1 can just as well be taken over lean kernels of formulas from $\mathrm{CNF}(m)$. But there is only a finite number of lean kernels with $\leq m$ clauses, because their number of variables is bounded by their number of clauses, and so the maximum is indeed well defined and finite.

Having cleared the question of their existence, we begin by showing the unsurprising basic fact that the resolution hardness numbers form an increasing sequence.

Lemma 2. *$h_{m+1} \geq h_m + 2$.*

Proof. Let $F \in H(m)$, let $x \notin var(F)$ be a fresh variable. Let $G = \{\{x\}\} \cup \{C \cup \{\overline{x}\} \mid C \in F\}$. Since $F = G|_x$, any proof of G can be used to obtain a proof of F of the same or smaller size. Since $G \setminus \{\{x\}\}$ is not unsatisfiable, every proof of G contains the axiom $\{x\}$, and consequently at least one resolvent of that axiom. Hence, by restricting by x, the proof loses at least two clauses—the axiom, and at least one resolution step that used it. The statement of the lemma follows. □

Lemma 3. *All formulas in $H(m)$ are minimally unsatisfiable.*

Proof. Let $F \in H(m)$, and let $F' \subseteq F$ be minimally unsatisfiable. Because $F' \subseteq F$, every proof of F' is also a proof of F, and thus $h_m = h(F) \leq h(F') \leq h_{|F'|}$. By monotonicity of the hardness sequence, established by Lemma 2, we thus have $m \leq |F'|$. Since $F' \subseteq F$, this means $F' = F$, and by extension that F is minimally unsatisfiable. $\square$

Lemma 3 means that from now on we can restrict our attention to minimally unsatisfiable formulas. An easy counting argument gives us a very simple lower bound for the hardness of minimally unsatisfiable formulas.

Lemma 4. *Let F be a minimally unsatisfiable formula with m clauses. Then $h(F) \geq 2m-1$.*

Proof. Consider the proof DAG $G(P)$ of a shortest proof P of F. Because F is minimally unsatisfiable and P is a shortest proof, we know that every line $L \in P$, axiom or derived, other than the final empty clause, is used in some resolution step, and hence

$$\sum_{1 \leq i \leq |P|} d_{\text{out}}(L_i) \geq |P| - 1.$$

On the other hand, every resolution step has exactly two premises, and the m axioms have no premises, so

$$\sum_{1 \leq i \leq |P|} d_{\text{in}}(L_i) = 2(|P| - m).$$

The statement follows from the fact that the sum of in- and out-degrees must be equal in any directed graph. $\square$

A formula is *saturated minimally unsatisfiable (SMU)* 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 literal whose variable is not yet present in the formula to a clause has the same effect as deleting the clause.

Example 1. Let $F = \{\{x, y\}, \{\overline{x}\}, \{\overline{y}\}\}$. It is easy to see that F is unsatisfiable, but removing any clause makes it satisfiable—in other words F is minimally unsatisfiable. But F is not saturated; we can add the literal y to the clause $\{\overline{x}\}$ and the resulting formula $F' = \{\{x, y\}, \{\overline{x}, y\}, \{\overline{y}\}\}$ is still (minimally) unsatisfiable. F', though, cannot be extended any further this way: adding either x or $\overline{x}$ to the clause $\{\overline{y}\}$ does make the formula satisfiable. Thus, F' is saturated minimally unsatisfiable.

Lemma 5. *$H(m)$ contains a saturated minimally unsatisfiable formula.*

Proof. Let F be an arbitrary formula in $H(m)$ ($H(m)$ cannot be empty by definition). By Lemma 3, 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 an 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 $F' \in H(m)$. $\square$

Saturated minimally unsatisfiable formulas have two key properties that make them suitable for our purpose: they are the hardest of minimally unsatisfiable formulas; and they have a well studied structure we can exploit.

3.1 Saturated Minimal Unsatisfiability and Singular Literals

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 $r \geq 0$ (Szeider, 2004). 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 (Szeider, 2004). We say that $\mathrm{DP}_x(F)$ is obtained from F by *Davis-Putnam reduction* or short *DP-reduction* on x (Davis & Putnam, 1960). For an in-depth study of DP-reduction, particularly with respect to minimal unsatisfiability, we refer to a work of Kullmann and Zhao (Kullmann & Zhao, 2013). 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*.[3]

The following result by Kullmann and Zhao (2013, Lemma 12) establishes an important link between DP-reduction on a singular variable and saturated minimal unsatisfiability.

Lemma 6 (Kullmann & Zhao, 2013)**.** *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 $C_1, \dots, C_r$ 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:*

1. $\mathrm{DP}_x(F) \in \mathrm{SMU}(n-1, m-1)$,
2. $C_0 \setminus \{x\} = \bigcap_{i=1}^{r} C_i \setminus \{\overline{x}\}$, *and*
3. *for every $C' \in F \setminus \{C_0, \dots, C_r\}$ there is some literal $\ell \in C_0 \setminus \{x\}$ which does not belong to C'.*

We will turn Lemma 6 into an algorithm that performs DP-lifting. The task we need to accomplish is the following: given a formula F, generate all formulas F' such that $F = \mathrm{DP}_x(F')$ for a fresh variable $x \notin \mathit{var}(F)$. Lemma 7, a direct consequence of Lemma 6, shows that in the context of saturated minimally unsatisfiable formulas, DP-lifting is uniquely determined by a (suitable) subset of the clauses of the lifted formula. Hence, it is sufficient to go through all subsets of F, and generate the corresponding unique lifting for each, if there is any.

3. The formula G used in the proof of Lemma 2 was DP-lifted from F.

Lemma 7. *Let $m > n \geq 1$ and let $F' \in \mathrm{SMU}(n-1, m-1)$. 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 $C_i = C_i' \cup \{\overline{x}\}$ and $C_0 = \bigcap_{i=1}^r C_i' \cup \{x\}$.*

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 8. *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 (Aharoni & Linial, 1986), 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: for each clause there must be an assignment that falsifies it while satisfying every other clause, whence we infer that the number of assignments bounds the number of clauses. Putting the two inequalities together yields Lemma 9.

Lemma 9 (Aharoni & Linial, 1986)**.** *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 (Kleine Büning & Kullmann, 2009). 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 $m \geq 4$ (Kleine Büning, 2000, $m \geq 4$ because otherwise, there are no minimally unsatisfiable formulas of deficiency 2). We pick the unique representative $\mathcal{F}_m^2$ for $\mathrm{Iso}(\mathrm{RSMU}(m-2, m))$, which consists of the clauses ($n = m-2$)

$$\{\overline{x_1}, x_2\}, \ldots, \{\overline{x_{n-1}}, x_n\}, \{\overline{x_n}, x_1\}, \{x_1, \ldots, x_n\}, \{\overline{x_1}, \ldots, \overline{x_n}\}.$$

Thanks to their simple structure, we can determine the resolution hardness of both $\mathrm{SMU}(m-1, m)$ 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 (Davydov, Davydova, & Kleine Büning, 1998, Theorem 12), so the statement follows by induction from Lemma 8 (upper bound) and Lemma 4 (lower bound). □

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

Proof. $\mathcal{F}_m^2$ 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 literal 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 a proof exists for every m. $\square$

Propositions 2 and 3, together with Lemma 9, give us a refined 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 9 rules out formulas with deficiency higher than 1, showing $h_1 = 1$, $h_2 = 3$, and $h_3 = 5$ by Proposition 2. The rest is a direct consequence of Proposition 3. $\square$

In order to determine h_m for $m \geq 4$, we will need to generate a set $\chi(m)$ that contains at least one formula from $H(m)$, and then compute its shortest proof. The results so far show that $\chi(m) = \mathrm{Iso}(\mathrm{SMU}(n,m))$ is a good candidate. We will compute the sets $\mathrm{Iso}(\mathrm{SMU}(n,m))$ for $n = \lceil \log_2(m) \rceil, \dots, m-1$, 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 the set $\mathrm{Iso}(\mathrm{RSMU}(n,m))$ 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 $\mathrm{RSMU}(n,m)$. We can limit ourselves to 2-graphs $G = (V_1 \uplus V_2, E)$ where $|V_1| = 2n$ and $|V_2| = m$, 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 (McKay & Piperno, 2014) 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 $\mathrm{SSMU}(n,m)$ 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 by Lemma 9. Hence, in those cases $\mathrm{RSMU}(n,m) = \mathrm{SMU}(n,m)$, and we already have $\mathrm{Iso}(\mathrm{SMU}(n,m))$. From these starting points, we repeatedly apply Lemma 7 to every formula in $\mathrm{Iso}(\mathrm{SMU}(n,m))$ to obtain $\mathrm{Iso}(\mathrm{SSMU}(n+1,m+1))$. Together with $\mathrm{Iso}(\mathrm{RSMU}(n+1,m+1))$ we then obtain $\mathrm{Iso}(\mathrm{SMU}(n+1,m+1))$.

The rationale for splitting the computation of $\mathrm{SMU}(n,m)$ into two pieces is the following. Enumerating non-isomorphic clause-literal graphs by Nauty for given parameters n and m is one of the bottlenecks in the process. We often need to enumerate a significantly larger set than $\mathrm{SMU}(n,m)$. 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 Lemma 7 afterwards inductively to the rather small set $\mathrm{SMU}(n,m)$ is computationally affordable (as long as the set $\mathrm{SMU}(n,m)$ remains reasonably small, which it does in our cases).

Since we are interested only in SMU formulas, but it is difficult to generate only SMU formulas directly, we will have to resort to generating a superset and filtering for saturated minimal unsatisfiability. We will now present two approaches to accomplish this.

The first, which we used early on, uses multiple calls to a SAT solver on different formulas—and our goal is to show how these only loosely related SAT calls can be recast as solving the same formula with different assumptions.

The second approach is based on brute force, and we employed it after realizing that the overhead of constructing even a single SAT instance is too prohibitive (for details see Section 6). Here, we will show how a single pass through the set of assignments of a formula can determine saturated minimal unsatisfiability, if we keep track of the right information.

3.2 Testing SMU Using a SAT Solver

Testing saturated minimal unsatisfiability of a formula F by definition requires testing satisfiability of a number of formulas constructed by making small changes to F. Re-initializing a SAT solver with different formulas for different tests is expensive, and therefore it is desirable to pack as many SAT calls closely together either by adding clauses incrementally or by using assumptions. While testing minimal unsatisfiability incrementally 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 tested formula is unsatisfiable).

Lemma 10. *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 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. $\square$

Lemma 10 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 at least one of these formulas is unsatisfiable, the clause C can be extended with x preserving unsatisfiability, meaning that F is not saturated; otherwise F is saturated. This can be implemented in an assumption-based fashion with a single formula by augmenting every clause of F with a fresh selector variable to obtain $F^* = \{\, C \cup \{\overline{s_C}\} \mid C \in F \,\}$, and querying satisfiability of the formulas $F^*[\tau_{C,x}]$ for $C \in F, x \notin C$, where $\tau_{C,x}(s_C) = 0, \tau_{C,x}(s_D) = 1$ for $D \neq C$, and $\tau_{C,x}(x) = 1$, using assumptions.

3.3 Testing SMU by Brute Force

Motivated by the observation that the process described in Subsection 3.2, and in fact just constructing F^* and loading it into the SAT solver, still takes up most of the time, we devised a brute-force algorithm that is much more efficient for a large number of tiny formulas. Algorithm 1 runs through all assignments τ of a formula F explicitly and calculates the set of clauses falsified by τ in order to check that F is unsatisfiable in the first place. Additionally, for every clause C, the algorithm keeps track of the set of *critical* assignments—the ones that falsify *only* C. Given all this information, saturated minimal unsatisfiability can be decided easily: minimal unsatisfiability boils down to the existence of a C-critical assignment for every clause; asking whether some clause C can be extended with a literal l while maintaining unsatisfiability is the same as making sure that adding the literal l will not turn any C-critical assignment into a satisfying one (a non-critical assignment remains falsifying). Correctness of Algorithm 1 follows from Lemma 11.

Lemma 11. *Let F be an unsatisfiable formula. For a clause $C \in F$, let C^{-1} denote the set of C-critical assignments. Then*

1. *F is minimally unsatisfiable if, and only if, for every clause $C \in F$, $C^{-1} \neq \emptyset$;*
2. *F is saturated minimally unsatisfiable if, and only if, for every clause $C \in F$ and every variable $x \notin var(C)$, there are C-critical assignments $\tau_0, \tau_1 \in C^{-1}$ such that $\tau_0(x) \neq \tau_1(x)$;*

Proof. 1. follows by definition. For 2. assume F is not saturated minimally unsatisfiable. This is iff there is a clause C and a variable $x \notin var(C)$ such that wlog $G := F \cup (C \cup \{x\}) \setminus C$ is unsatisfiable. Consider a C-critical assignment τ (in F). Since τ cannot satisfy $C \cup \{x\}$ (then it would satisfy G), we must have $\tau(x) = 0$. Hence, this is further equivalent to the fact that all critical assignments have the same value at x. □

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 $var(F) = \{x_1, \ldots, x_n\}$, 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 `SHORT(`F, s`)`.

It is easy to see that `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 `UNSAT(`F`)` = `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 `SHORT(`F, s`)` is an open problem—our intuition, based on our inability to construct a deterministic single-exponential-time algorithm for `SHORT(`F, s`)`, is that it might be NEXPTIME-complete.

The basic idea of our encoding is to have variables $\text{pos}[i, v]$ and $\text{neg}[i, v]$ that determine whether v and $\overline{v}$ occur in L_i, and variables $\text{arc}[i, j]$, 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 recently proposed by Marques-Silva and Mencía (2019, henceforth referred to as MSM), but made improvements 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

Algorithm 1 Deciding saturated minimal unsatisfiability in one pass through the set of assignments.

```
procedure IsSMU($F$)
    crit $= \emptyset$                    $\triangleright$ the set of clauses which have critical assignments
    for $C \in F$ do
        for $x \in var(F)$ do
            max_val$[C][x] = 0$           $\triangleright$ maximal value of $x$ in $C$-critical assignments
            min_val$[C][x] = 1$           $\triangleright$ minimal value of $x$ in $C$-critical assignments
        end for
    end for
    for $\tau : var(F) \to \{0,1\}$ do
        $\mathcal{Z}(\tau) = \{C \in F \mid \tau(C) = 0\}$
        if $\mathcal{Z}(\tau) = \emptyset$ then
            return False                  $\triangleright$ satisfiable
        end if
        if $\mathcal{Z}(\tau) = \{C\}$ then
            crit $\cup= \{C\}$
            max_val$[C]$ $\vee= \tau$
            min_val$[C]$ $\wedge= \tau$
        end if
    end for
    if crit $\neq F$ then
        return False                      $\triangleright$ not minimal
    end if
    for $C \in F$ do
        for $x \in var(F) \setminus var(C)$ do
            if max_val$[C][x] =$ min_val$[C][x]$ then
                return False              $\triangleright$ not saturated, can extend $C$ with a literal on $x$
            end if
        end for
    end for
    return True
end procedure
```

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 all axioms are used in the proof.

On the other hand, we extend the encoding with powerful symmetry breaking constraints. These constraints, 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 standalone theoretical contribution. Moreover, thanks to this additional symmetry breaking, we were able to compute shortest proofs of many formulas, for which MSM failed to produce an answer in our experiments. 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, similarly to Lemma 4. 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: Core variables used by the shortest-proof encoding. The symbol v is understood to range over $V := var(F)$, 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 ($n \leq m \leq s$).

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, as follows:

$$\begin{array}{rcl}
x \leq y & : & \overline{x} \vee y \\
x \geq y & : & x \vee \overline{y} \\
x \ = \ y_1 \wedge \cdots \wedge y_k & : & \bigwedge_{i=1}^{k} \left(\overline{x} \vee y_i\right) \ \wedge \ (x \vee \overline{y_1} \vee \cdots \vee \overline{y_k}) \\
x \ = \ y_1 \vee \cdots \vee y_k & : & \bigwedge_{i=1}^{k} \left(x \vee \overline{y_i}\right) \ \wedge \ (\overline{x} \vee y_1 \vee \cdots \vee y_k) \\
\bigwedge_{i=1}^{k} x_i \ \rightarrow \ \bigvee_{j=1}^{l} y_j & : & \overline{x_1} \vee \cdots \vee \overline{x_k} \vee y_1 \vee \cdots \vee y_l \\
\bigvee_{i=1}^{k} x_i \ \rightarrow \ \bigwedge_{j=1}^{l} y_j & : & \bigwedge_{i=1}^{k} \bigwedge_{j=1}^{l} (\overline{x_i} \vee y_j)
\end{array}$$

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 $\sum_{x \in X} x \,\triangle\, k$, $\triangle \in \{\leq, \geq, =\}$, which can be encoded using an arbitrary CNF cardinality-constraint encoding. We use the sequential counter (Sinz, 2005), which seemed to perform best in our tests, but our implementation allows to pass the cardinality encoding to be used as a parameter (see Section 6 for more details).

4.1 Definitions

Definition of isax$[i, j]$: the i-th clause is the j-the axiom if it contains precisely its literals.

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

The i-th clause is an axiom (isax$[i]$) if it is some axiom according to the constraint above.

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

The "carry" variables help us simplify reasoning around how literals are carried between clauses; {pos|neg}carry$[i, j, v]$ captures the conjunction of the facts that a v-literal appears in the i-th clause and that the i-th clause is a premise of the j-th clause.

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

Through carries, we capture the union of the two premises of a resolution step.

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

The pivot variable is defined as a variable that appears in both polarities in the union of premises. In a later constraint, we will enforce the uniqness of a pivot.

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

4.2 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: $\bigwedge_{1\le i<j\le s} \mathrm{isax}[j] \to \overline{\mathrm{arc}[i,j]}$.

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.

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

No new literals are introduced into resolvents.

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

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

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

4.3 Redundant Constraints

If we search for the proof by iteratively incrementing the bound s, we know that every clause must be used:

$$\bigwedge_{1\le i<s} \bigvee_{i<j\le s} \mathrm{arc}[i,j].$$

In that case, we know no proof shorter than s exists, and so every clause but the last is non-empty:

$$\bigwedge_{1\le i<s} \left(\bigvee_{v\in V} \mathrm{pos}[i,v] \vee \mathrm{neg}[i,v] \right).$$

Axioms do not have pivots: $\bigwedge_{\substack{1<i\le s\\ v\in V}} \mathrm{isax}[i] \to \overline{\mathrm{piv}[i,v]}$. We require that the axioms are placed at the beginning of the proof $\bigwedge_{1\le i<s} \mathrm{isax}[i+1] \to \mathrm{isax}[i]$, 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: $\bigwedge_{i=1}^{m} \mathrm{ax}[i,i] \bigwedge_{i=m+1}^{s} \overline{\mathrm{isax}[i]}$.

4.4 Counting the In- and Out-Degrees

Consider the proof DAG $G(P)$ defined in Section 2. 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. The goal of this constraint is to discover early that a partially constructed proof DAG cannot be extended to a full DAG, and reject it. This feature is based on the simple identity $\sum_{v \in V} d_{\text{out}}(v) = \sum_{L \in V} d_{\text{in}}(v)$, which holds in every directed graph $G = (V, E)$, and which we already used in the proof of Lemma 4 in order to establish a lower bound on the length of a proof of a minimally unsatisfiable formula.

The core idea is that we know the in-degrees, and so the total number of arcs there must be in a proof DAG, and we also know that every clause except the last has an outgoing arc. Suppose A is a partial assignment of arcs in a proof of length s, i.e., A is a sub-DAG of a proof DAG with s vertices, and suppose A has $k(A)$ arcs, and $t(A)$ vertices with out-degree 0. We know that in any proof DAG that extends A, each of the $t(A)$ clauses except the last will have at least one outgoing arc; so any extension of A must have at least $k(A)+t(A)-1$ arcs. But at the same time, any extension of A must have $2(s-m)$ arcs (for the m axioms of in-degree 0 and $s-m$ resolvents of in-degree 2); we reach a contradiction if $k(A)+t(A)-1 > 2(s-m)$. Our goal now, is to produce a constraint that will perform this reasoning and enforce for every partial arc assignment A that $k(A)+t(A)-1 \leq 2(s-m)$. The main challenge is to capture the value of $t(A)$; the rest can be achieved via a cardinality constraint.

To capture the value of $t(A)$, 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* (this includes symmetry breaking: the single non-extra arc is the one with the lowest index). Let $e(A)$ be the number of extra arcs in the partial proof DAG A. It can be seen that $k(A)+t(A)-1 = e(A)+s-1$: starting from the $k(A)$ arcs in A, add $t(A)-1$ non-extra arcs going out of the clauses with out-degree 0, for a total of $s-1$ non-extra arcs and $e(A)$ extra arcs (no new extra arcs were added). Hence, the inequality $k(A)+t(A)-1 \leq 2(s-m)$ can be rewritten as $e(A) \leq s-2m+1$. If we can keep track of the number of extra arcs, we can simply encode this as a cardinality constraint. 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.

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

This cardinality constraint ensures that a partial proof does not have too many extra arcs, and there is enough "arc budget" left to add at least one arc to every clause that does not yet have one. Since the cardinality constraint alone is unsatisfiable if the right-hand side is negative, we obtain Lemma 4 as a special case.

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 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 (2020) who introduced it in a different context; Fichte, Hecher, and Szeider (2020) 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 $\mathrm{geq}[i, j, l]$, which say that, when restricted to literals up to position l, the clause L_j is a superset of L_i. Here we are using the trick to encode a lexicographic ordering where we relax the condition that the previous bits are equal to the condition that they are greater than or equal, exploiting the fact that the opposite inequality has been enforced on the previous bits already (Lemma 10.7.1, Sakallah, 2009).

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

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

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

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).

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

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

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

The last set of unit clauses ensures that a superclause of a previously derived clause should never be derived.

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

Finally, Theorem 1 summarizes the properties of our encoding.

Theorem 1. *Let F be a propositional formula on n variables and m clauses and let $\mathtt{short}_s(F)$ be the formula defined in Sections 4 and 5. Then the following statements hold:*

1. *the size of $\mathtt{short}_s(F)$ is $O(\max(n,m,s)^3)$ variables and clauses; (s can be exponential in the input length). For MU formulas we have $n < m \le s$, i.e., size $O(s^3)$;*

2. *$\mathtt{short}_s(F)$ 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 $\mathtt{short}_s(F)$ 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 $\mathtt{short}_s(F)$ is unsatisfiable. As soon as $\mathtt{short}_s(F)$ becomes satisfiable, s is the length of a shortest resolution refutation of F, and the refutation itself can be extracted from a model of $\mathtt{short}_s(F)$. An improvement is possible for MU formulas, by starting not at $s = 1$, but $s = 2m - 1$, as described in Section 4.

6. Experimental Setup

In this section, we describe how we performed our computations; Section 7 afterwards is devoted to a discussion of the results we obtained. We will refer to formulas and graphs interchangeably throughout these two sections, 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 from the graph automorphism package Nauty (McKay & Piperno, 2014), provided to us by Brendan McKay, 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` and a custom filtering routine that catches some satisfiable and non-minimal formulas early. The meaning of our parameter settings is: 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 at least 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 neighborhoods of vertices from V_2 are one subset of the other (`-A`). This gives us a set $S(n,m)$ of graphs that contains $\mathrm{Iso}(\mathrm{RSMU}(n,m))$, 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.

Our filtering subroutine takes a partially constructed graph and tries to determine that it cannot be extended to an unsatisfiable or a minimally unsatisfiable formula. A partial graph corresponds to a formula with some clauses already fixed, and others left yet undetermined. We count the number of satisfying assignments of a partial graph, and using the fact that Nauty adds clauses in increasing order of size, we can trivially lower-bound the number of satisfying assignments of the final formula.

Lemma 12. *If F has at least μ satisfying assignments, and each clause in G has width at least ω, then $F \cup G$ has at least $\mu - 2^{n-\omega}|G|$ satisfying assignments, where $n = |var(F \cup G)|$.*

Because the clauses come in increasing order, we can put $\omega = \max_{C \in F} |C|$, and when we are generating n-variable, m-clause formulas, we know both n and also that $|G| = m - |F|$. Thus, if the number $\mu - 2^{n-\omega}(m - |F|)$ is positive, we can immediately discard the current branch of the search (all descendants of F). Similarly, if a partially constructed formula is already unsatisfiable, we know that the final formula will not be minimally unsatisfiable, and we can discard as well. This pruning is critical as without it the vast majority of graphs generated would be satisfiable, causing most of the work to go to waste. As an example, without our filtering routine, for $n = 5$ and $m = 9$, out of the total more than 9 billion generated graphs, fewer than 0.05% were unsatisfiable.

The pruning described in Lemma 12 as well as SMU filtering both require solving SAT, or even counting models. In early stages of our work, we used a SAT solver for that purpose (CryptoMiniSAT via its `C` API; Soos, Nohl, & Castelluccia, 2009), resorting to a lower bound based on clause sizes instead of counting models exactly (any formula F trivially has at least $2^{|var(F)|}\left(1 - \sum_{C \in F} 2^{-|C|}\right)$ models), and using Lemma 10 for SMU testing. However, we have since found out that using brute force (going through all assignments explicitly) instead of a SAT solver is much more efficient in these cases, and also allows exact model counting at no extra cost. Nauty encodes sets via their characteristic vectors, and the vectors in turn are represented by the bits of an `unsigned int`. In our case, we work with small sets (clauses on < 32 literals), which means both a full assignment and a clause can fit into one CPU word. Operations such as determining whether an assignment satisfies a given clause conveniently reduce to a single CPU instruction, and the entire process is extremely cache friendly. This stands in contrast with the cumbersomeness of having to construct a SAT-solver-friendly representation of the graph output by Nauty, where a huge chunk of the time is wasted. Checking satisfiability and counting satisfying assignments is trivial by brute force, for testing saturated minimal unsatisfiability we use Algorithm 1. For an example of the kind of speedup obtained, consider generating SMU$(4, 8)$, which takes 150 seconds with the SAT-solver-based method, while the brute-force method finishes in under a second. We do not report more detailed comparisons, because as we gradually switched to the improved version for larger values of n and m, we left the slower versions behind.

Once we have generated Iso(RSMU(n, m)), which is equal to Iso(SMU(n, m)) for values of n, m where MU$(n - 1, m - 1)$ is empty, we use Lemma 7 to compute SMU$(n + 1, m + 1)$. Whenever we have computed 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 (Ignatiev, Morgado, & Marques-Silva, 2018). Our tool allows the user to choose their favorite SAT solver—among the ones included in PySAT, or indeed any other that they have installed. Similarly, the user can specify which CNF cardinality encoding (among the ones implemented in PySAT) to use within our encoding. Since these parameters can be expected to have a significant impact on performance, before we started our experiments, we performed a grid search through all pairs of SAT solver (the ones in PySAT + Kissat and CryptoMiniSAT 5.8) and cardinality encoding, using a randomly sampled set of RSMU(8), RSMU(9), SSMU(8), and SSMU(9) formulas, 32 of each kind. From this training phase we concluded that the configuration with CaDiCaL [4] and sequential counter performed best, and we used it throughout the rest of the experiment.

We note that any training process is bound to be imperfect, because the whole workflow suffers from aggressively heavy-tailed behavior. It is easy to parallelize the computation of shortest proofs across different formulas, and it is even possible to parallelize the individual $\texttt{short}_s(F)$ queries for one given formula. However, as we illustrate in the next section in Figure 3, this only helps so much, because the bulk of the computation rests in the optimality queries for a tiny set of the hardest formulas. In order to scale beyond our current results, a cube-and-conquer (Heule, Kullmann, & Biere, 2018) approach to solve the hardest queries may be called for. Apart from fully automated cubing algorithms, another promising option is to exploit the high-level structure of our problem—search for a (proof) DAG. A partitioning of the set of candidate DAGs into subsets of roughly equal size which can be expressed succinctly as a set of cubes could in theory bring close to linear speedups. Coming up with such a partitioning appears quite non-trivial; but it also looks like something that might be within the reach of followup work.

Our encoding (Peitl & Szeider, 2021b) and our catalog of SMU formulas (Peitl & Szeider, 2021a), including the data from our computation, are publicly available.

7. Results

In this section we present and explore in detail the results of our computation. We begin with the analysis of the hardness sequence and the finer structure of hardness in SMU formulas.

Table 2 lists the length of the longest shortest proof required by an SMU(n, m) formula, and, by taking the maximum in each column, also values of h_m. In particular, we obtain

4. https://fmv.jku.at/cadical

$n\backslash m$	1	2	3	4	5	6	7	8	9	10
0	**1** (1)	-	-	-	-	-	-	-	-	-
1	-	**3** (1)	-	-	-	-	-	-	-	-
2	-	-	**5** (1)	**7** (1)	-	-	-	-	-	-
3	-	-	-	**7** (2)	**10** (1)	11 (3)	13 (1)	15 (1)	-	-
4	-	-	-	-	9 (3)	**13** (1)	15 (1)	**19** (1)	20 (1)	21 (5)
5	-	-	-	-	-	11 (6)	**16** (1)	18 (3)	**22** (1)	25 (1)
6	-	-	-	-	-	-	13 (11)	**19** (1)	**22** (3)	**26** (3)
7	-	-	-	-	-	-	-	15 (23)	**22** (1)	25 (24)
8	-	-	-	-	-	-	-	-	17 (46)	25 (1)
9	-	-	-	-	-	-	-	-	-	19 (98)

Table 2: Values of $h(\mathrm{SMU}(n,m))$,i.e., the lengths of the longest shortest proof required by a saturated minimally unsatisfiable formula with n variables and m clauses, and in parentheses the number of formulas in $\mathrm{SMU}(n,m)$ that require resolution proofs of length $h(\mathrm{SMU}(n,m))$. For all $3 \leq n \leq 9$ and $n+2 \leq m \leq 10$, we found that all hardest $\mathrm{SMU}(n,m)$ formulas are regular, except for $\mathrm{SMU}(7,10)$, which also contains 19 singular formulas. All formula counts are modulo isomorphisms. By Proposition 2, $h(H(m-1,m)) = 2m-1$, and so no computation is necessary. By Lemma 9, there are no minimally unsatisfiable formulas in the areas marked by -.

the first ten resolution hardness numbers:

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

Since all deficiency-1 formulas have the same hardness, the corresponding numbers in parentheses in Table 2 give the number of $\mathrm{SMU}(m-1,m)$ formulas modulo isomorphisms. This is Sequence A001190 of the Online Encyclopedia of Integer Sequences (http://oeis.org/A001190), and indeed the correspondence of deficiency-1 saturated minimally unsatisfiable formulas to rooted binary trees is known (Kullmann, 2000b). However, deficiency-1, and indeed singular SMUs in general, are not too interesting from the hardness standpoint; for $m \geq 5$, all formulas with maximum hardness h_m are regular. The numbers of RSMU formulas with a given number of variables and clauses are shown in Table 3.

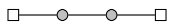
It is known that every $\mathrm{MU}(n,m)$ formula has a proof of length at most $2^{m-n-1}n + m$ (Kleine Büning & Kullmann, 2009, Section 11.3), and, along with the existence of formulas which require superpolynomially long proofs (Haken, 1985), this implies that maximum hardness cannot forever be attained 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. Below, we show the clause-literal graphs of all these and other hardest formulas for each number of clauses $m \leq 10$. The label $h(F_{n,m,i}) = s$ indicates the formula belongs to $\mathrm{SMU}(n,m)$ and requires a resolution proof of length s; i is an identifier to distinguish formulas with the same number of variables and clauses.

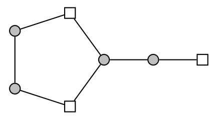
$n\backslash m$	1	2	3	4	5	6	7	8	9	10
0	1									
1		0								
2			0	1						
3				0	1	3	1	1		
4					0	1	14	59	87	96
5						0	1	45	755	5664
6							0	1	92	5928
7								0	1	154
8									0	1
9										0

Table 3: Iso(RSMU(n, m)) counts, i.e., the numbers of non-isomorphic regular saturated minimally unsatisfiable formulas with n variables and m clauses. There are no minimally unsatisfiable formulas in the blank areas by Lemma 9.

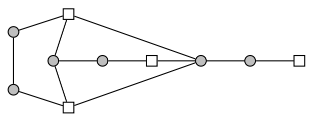
$m = 1, 2, 3$**:** up to isomorphism, there exists exactly one hardest formula with 1, 2 or 3 clauses, requiring a resolution proof of length 1,3, and 5, respectively.

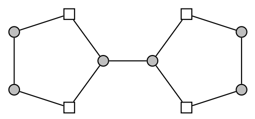
$h(F_{0,1,1}) = 1$

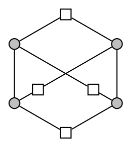

$h(F_{1,2,1}) = 3$


$h(F_{2,3,1}) = 5$

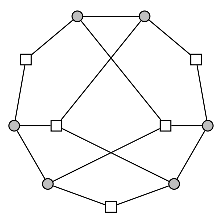
$m = 4$**:** up to isomorphism, there exist exactly 3 hardest formulas with 4 clauses, which are saturated minimally unsatisfiable and require a resolution proof of length 7. Two of them have deficiency 1 and are singular, one has deficiency 2 and is regular (and so it is $\mathcal{F}_4^2$).

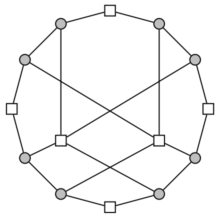

$h(F_{3,4,1}) = 7$

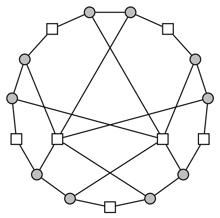

$h(F_{3,4,2}) = 7$


$h(F_{2,4,1}) = 7$

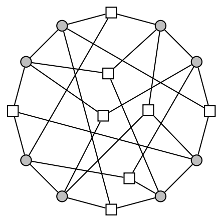
$m = 5, 6, 7$**:** up to isomorphism, there exists exactly one hardest formula with 5, 6 or 7 clauses which is saturated minimally unsatisfiable. It is always the respective unique regular deficiency 2 formula $\mathcal{F}_m^2$, and it requires a resolution proof of length 10, 13, and 16, respectively. The clause-literal graphs of these, and in fact all $\mathcal{F}_m^2$ formulas are planar, but here we chose to draw them with crossings in order to highlight the cycle which contains all variable edges. Interestingly, other regular formulas with hardness h_m do not have planar clause-literal graphs (as verified with Nauty's `planarg`).

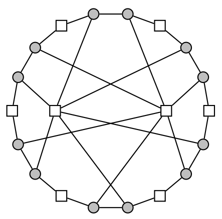

$h(F_{3,5,1}) = 10$


$h(F_{4,6,1}) = 13$

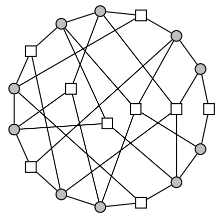

$h(F_{5,7,1}) = 16$

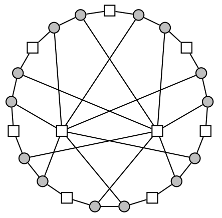
$m = 8$**:** up to isomorphism, there exist exactly two hardest formulas with 8 clauses which are saturated minimally unsatisfiable. One of them has deficiency 2 and is $\mathcal{F}_8^2$, the other has deficiency 4. Both are regular and require a resolution proof of length 19. $F_{4,8,1}$ stands out among the hardest formulas with an unusually large automorphism group—48 elements.

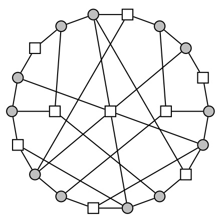

$h(F_{4,8,52}) = 19$

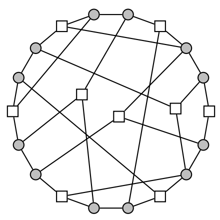

$h(F_{6,8,1}) = 19$

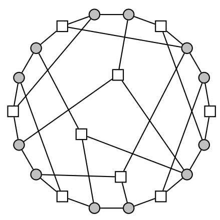
$m = 9$**:** up to isomorphism, there exist exactly five hardest formulas with 9 clauses which are saturated minimally unsatisfiable. One of them has deficiency 2 and is $\mathcal{F}_9^2$, three have deficiency 3, and one has deficiency 4. All five formulas are regular and require a resolution proof of length 22. $F_{6,9,1}$ and $F_{6,9,2}$ stand out among the hardest formulas, being the only ones with an automoprhism group of size 1—without non-trivial automorphisms.


$h(F_{5,9,1}) = 22$


$h(F_{7,9,1}) = 22$


$h(F_{6,9,1}) = 22$


$h(F_{6,9,2}) = 22$


$h(F_{6,9,3}) = 22$

$m = 10$**:** up to isomorphism, there exist exactly three hardest formulas with 10 clauses which are saturated minimally unsatisfiable. All have deficiency 4 and require a resolution proof of length 26. We point out that these three formulas are drawn symmetrically around

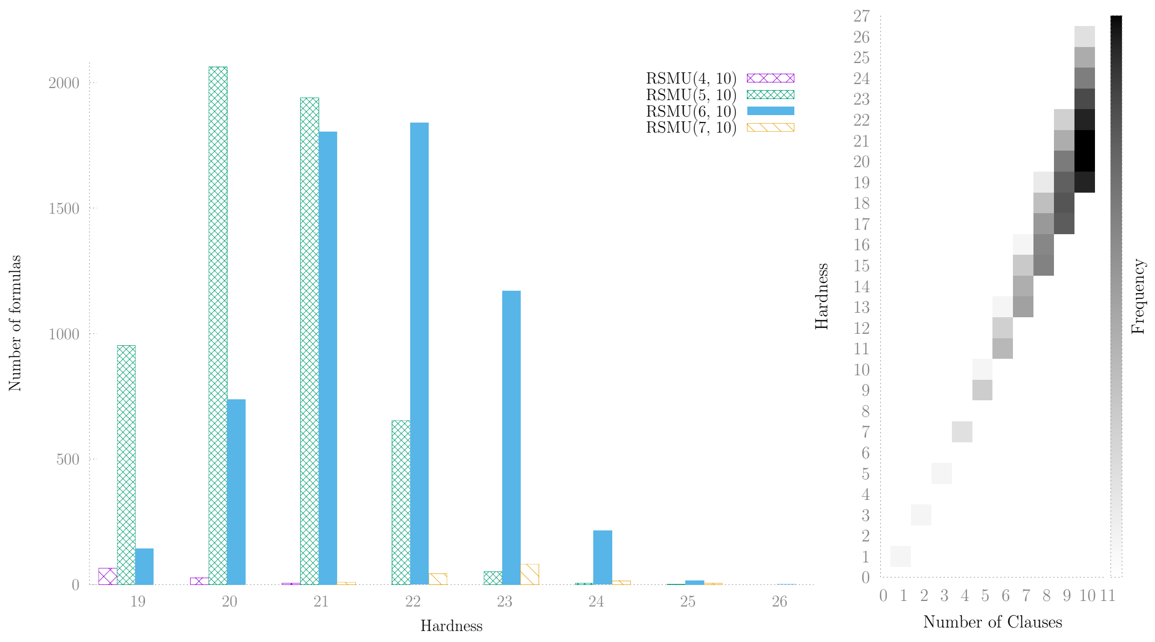
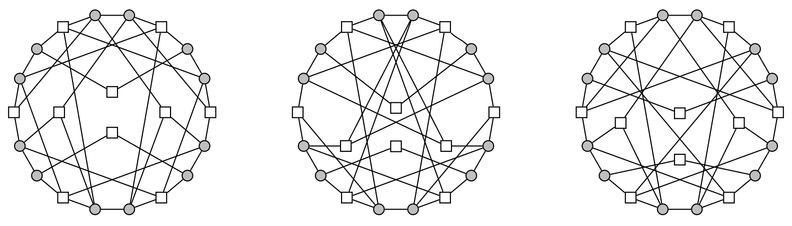


Figure 1: Left: hardness distribution among regular saturated minimally unsatisfiable formulas with 10 clauses and 4-7 variables. Right: hardness distribution over all saturated minimally unsatisfiable formulas on ≤ 10 clauses.

the vertical center line. Apart from $\mathcal{F}_m^2$, these are the only hardest formulas that can be drawn in such a way—symmetrically and with a cycle containing all variable edges drawn as a geometric circle. We calculated these drawings (and the non-existence for other graphs) with MiniZinc (Nethercote et al., 2007; Stuckey et al., 2014).



$h(F_{6,10,1}) = 26$ $h(F_{6,10,2}) = 26$ $h(F_{6,10,3}) = 26$

The resolution hardness numbers and the corresponding hardest formulas are at the extreme end of the hardness *distribution*, i.e., the detailed picture of how many formulas with a given number of clauses and hardness there are. Figure 1 shows the distributions of hardness: within RSMU(10), broken down by the number of variables (RSMU(8, 10) is omitted, as it only contains one formula with hardness 25 by Lemma 3); and overall in each set SMU(m) for $m \leq 10$.

It is intuitively clear even without extensive analysis that as formulas grow in the number of clauses, they get harder, at least in the worst case. Is, however, the same true when we fix the number of clauses, and vary formula *length*—the number of literal occurrences in F

(i.e., $\sum_{C \in F} |C|$)? Figure 2 answers this question negatively for RSMU(10)—there appears to be no correlation between length and hardness.

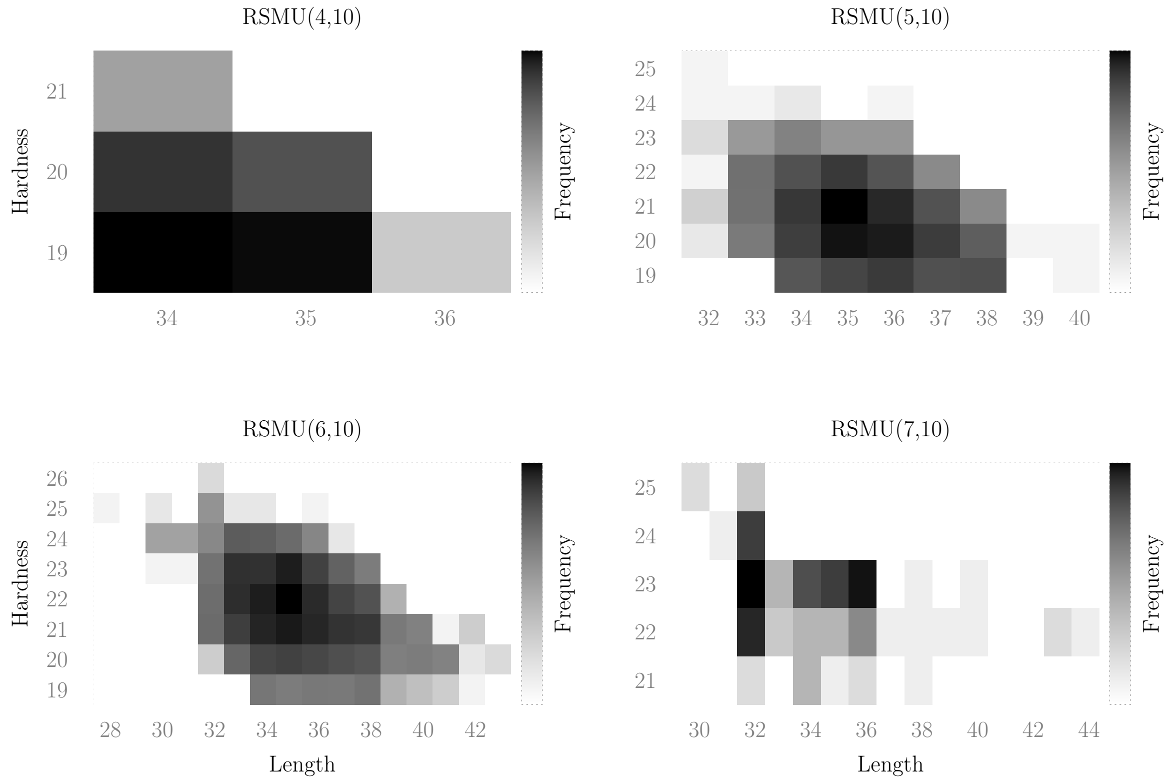


Figure 2: Regular saturated minimally unsatisfiable formulas on 10 clauses broken down by formula length and hardness. The frequency says how many formulas of the given kind there are. The rather trivial case of deficiency two is, again, omitted.

One could naturally ask why we stopped the computation at $m = 10$, and whether we could not continue to higher values. Figure 3 shows the running time needed to compute shortest proofs as a function of their length, on a 10-core 2.40GHz Intel Xeon E5-2640 v4. Experimentally, it appears that with each increment in formula hardness, computing a shortest proof becomes roughly 5 times harder. Moreover, as illustrated on the three hardest formulas, the bulk of the time is spent on the last query $\texttt{short}_{h(F)-1}(F)$. Based on the arguably likely assumption that $h_{11} \geq h_{10} + 3$, and the fitted curve (computed using SciPy and NumPy, Virtanen et al., 2020; Harris et al., 2020), we can estimate it will take over a year of CPU time to compute a shortest proof of one of the next hardest formulas. This work will be difficult to parallelize, as most of it is concentrated in solving the last query. Clever parallelization, for instance using cube-and-conquer, thus appears necessary to tackle h_{11}.

Generating formulas with 11 clauses is also a significant challenge. Even though Nauty provides built-in parallelization facilities, the number of graphs that need to be enumerated still grows very fast and algorithmic improvements will probably be necessary in order to

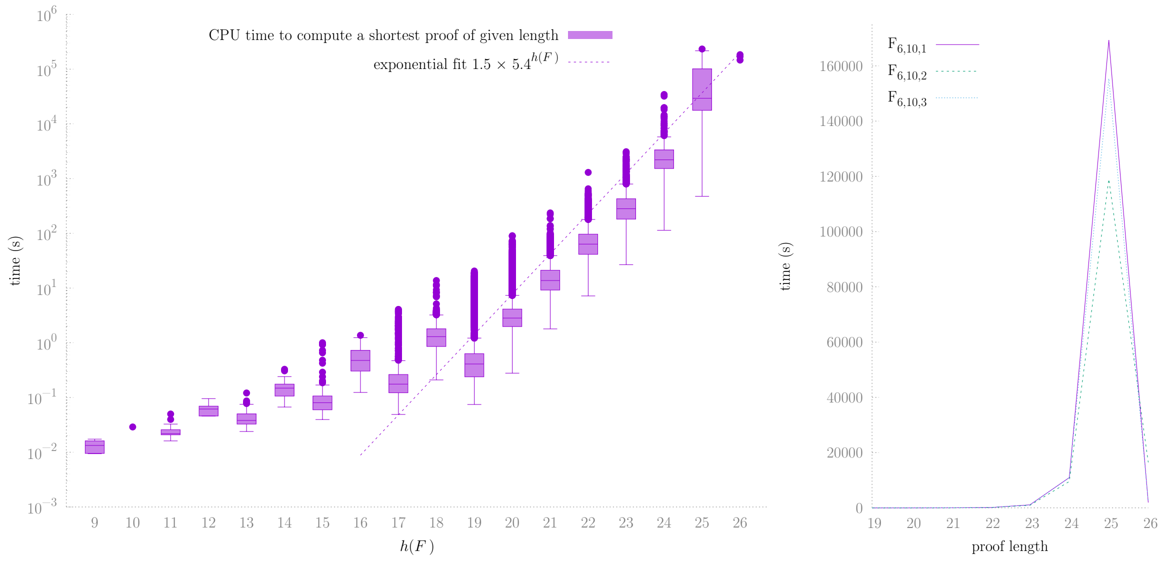


Figure 3: Left: time to compute a shortest proof depending on its length. Boxes show quartiles, whiskers extend to 1.5 × the interquartile range. Right: for the hardest 10-clause formulas, the distribution of solving time among the $\texttt{short}_s(F)$ queries. Over 80% of the work is spent on the optimality query; in two cases over 90%.

obtain SMU(11). One promising example is to detect early on partial graphs that cannot be extended to a saturated minimally unsatisfiable formula, similarly to how we check for graphs that cannot be extended to an unsatisfiable formula. This is left for future work.

8. 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.

Acknowledgments

This research was supported by the FWF (projects P32441 and J-4361) and by the WWTF (project ICT19-065). Special thanks to Brendan McKay for modifying Nauty's `genbg` for our purpose. A preliminary version of this paper appeared at CP (Peitl & Szeider, 2020).

References

Aharoni, R., & Linial, N. (1986). Minimal non-two-colorable hypergraphs and minimal unsatisfiable formulas. *J. Combin. Theory Ser. A*, *43*, 196–204.

Atserias, A., & Bonet, M. L. (2004). On the automatizability of resolution and related propositional proof systems. *Inf. Comput.*, *189*(2), 182–201.

Codish, M., Miller, A., Prosser, P., & Stuckey, P. J. (2019). Constraints for symmetry breaking in graph representation. *Constraints An Int. J.*, *24*(1), 1–24.

Davis, M., & Putnam, H. (1960). A computing procedure for quantification theory. *J. of the ACM*, *7*(3), 201–215.

Davydov, G., Davydova, I., & Kleine Büning, H. (1998). An efficient algorithm for the minimal unsatisfiability problem for a subclass of CNF. *Ann. Math. Artif. Intell.*, *23*, 229–245.

Diestel, R. (2012). *Graph Theory, 4th Edition*, Vol. 173 of *Graduate texts in mathematics*. Springer.

Fichte, J. K., Hecher, M., & Szeider, S. (2020). 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. this volume.

Haken, A. (1985). The intractability of resolution. *Theoretical Computer Science*, *39*, 297–308.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., Fernández del Río, J., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., & Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*, 357–362.

Heule, M. J. H. (2018). 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.

Heule, M. J. H., Kullmann, O., & Biere, A. (2018). Cube-and-conquer for satisfiability. In Hamadi, Y., & Sais, L. (Eds.), *Handbook of Parallel Constraint Reasoning*, pp. 31–59. Springer International Publishing, Cham.

Heule, M. J. H., Kullmann, O., & Marek, V. W. (2016). 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*, Vol. 9710 of *Lecture Notes in Computer Science*, pp. 228–245. Springer Verlag.

Ignatiev, A., Morgado, A., & Marques-Silva, J. (2018). PySAT: A Python toolkit for prototyping with SAT oracles. In *SAT*, pp. 428–437.

Kleine Büning, H. (2000). On subclasses of minimal unsatisfiable formulas. *Discr. Appl. Math.*, *107*(1–3), 83–98.

Kleine Büning, H., & Kullmann, O. (2009). Minimal unsatisfiability and autarkies. In Biere, A., Heule, M. J. H., van Maaren, H., & Walsh, T. (Eds.), *Handbook of Satisfiability*, Vol. 185 of *Frontiers in Artificial Intelligence and Applications*, chap. 11, pp. 339–401. IOS Press.

Kullmann, O. (2000a). An application of matroid theory to the SAT problem. Tech. rep. TR00–018, *Electronic Colloquium on Computational Complexity* (ECCC).

Kullmann, O. (2000b). An application of matroid theory to the SAT problem. In *Fifteenth Annual IEEE Conference on Computational Complexity*, pp. 116–124. See also TR00-018, Electronic Colloquium on Computational Complexity (ECCC), March 2000.

Kullmann, O. (2000c). Investigations on autark assignments. *Discr. Appl. Math.*, *107*(1-3), 99–137.

Kullmann, O., & Zhao, X. (2013). On Davis-Putnam reductions for minimally unsatisfiable clause-sets. *Theoretical Computer Science*, *492*, 70–87.

McKay, B. D., & Piperno, A. (2014). Practical graph isomorphism, {II}. *Journal of Symbolic Computation*, *60*(0), 94 – 112.

Mencía, C., & Marques-Silva, J. (2019). 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*, Vol. 11805 of *Lecture Notes in Computer Science*, pp. 539–551. Springer.

Michel, P. (2019). The busy beaver competition: a historical survey. *arXiv preprint*, *math.LO*(0906.3749).

Nethercote, N., Stuckey, P. J., Becket, R., Brand, S., Duck, G. J., & Tack, G. (2007). Minizinc: Towards a standard cp modelling language. In Bessière, C. (Ed.), *Principles and Practice of Constraint Programming – CP 2007*, pp. 529–543, Berlin, Heidelberg. Springer Berlin Heidelberg.

Peitl, T., & Szeider, S. (2020). Finding the hardest formulas for resolution. In Simonis, H. (Ed.), *Principles and Practice of Constraint Programming - 26th International Conference, CP 2020, Louvain-la-Neuve, Belgium, September 7-11, 2020, Proceedings*, Vol. 12333 of *Lecture Notes in Computer Science*, pp. 514–530. Springer.

Peitl, T., & Szeider, S. (2021a). *Saturated Minimally Unsatisfiable Formulas on up to Ten Clauses.* Zenodo. https://doi.org/10.5281/zenodo.3951545.

Peitl, T., & Szeider, S. (2021b). *short.py: Encoding for the shortest resolution proof.* Zenodo. https://doi.org/10.5281/zenodo.3951549.

Pipatsrisawat, K., & Darwiche, A. (2009). 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*, Vol. 5732 of *Lecture Notes in Computer Science*, pp. 654–668. Springer Verlag.

Prestwich, S. D., & Lynce, I. (2006). Local search for unsatisfiability. In Biere, A., & Gomes, C. P. (Eds.), *Theory and Applications of Satisfiability Testing - SAT 2006, 9th International Conference, Seattle, WA, USA, August 12-15, 2006, Proceedings*, Vol. 4121 of *Lecture Notes in Computer Science*, pp. 283–296. Springer.

Pudlák, P. (2003). On reducibility and symmetry of disjoint NP pairs. *Theor. Comput. Sci.*, *295*, 323–339.

Rado, T. (1962). On non-computable functions. *Bell System Technical Journal*, *41*(3), 877–884.

Sakallah, K. A. (2009). Symmetry and satisfiability. In Biere, A., Heule, M., van Maaren, H., & Walsh, T. (Eds.), *Handbook of Satisfiability*, Vol. 185, pp. 289–338. IOS Press.

Schidler, A., & Szeider, S. (2020). 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.

Silva, J. P. M., & Sakallah, K. A. (1996). GRASP - a new search algorithm for satisfiability. In *International Conference on Computer-Aided Design (ICCAD '96), November 10-14, 1996, San Jose, CA, USA*, pp. 220–227. ACM and IEEE.

Sinz, C. (2005). 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*, Vol. 3709 of *Lecture Notes in Computer Science*, pp. 827–831. Springer Verlag.

Soos, M., Nohl, K., & Castelluccia, C. (2009). 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*, Vol. 5584 of *Lecture Notes in Computer Science*, pp. 244–257. Springer.

Stuckey, P., Feydy, T., Schutt, A., Tack, G., & Fischer, J. (2014). The minizinc challenge 2008-2013. *AI Magazine*, *35*, 55–60.

Szeider, S. (2004). Minimal unsatisfiable formulas with bounded clause-variable difference are fixed-parameter tractable. *J. of Computer and System Sciences*, *69*(4), 656–674.

Urquhart, A. (1995). The complexity of propositional proofs. *Bull. of Symbolic Logic*, *1*(4), 425–467.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., & SciPy 1.0 Contributors (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, *17*, 261–272.