Hard Examples for Common Variable Decision Heuristics

Marc Vinyals

November 13, 2019

Abstract

The CDCL algorithm for SAT is equivalent to the resolution proof system under a few assumptions, one of them being an optimal non-deterministic procedure for choosing the next variable to branch on. In practice this task is left to a variable decision heuristic, and since the so-called VSIDS decision heuristic is considered an integral part of CDCL, whether CDCL with a VSIDS-like heuristic is also equivalent to resolution remained a significant open question.

We give a negative answer by building a family of formulas that have resolution proofs of polynomial size but require exponential time to decide in CDCL with common heuristics such as VMTF, CHB, and certain implementations of VSIDS and LRB.

1 Introduction

Conflict-Driven Clause Learning (CDCL) is nowadays the top-performing algorithm to solve SAT in practice. When a CDCL solver finds a formula to be unsatisfiable, a proof of this fact can be read from its execution trace, and that proof can be formalized in the resolution proof system [BKS04]. It is not obvious whether all resolution proofs can be obtained using a CDCL solver; for instance, the preceding DPLL algorithm is restricted to producing tree-like resolution proofs, which are exponentially weaker than general resolution [Goe93, Urq95, BEGJ00].

Progress towards this question was made showing that tweaked versions of CDCL polynomially simulate resolution—in other words, the algorithm can produce proofs that are only polynomially worse than the optimal resolution proof—either allowing the solver to not resolve conflicts immediately [BKS04] or making an additional pre-processing step [HBPV08]. The question was largely settled by Pipatsrisawat and Darwiche [PD11], who showed that under a few assumptions CDCL polynomially simulates resolution.

Naturally there has been work towards understanding which assumptions are needed; to explain them we have to sketch how the CDCL algorithm works. Similarly to DPLL, the algorithm assigns values to variables, either by choice or forced by a clause, until either a solution or a conflict is found. Upon reaching a conflict DPLL backtracks and makes the last choice forced to its opposite. CDCL also backtracks, but analyses the conflict to learn a new clause that precludes the same conflict—and potentially many others—from occurring again. Since part of the state is kept in the learned clauses it makes sense to occasionally restart the search, and for memory management reasons to forget some of the clauses.

A first, mild assumption is that the solver restarts often enough. This is a reasonable assumption to make since solvers do restart frequently in practice, but it is still interesting to study what would be the case without restarts, if only to understand the power of restarts. The pool resolution subsystem [Van05] captures proofs produced by solvers that do not restart, and a few candidates for a separation between pool and general resolution were proposed but all turned out to have short pool resolution proofs [BBJ14, BK14]. Long story short, we still do not know if restarts are needed.

Another assumption is that the solver never forgets a clause. This is an unrealistic assumption to make since in practice solvers do throw out many of the clauses they learn. A much weaker

assumption is needed, in that solvers need to keep at least a few selected clauses in order to produce optimal proofs [EJL⁺16], but there is a large stretch between having to keep a few selected clauses and all clauses, and we do not know where in this line the truth lies.

The last assumption we consider is that when the solver chooses which literal to branch on, it makes optimal non-deterministic choices—that is, with external oracle advice. Of course this is not at all what happens in practice, in fact furnishing CDCL with a conflict-based decision heuristic is considered crucial for CDCL to work well if not an inextricable part of CDCL [MSLM09, KSM11, BF15], hence we cannot close the question whether CDCL polynomially simulates resolution unless we allow a heuristic that is closer to practice.

For instance, we can relax the non-deterministic assumption to making frequent enough random choices and obtain a weaker conclusion. In this case, and still using the first two assumptions, CDCL can simulate bounded-width resolution [AFT11]. At the same time, it is most likely that an assumption along these lines is needed, since a recent result shows that no deterministic algorithm can polynomially simulate resolution unless $P = NP$ [AM19].

1.1 Our Results

In this work we look into this last assumption, non-deterministic branching, and we prove unconditionally—without depending on whether $P = NP$—that if the CDCL algorithm uses conflict-based heuristics, then it cannot polynomially simulate resolution. This answers in the negative a question of Mikša and Nordström [MN14], who ask whether CDCL with VSIDS, 1UIP, and phase saving polynomially simulates resolution.

Theorem 1.1 (Informal). *There exists a family of formulas that have polynomial-size resolution proofs but CDCL with conflict-based decision heuristics runs in exponential time.*

We explain exactly what we mean with conflict-based in Sections 2.3 and 3, and for now we just say that the decision heuristics we look into all have in common that they select variables that have been recently involved in generating conflicts. These include VMTF, CHB, and certain implementations of VSIDS and LRB.

We do not need to make assumptions on the remaining parts of the algorithm, but it is worth mentioning a caveat that applies to which decision heuristics our result applies to. For technical reasons in some heuristics we need to impose that variable scores are stored either with arbitrary precision or in a stable data structure. We opt for the latter choice since, as we argue in Section 2.4, it is a plausible condition that does not make much difference in practice.

In addition to proving a theoretical result, we run experiments on the formulas we craft. The experiments confirm our theoretical findings, that is the formulas are very hard for SAT solvers that employ the heuristics we discuss, but trivial if we specify a static variable order. Even more, for certain parameter ranges it is more effective to make many decisions at random.

1.2 Related Work

We know of other branching heuristics that are not enough to simulate resolution, for instance in the context of encoding constraints into CNF a branching heuristic that only picks input variables (that is variables from the problem domain, as opposed to auxiliary variables introduced during translation to CNF) is exponentially weaker than full CDCL [JJ09]. However, our work is the first to expose the theoretical limits of conflict-based heuristics.

By no means we are the first to observe that there are formulas where VSIDS performs badly or that making random decisions can be helpful—in fact the appendix to the MiniSat solver description [ES04] discusses random decisions, and one may find plenty of concrete examples where a static variable order drastically outperforms VSIDS in the thorough evaluation of Elffers et al. [EGG⁺18].

1.3 Techniques

To prove our result we design a formula with the explicit purpose of tricking decision heuristics. Our formula consists essentially of an easy part that has polynomial-length proofs, a hard part that requires exponential-length proofs, and a set of pitfall gadgets that trick the solver into exploring the hard part before the easy part. Coming up with robust gadgets is a significant part of the work.

Proving an upper bound on resolution length is straightforward, but proving a lower bound on CDCL time becomes a delicate task. To do so we mix a low-level step-by-step simulation with a high-level proof-complexity argument. The simulation is an ad-hoc argument where we run the algorithm for a few steps, carefully controlling the state of the solver, until we reach a state where the solver is restricted to working on a reduced set of variables. Then we use a black-box proof-complexity result to show that any proof of the restricted formula must be of exponential size.

2 Preliminaries

2.1 Proof Complexity

A resolution refutation of a CNF formula F is a sequence of disjunctive clauses $C_1, \ldots, C_T$ such that every clause C_t either belongs to F or it can be obtained by applying the resolution rule

$$\frac{C \vee x \qquad D \vee \overline{x}}{C \vee D} \tag{1}$$

with $C \vee x = C_i$, $D \vee \overline{x} = C_j$, and $i, j < t$. The length of a refutation is T, the number of clauses in the refutation.

The restriction of a clause C by a truth value assignment ρ is $C{\restriction}_\rho = \bigvee_{l \in C} \rho(l)$. The restriction of a set of clauses is the result of restricting each individual clause and eliminating tautologies. If π is a resolution refutation of F, then $\pi{\restriction}_\rho$ is a valid resolution refutation of $F{\restriction}_\rho$.

The Tseitin formula of a graph $G(V, E)$ and a charge function $\chi : V \to \{0, 1\}$ is a CNF with one variable per edge and the set of constraints $Ц(G, \chi) = \bigwedge_{v \in V} Parity(v)$ where $Parity(v)$ is the CNF encoding $\bigoplus_{e \ni v} x_e = \chi(v)$. A Tseitin formula is unsatisfiable if and only if $\sum_{v \in V} \chi(v)$ is odd, which we assume from now on. Tseitin formulas are hard to prove within the resolution proof system.

Theorem 2.1 ([Urq87]). *There is a family of graphs on n vertices and constant degree such that $Ц(G, \chi)$ requires resolution proofs of length $\exp(\Omega(n))$.*

2.2 The CDCL Algorithm

We follow the description of CDCL of Elffers et al. [EJL$^+$16]. The input is a CNF formula F. The state is formed by the trail $\langle \rho \rangle$, which is a sequence of assignments, and the clause database $\mathcal{D}$, which is a set of clauses. Initially $\langle \rho \rangle = \emptyset$ and $\mathcal{D} = F$. The algorithm runs in a loop and the action to take at each step depends on $\mathcal{D}{\restriction}_\rho$. If the empty clause $\bot$ is in $\mathcal{D}{\restriction}_\rho$ then we say there is a conflict. If we can resolve it then we learn a clause and add it to $\mathcal{D}$, otherwise the formula is unsatisfiable. If there is a unit clause $x^b \in \mathcal{D}{\restriction}_\rho$ then we propagate its value by appending $x = b$ to $\langle \rho \rangle$. Otherwise we say that we are in a stable state. In this case we might forget a few clauses and use a reduced database $\mathcal{D}' \subseteq \mathcal{D}$, we might restart the search by resetting the trail to $\langle \rho \rangle = \emptyset$ but keeping the learned clauses, or we decide the value of some unassigned variable by appending $x = b$ to the trail, unless all variables are assigned in which case we found a satisfying assignment.

In order to fully determine the CDCL algorithm we need to specify its learn, propagate, forget, restart, and decide components. We assume about the learning component that any learned clause can be derived from the conflict clause and unit-propagating clauses [BKS04] but nothing more. The clauses used in such derivation are said to *participate* in a conflict, and so are all the variables contained in clauses that participate in a conflict.

Algorithm 1: CDCL

1 **while** *not solved* **do**
2   **if** *conflict* **then** learn()
3   **else if** *unit* **then** propagate()
4   **else**
5   maybe forget()
6   maybe restart()
7   decide()

Lemma 2.2 ([BKS04]). *If a formula F over n variables is found unsatisfiable by a CDCL solver in T conflicts, then there is a resolution refutation of F of length at most $n \cdot T$ using only conflict and unit-propagating clauses.*

We do not make any assumptions about which variable is propagated first when multiple choices are possible, nor about which clauses are forgotten, nor when to restart the search, and we discuss branching heuristics next.

2.3 Branching Heuristics

In the heuristics we consider the decision of which literal to branch on is actually taken by running two procedures, first a variable decision heuristic, or decision heuristic for short, and then a phase decision heuristic. We assume that the phase is chosen according to the phase-saving heuristic, that is with the same sign as the last time that the selected variable was last assigned. If a variable has never been assigned, we assume that it is first assigned to 0.

The variable decision heuristics we consider all assign a score q to each variable, which is updated as the search progresses, and at the time to make a decision they choose the variable with the largest score. Tie-breaking is often considered an implementation detail.

We consider the top performing decision heuristics evaluated by Biere and Fröhlich [BF15], and in addition two later heuristics that outperform VSIDS on certain benchmarks [LGPC16a, LGPC16b].

In VSIDS [MMZ$^+$01, ES04] the score of a variable is bumped each time it participates in a conflict and decays at every conflict. More formally, the score of variable x after conflict t is $q(x, t) = b(x, t) + \delta \cdot q(x, t - 1)$, where $0 < \delta < 1$ is the decay factor and $b(x, t) \in \{0, 1\}$ is 1 if x participated in the conflict[1] The decay factor can be a constant or change over time.

In VMTF [Rya04] the score is $q(x, t) = t$ if x participated in the conflict[2] and $q(x, t-1)$ otherwise. The relative order among tied variables is preserved.

In ACIDS [BF15] the score is $(t + q(x, t))/2$ if x participated in the conflict and $q(x, t - 1)$ otherwise.

In CHB [LGPC16a] the score is updated after every set of propagations leading to a stable state or conflict s, but only for variables added to the trail since state $s - 1$; the rest is not modified. If variable x is updated at state s, then $q(x, s) = \alpha \cdot r(x, s) + (1 - \alpha) \cdot q(x, s - 1)$. $r(x, s) = \mu(s)/\Delta(x, s)$, where $\mu(s) = 1$ if we reached a conflict and 0.9 otherwise, and $\Delta(x, s)$ is the number of conflicts since x last participated in a conflict. The MapleSAT implementation of CHB has α decreasing after each conflict.

In LRB [LGPC16b] the score is updated after each conflict or restart, but only for variables getting unassigned. We have $q(x, s)$ defined as in CHB, but $r(x, s) = P(x, s)/I(x, s)$, where $P(x, s)$ is the number of times x participated in a conflict or belonged to a clause that propagated a literal found

---

[1]The original Chaff implementation has a score for each literal, a literal is bumped if it belongs to the learned clause, and decays and reordering happen once every 256 conflicts.

[2]The original Siege implementation only bumps variables in the learned clause.

in the learned clause, and $I(x,s)$ is the number of conflicts, all of these measured since x was last assigned. We also decay the score of all variables by a factor δ after each conflict.

2.4 Implementation Details

2.4.1 Precision

If we expand the definition of VSIDS we get that $q(x,t) = \sum_{i=1}^{t} \delta^{t-i} b(x,i)$. However, since in practice we keep the scores using floating point numbers, it is more accurate for our model to drop the tail of the sum. In other words we only need to account for the last K conflicts, where K is such that

$$\sum_{i=1}^{t-K} \delta^{t-i} = \frac{\delta^K - \delta^t}{1-\delta} < \frac{\delta^K}{1-\delta} < \epsilon \tag{2}$$

with ϵ being the minimum number that can be represented with a particular floating point type. If we have e bits of exponent, this means

$$K > \frac{\log(\epsilon(1-\delta))}{\log(\delta)} = \frac{-2^e + \log(1-\delta)}{\log(\delta)} \ . \tag{3}$$

To put this in numbers, MiniSat uses $e = 10$ and $\delta = 0.95$, hence only about the last $14\,000$ conflicts matter.

Asymptotically, if we make the assumption that to solve a problem of size n we use data types of size $\Theta(\log n)$ in bits, then we have $\Theta(\log n)$ bits of floating point exponent, which gives a value for K of $2^{\Theta(\log n)} = n^{\Theta(1)}$, that is only a polynomial number of recent conflicts matter.

In our analysis we need to distinguish variables that participated in a conflict up to exponentially many conflicts ago from variables that did not participate in a conflict at all, but as we just discussed their scores will be both indistinguishable from 0. Therefore we need to pay attention to tie-breaking. Ultimately which variable gets decided among variables of equal score depends on which data structure is used to implement the priority queue of unassigned variables, hence we consider two kinds of data structures depending on whether they preserve the preexisting order.

A stable priority queue—and, by extension, a stable decision heuristic—has the property that if at a state s we have $q(x,s) > q(y,s)$ and for every state $s' > s$ we have $q(x,s') \geq q(y,s')$, then x is dequeued before y. An example of a stable priority queue is a radix heap [BF15], while an example of an unstable priority queue is the binary heap used in MiniSat. The split queue present in recent versions of Lingeling [Bie15] is only stable with respect to zero and near-zero scores, hence for our purposes we consider it to be stable.

Since in practice the choice of data structure is not particularly consequential [BF15], whenever we need to we will assume that we use a stable priority queue and make an explicit note of it.

2.4.2 Initialization

As we mentioned we assume that when deciding a variable that has never been assigned we set its phase statically to 0. Other ways to choose an initial phase are with the one-sided Jeroslow–Wang heuristic or at random, but we do not consider those.

Some ways to pick the initial variable order are to use the same static order as the input, the JW order, or to pick a random order. To make our model more robust we assume the order is random and prove that our result holds except for an exponentially small fraction of initial orders.

2.5 Preprocessing

Most solvers use preprocessing techniques in an attempt to simplify their inputs before and even interleaved with the search. While we do not aim to cover all of these techniques, we do not want

our formulas to contain pure literals, unit clauses, or similar artefacts that can be easily removed. Therefore we look into the following simple preprocessing rules.

- Self-subsuming resolution [EB05]. If we have clauses $C \vee l$ and $D \vee \bar{l}$, and $C \subseteq D$, then we remove $\bar{l}$ from D.

- Bounded variable elimination [EB05]. Resolve all clauses containing x with all clauses containing $\overline{x}$. If the resulting set is not larger, replace the original set.

- Failed literal elimination [LB01]. If setting a literal l unit propagates a conflict, add the unit clause $\bar{l}$.

- Blocked clause elimination [JBH10]. If a clause contains a literal l such that resolving C with all clauses containing $\bar{l}$ only yields tautologies, then we remove C.

3 Separation for Conflict-Rewarding Heuristics

The key property that we use from the decision heuristics we study is that they give priority to variables participating in conflicts.

Definition 3.1. We say that a decision heuristic *rewards conflicts* if a variable that has participated in a conflict is always selected before a variable that has never been assigned.

To prove that concrete heuristics reward conflicts we assume that on a problem of size n the solver uses a floating point type with $\Theta(\log n)$ bits of exponent and runs for $\exp(O(n))$ many steps.

Lemma 3.2. *Stable VSIDS, VMTF, ACIDS, CHB, and Stable LRB reward conflicts.*

Proof. Let x be a variable that participated in conflict t, and y a variable that has never been assigned. All heuristics we consider satisfy $q(x,s) \geq 0$ and $q(y,s) = 0$ for all states s, hence for stable decision heuristics it is enough to show that $q(x,t) > 0$, while for the rest we have to show that $q(x,s) > 0$ for all states $s \geq t$.

It is immediate from the definitions that for stable VSIDS we have $q(x,t) \geq 1$, and for VMTF and ACIDS we have $q(x,s) \geq q(x,t) = t$.

For CHB we have $q(x,s) \geq \alpha\mu/\Delta(x,s) > 1/(20s)$ for all $s \geq t$, and since $s = \exp(O(n))$, a floating-point representation can distinguish $1/(20s)$ from 0.

Finally, for stable LRB, even if we cannot guarantee that $q(x,t) > 0$, we only need to prove that x is selected before y when both choices are available, and as soon as we reach a state $s > t$ where x is unassigned the score of x is updated to $q(x,s) \geq \alpha/t > 1/(20t)$. $\square$

To exploit this property we craft a formula with an easy part and a hard part, but such that setting variables from the easy part makes the solver find a conflict involving all variables from the hard part.

The formula has 5 types of variables, each type subdivided into k blocks: $X = \bigcup_{j \in k} X_j$ with $|X_j| = m$ are hard variables, $Y = \bigcup_{j \in k} Y_j$ with $|Y_j| = n$ are easy variables, and $P = \bigcup_{j \in k} P_k$ with $|P_j| = m + n$, $Z = \bigcup_{j \in k} Z_j$ with $|Z_j| = n$, and $A = \bigcup_{j \in k} A_j$ with $|A_j| = 3$ are auxiliary variables. This makes a total of $k(2m + 3n + 3)$ variables. We think of k as a constant and assume that $m = \Theta(n)$.

Next we describe a few gadgets out of which we build the formula. The hard gadget consists of a padded formula, that is given a formula F and a clause C, the padded formula $F \vee C$ is $\{D \vee C \mid D \in F\}$.

A *pitfall gadget* $\Psi(y_1, y_2)$ is is the CNF $y_1 \vee y_2 \vee \overline{p_{j,1}}, y_1 \vee y_2 \vee \overline{p_{j,2}}, \ldots, y_1 \vee y_2 \vee \overline{p_{j,m+n}}$, where j is the block index of both y_1 and y_2. We call y_1 and y_2 the trap variables. The goal of a pitfall gadget is to deviate the solver's attention from Y variables towards X variables via P variables: observe that setting $y_1 = 0, y_2 = 0$ leads to a parallel propagation of P_j.

A *pipe gadget* $\Pi(y)$ is the CNF $y \vee \bigvee_{i \neq 1} p_{j,i} \vee \overline{x_{j,1}}$, $y \vee \bigvee_{i \neq 2} p_{j,i} \vee x_{j,1} \vee \overline{x_{j,2}}, \ldots, y \vee \bigvee_{i \neq m} p_{j,i} \vee x_{j,1} \vee \cdots \vee \overline{x_{j,m}}$, $y \vee \bigvee_{i \neq m+1} p_{j,i} \vee x_{j,1} \vee \cdots \vee x_{j,m} \vee \overline{z_{j,1}}, \ldots, y \vee \bigvee_{i \neq m+n} p_{j,i} \vee x_{j,1} \vee \cdots \vee x_{j,m} \vee z_{j,2} \vee \cdots \vee \overline{z_{j,n}}$, where j is the block index of y. Note that the last clause does not contain $z_{j,1}$. The goal of a pipe gadget is to follow-up on a pitfall gadget: setting y and all P_j variables to 0 leads to a sequential propagation of all variables in $X_j \cup Z_j$.

A *tail gadget* $\Delta(y, z)$ is the CNF $\overline{a_{j,1}} \vee a_{j,3} \vee \overline{z}$, $\overline{a_{j,2}} \vee \overline{a_{j,3}} \vee \overline{z}$, $a_{j,1} \vee \overline{z} \vee \overline{y}$, $a_{j,2} \vee \overline{z} \vee \overline{y}$, where j is the block index of both y and z. The goal of a tail gadget is to ensure that Z variables do not flip values: if $y = 1$, then setting $z = 1$ immediately leads to a conflict.

We denote $\Psi_j = \left\{ \Psi(y_1, y_2) \mid (y_1, y_2) \in \binom{Y_j}{2} \right\}$, $\Pi_j = \{\Pi(y) \mid y \in Y_j\}$, and $\Delta_j = \{\Delta(y, z) \mid (y, z) \in Y_j \times Z_j\}$.

Definition 3.3. The pitfall formula $\Phi(G, \chi, n, k)$ consists of k padded copies of an unsatisfiable Tseitin formula, $Ц_j = Ц(G, \chi) \vee \bigvee_{z_i \in Z_j} z_i$, k pitfall gadgets Ψ_j, k pipe gadgets Π_j, k tail gadgets Δ_j, and $n/2$ clauses $\Gamma_\ell = \bigvee_{j \in [k]} \overline{y_{j,2\ell}} \vee \overline{y_{j,2\ell+1}}$, which we refer to as easy clauses.

The total number of clauses on a graph of degree d is therefore $k \cdot \big((2m/d) \cdot 2^{d-1} + \big(\binom{n}{2} + n \big) \cdot (m + n) + 4n^2 + n/2 \big)$. Note that there is nothing special about a Tseitin formula, and we could replace it with other hard formulas.

We claim that the formula is not affected by the preprocessing rules we described in Section 2.5.

Lemma 3.4. Φ *is invariant with respect to self-subsuming resolution.*

Proof. The support of each gadget is $Ц(X_j, Z_j)$, $\Psi(Y_j, P_j)$, $\Pi(Y_j, P_j, X_j, Z_j)$, $\Delta(Y_j, Z_j, A_j)$, and $\Gamma(Y)$. We only need to worry about self-subsuming resolution from Tseitin and pitfall gadgets to pipe gadgets, or between the same gadget, since all other gadgets have supports that are incomparable with respect to set inclusion.

Since Tseitin clauses contain all Z_j variables but no pipe clause does, no subsumption occurs in the first two cases. Pitfall gadgets contain two Y_j variables but pipe gadgets only contain one, covering the second case.

Pairs of Tseitin clauses either have incomparable supports or cannot be resolved, and pairs of pitfall clauses, pipe clauses, tail clauses, and easy clauses all have incomparable supports respectively, hence no subsumption occurs in the last case. $\square$

Lemma 3.5. Φ *is invariant with respect to bounded variable elimination.*

Proof. For each type of variable we count how many clauses a variable appears in, and how large the result of eliminating the variable is.

Variable $x_{j,i}$ appears in 2^d Tseitin clauses, $m - i + n$ times positively in n pipe gadgets, and once negatively in the same gadgets, which is $O(n^2)$ in total. If we resolve clauses $C_i \in \Pi$ and $\{C'_{m+1}, \ldots, C'_{m+n}\} \subset \Pi'$ for any pair of pipe gadgets we obtain different clauses, hence eliminating $x_{j,i}$ produces $\Omega(n^3)$ clauses.

Variable $y_{j,i}$ appears $m + n$ times positively in $n - 1$ pitfall gadgets, $m + n$ times positively in a pipe gadget, twice negatively in n tail gadgets, and once in an easy clause, or $O(n^2)$ in total. If we resolve clauses from a pitfall gadget and clauses from a tail gadget we obtain different clauses, hence eliminating $y_{j,i}$ produces $\Omega(n^3)$ clauses.

Variable $p_{j,i}$ appears once negatively in $\binom{n}{2}$ pitfall gadgets and $n + m - 1$ times positively in n pipe gadgets, or $O(n^2)$ times in total. If we resolve these clauses we obtain $\Omega(n^4)$ clauses, of which $O(n^3)$ are repeated, therefore $\Omega(n^4)$ are distinct clauses.

Variable $z_{j,i}$ appears once positively in $v \cdot 2^{d-1}$ Tseitin clauses, $\min(n-2, n-i)$ times positively in n pipe gadgets, and once negatively in the same gadgets, and 4 times negatively in n tail gadgets, or $n^2 + O(n)$ times in total. If we resolve Tseitin clauses and tail clauses we obtain different clauses, hence eliminating $z_{j,n}$ produces $v \cdot 2^{d+1} \cdot n > n^2$ clauses.

Variable $a_{j,i}$ appears once positively and once negatively in n^2 tail gadgets. If we resolve the corresponding clauses for any pair of tail gadgets we obtain different clauses, hence eliminating $a_{j,i}$ produces n^4 clauses. $\qquad\square$

Lemma 3.6. Φ *is invariant with respect to failed literal elimination.*

Proof. Since Φ does not contain any binary clauses, no conflict occurs after assigning only one literal. $\qquad\square$

Lemma 3.7. Φ *is invariant with respect to blocked clause elimination.*

Proof. For each clause C and literal l we show that there is a clause D containing $\bar{l}$ such that resolving C and D is not tautological.

Tseitin clauses can be resolved over a variable $x_{j,i}$ with the Tseitin clauses corresponding to the opposite endpoint of edge x_i, and over a literal $z_{j,i}$ with clauses on a tail gadget.

Pitfall clauses can be resolved over a literal $y_{j,i}$ with clauses on a tail gadget, and over a literal $\overline{p_{j,i}}$ with clauses on a pipe gadget.

Pipe clauses can be resolved over a literal $y_{j,i}$ with clauses on a tail gadget, over a literal $p_{j,i}$ with clauses on a pitfall gadget, and over a literal $x_{j,i}$ or $z_{j,i}$ with a pipe clause containing $\overline{x_{j,i}}$ or $\overline{z_{j,i}}$ and vice versa, except for the literal $\overline{z_{j,n}}$, which can be resolved with a Tseitin clause where all X variables appear positively. Such a clause is present in the expansion of $Parity(v)$ with $\chi(v)$ odd.

Tail clauses can be resolved over a variable $a_{j,i}$ with another tail clause, over a literal $\overline{y_{j,i}}$ with a pitfall clause, and over a variable $\overline{z_{j,i}}$ with a Tseitin clause.

Easy clauses can be resolved with pipe clauses. $\qquad\square$

Next we move into proving Theorem 1.1. First we show how to build short proofs of pitfall formulas.

Lemma 3.8. *There is a resolution refutation of $\Phi(G, \chi, n, k)$ of length $2^{O(k^2)} + O(kn^2(n+m))$.*

Proof. For any pair of variables $y_{j,\ell}, y_{j,\ell'}$ we can derive the clause $y_{j,\ell} \vee y_{j,\ell'}$ in $O(n)$ steps as follows.

First we derive the clause $y_{j,\ell} \vee p_{j,1} \vee \cdots \vee p_{j,m+n}$. Let $\Pi(y_{j,\ell}) = C_1, \ldots, C_{m+n}$ be the clauses of the corresponding pipe gadget and let $D \in Ц_j$ be a clause that is not satisfied by setting $X_j = 0$, that is $D = x_{j,i_1} \vee \cdots \vee x_{j,i_d} \vee z_{j,1} \vee \cdots \vee z_{j,n}$. We resolve D with the pipe clauses in reverse, that is, we begin with $D_{m+n} = D$ and at each step from $i = m+n$ to 1 we resolve D_i with C_i to obtain D_{i-1}. Note that D_{i-1} is the same as C_i with the last literal removed and the literal $p_{i,j}$ added, and in particular $D_0 = y_{j,\ell} \vee y_{j,\ell'} \vee p_{j,1} \vee \cdots \vee p_{j,m+n}$.

This allows us to derive the clause $y_{j,\ell} \vee y_{j,\ell'}$ in $m + n$ steps simply by resolving $y_{j,\ell} \vee p_{j,1} \vee \cdots \vee p_{j,m+n}$ with all the clauses in the corresponding pitfall gadget.

Using this procedure we can derive the clauses $y_{j,\ell} \vee y_{j,\ell'}$ for $j \in [k]$ and $(\ell,\ell') \in \binom{[2(k+1)]}{2}$ in $O(kn^2(n+m))$ steps. Together with the first $k+1$ easy clauses $\Gamma_1, \ldots, \Gamma_{k+1}$, these form a formula F over $2k(k+1) = O(k^2)$ variables. We claim that F is unsatisfiable, therefore there is a resolution refutation of F of length $2^{O(k^2)}$, proving the lemma.

To prove that F is unsatisfiable we reason by contradiction and assume there is an assignment that satisfies F. In order to satisfy all $\Gamma_1, \ldots, \Gamma_{k+1}$ clauses at least $k+1$ variables must be set to 0, hence by the pigeonhole principle there is at least one block j with two variables set to 1. But this falsifies a clause of the form $y_{j,\ell} \vee y_{j,\ell'}$, contradicting that an assignment exists. $\qquad\square$

We end the section proving the hard part of Theorem 1.1, namely that CDCL requires exponential time to solve pitfall formulas.

Theorem 3.9. *There exists $\epsilon > 0$ and a family of graphs G_n of constant degree such that CDCL with a decision heuristic that rewards conflicts runs for T steps on input $\Phi(G_n, \xi, n, k)$ with $\Pr[T \leq 2^{\epsilon m}] \leq 2^{-\epsilon n}$.*

To prove Theorem 3.9 we introduce the following solver states. In state (a) no conflict has occurred, all assigned variables are set to 0, no pair of Y_j variables is assigned, and for each set Z_j and P_j at least three variables are unassigned. State (b) is like state (a), except that a pair of Y_j variables is assigned. In state (c) variables $X_j \cup Z_j \cup \{p_{i,j} \in P_j \mid i > 1\}$ plus one $y_{j,i}$ variable have participated in a conflict; no variable with an index $j' \neq j$ has participated in a conflict; all assigned variables with index j' are set to 0; no pair of $Y_{j'}$ variables is assigned; and for each set $Z_{j'}$ and $P_{j'}$ at least three variables are unassigned.

The solver starts in state (a) and, since Φ is unsatisfiable, ends after reaching a conflict that cannot be resolved. We prove that with high probability the solver moves from state (a) to (b) to (c) and ends at state (c).

Lemma 3.10. *When the solver moves away from state (a), it moves to state (b) with probability* $1 - \exp(-\Omega(n))$.

Lemma 3.11. *The solver moves from state (b) to state (c).*

Lemma 3.12. *The solver never leaves state (c).*

This is enough to prove Theorem 3.9.

Proof of Theorem 3.9. Consider the resolution proof π induced by a solver run that moves from state (a) to (b) to (c). We hit π with a restriction ρ that sets variables in the following way: X variables remain unset, Y and P variables are assigned to 1, and Z and A variables are assigned to 0. Observe that ρ falsifies Φ, so for all we know it could be that $\pi\!\restriction_\rho$ is trivial, but in fact the only clauses that ρ falsifies are Γ clauses, and $(\Phi \setminus \Gamma)\!\restriction_\rho = \text{Ц}(G, \chi)$. Therefore if we show that π never uses Γ clauses as axioms then we will have that $\pi\!\restriction_\rho$ is a refutation of $\text{Ц}(G, \chi)$, hence by Theorem 2.1 it has length $\exp(\Omega(m))$, and by Lemma 2.2 it takes $T = \exp(\Omega(m))$ conflicts to produce.

To prove that π never uses Γ clauses, by Lemma 2.2 it is enough to to show that Γ clauses never unit-propagate or conflict, which is true since each easy clause contains two variables from each set Y_j, and for all but one of the sets Y_j there is only one variable ever assigned, so at least $k-1$ variables of each Γ clause remain unassigned at all times. □

We finish the proof of the lower bound by proving the main technical lemmas.

Proof of Lemma 3.10. We begin analysing which clauses might unit propagate. At an (a) state padded Tseitin clauses do not propagate because at least two of the Z variables are unset. The clauses of a pitfall gadget can propagate only if both trap variables are set; otherwise variable $p_{j,i}$ is set to 0 and satisfies the clause. Pipe clauses do not propagate because at least two of the P variables are unset. Tail gadgets do not propagate because literals $\overline{y}$ and $\overline{z}$ are either unset or satisfied, and easy clauses do not propagate because each clause has at least half of its variables unset.

Assume that we are at a stable (a) state and the next action is to decide a variable. If it is an X or A variable, then no propagation happens and the next state is a stable (a) state. If a Y variable $y_{j,\ell}$ is decided and another Y_j variable is already assigned, then the next state is a (b) state; otherwise no propagation happens and the next state is a stable (a) state. If a P or Z variable is decided then no propagation happens as long as the variable was not the $m + n - 2$-th P_j or $(n - 2)$-th Z_j variable to be assigned.

Therefore it is enough to prove that with high probability we decide a pair of Y_j variables before any set of $n - m - 2$ P_j or $n - 2$ Z_j variables are decided. In fact we prove that with high probability we decide $k + 1$ Y variables before any $n - 2$ variables in $P \cup Z$ are decided, which implies the first event.

Let E be the latter event. Then $\Pr[E] = \Pr[H \geq k + 1]$, where H is the random variable measuring the number of Y variables decided among the first $n + k - 1$ decided $Y \cup Z$ variables, that is a

9

hypergeometric random variable with a population of $N = k(3n + m)$, a success population of $K = kn$, and a number of draws of $n' = n + k - 1$.

Using the tail bound $\Pr[H \leq (K/N - t)n'] \leq \exp(-2t^2 n')$ for the hypergeometric distribution [Chv79] we obtain

$$\Pr[H \leq k] \leq \exp\left(-2 \cdot \left(\frac{n}{3n + m} - \frac{k}{n + k - 1}\right)^2 \cdot (n + k)\right)\exp(-\Omega(n)) \ . \tag{4}$$

$\square$

Proof of Lemma 3.11. Let $y_{j,\ell}, y_{j,\ell'}$ be the only pair of assigned Y variables and $\Psi^* = \Psi(y_{j,\ell}, y_{j,\ell'}) = C_1, \ldots, C_{m+n}$, $\Pi^\ell = \Pi(y_{j,\ell}) = D_1, \ldots, D_{m+n}$, and $\Pi^{\ell'} = \Pi(y_{j,\ell'}) = D'_1, \ldots, D'_{m+n}$ be the corresponding pitfall and pipe gadgets.

The analysis of which clauses unit propagate is analogous to that in Lemma 3.10 for (a) states, except that now the Ψ^*, Π^ℓ, and $\Pi^{\ell'}$ gadgets may propagate: for each assigned variable $p_{j,i}$ the clause C_i is satisfied, for each assigned variable $x_{j,i}$ clauses D_i and D'_i are satisfied, and for each assigned variable $z_{j,i}$ clauses D_{m+i} and D'_{m+i} are satisfied, but the remaining clauses of these gadgets can propagate.

Let h be the smallest number such that $z_{j,i} = 0$ for all $h < i \leq n$ and note that $2 < h \leq n$. Let h' be the smallest number such that $z_{j,i} = 0$ for all $h' < i \leq n$, $i \neq h$. Let σ be the sequence of assignments at state (b), and let τ be the trail $\langle x_{j,1} = 0, \ldots, x_{j,m} = 0, z_{j,1} = 0, \ldots, z_{j,n} = 0 \rangle$ interleaved with a permutation of the trail $\langle p_{j,1} = 0, \ldots, p_{j,m+n} = 0 \rangle$. We show by induction that up to the first conflict the trail is of the form $\langle \sigma, \tau' \rangle$, where τ' is a prefix of $\tau \setminus \sigma$.

The induction hypothesis is true at state (b) with $\tau = \emptyset$, at which point all Ψ^* clauses $C_1, \ldots C_{m+n}$ that are not satisfied propagate immediately and in an arbitrary order, hence we can assume that all $p_{j,i}$ variables are set to 0 from now on. The remaining clauses propagate in increasing order: assuming $x_{j,1}$ is unset, the first clause to propagate is either D_1 or D'_1, which sets $x_{j,1} = 0$. Only after this D_2 or D'_2 can propagate $x_{j,2} = 0$, and so on. Eventually we reach a state where $z_{j,1}, \ldots, z_{j,h'-1}$ are all assigned to 0, at which point the solver unit propagates $z_{j,h'} = 0$ and, since $z_{j,h'+1}, \ldots, z_{j,h-1}$ are also assigned to 0, we claim that the solver reaches a conflict with variable $z_{j,h}$.

More specifically we have that those clauses of Ц$_j$ that are not satisfied propagate $z_{j,h} = 1$—and since Ц is unsatisfiable at least one such clause exists—, and either D_{m+h} or D'_{m+h} propagates $z_{j,h} = 0$. Assume without loss of generality it is D_{m+h}. Note that even though clauses $D_{m+h+1}, \ldots, D_{m+n}$ and $D'_{m+h+1}, \ldots, D'_{m+n}$ contain the literal z_h, they are all satisfied. Hence, the first input resolution step during conflict analysis is to resolve D_{m+h} with some clause $E \in$ Ц$_j$. Note that D_{m+h} contains $X_j \cup \{p_{i,j} \in P_j \mid i > 1\}$ as well as $y_{j,\ell}$, and E contains Z_j, therefore all of these variables participated in the first conflict. $\square$

Proof of Lemma 3.12. At a state of type (c), the set of variables that have been involved in a conflict is bounded below by the set S_- comprising $X_j \cup Z_j \cup \{p_{i,j} \in P_j \mid i > 1\}$ and a variable $y \in Y_j$. The set of variables that have been ever assigned is bounded above by the set S_+ comprising all of X, P_j, Z_j, and A, one variable from each set $Y_{j'}$, and all but three variables from each set $Z_{j'}$ and P'_j. We need to prove that we never reach a conflict where $Y_{j'}$ variables participate, and in fact we prove that we never propagate any variable from a j' block.

We claim it is enough to prove that (i) setting all of the variables in S_- always produces a conflict and that (ii) setting all of the variables in S_+ does not propagate any variable with index j'. Indeed, since the decision heuristic rewards conflicts, all of the variables in S_- must be assigned before a variable not in S_+ is decided, and by item (ii) no such variable is propagated either. But by item (i), once all of the variables in S_- are assigned we reach a conflict. After resolving the conflict either part of S_- is unassigned, or we reach another conflict, or the formula is found unsatisfiable and the solver terminates.

To prove claim (i) we assume that all variables in S_- are assigned and consider the set of Z_j variables. If all variables are set to 0, then we have that $Ц_j{\restriction}_{Z_j=0}$ is an unsatisfiable formula over X variables, all of which are assigned, hence at least one clause of $Ц_j$ is falsified. Thus we can assume that some variable $z \in Z_j$ is set to 1. If $y = 1$ then we reach a conflict with one of the clauses of the tail gadget $\Delta(y, z)$, possibly after propagating $a_1 = 1$ and $a_2 = 1$, hence we can assume that $y = 0$. If any P_j variable is set to 1 then a variable y' gets propagated to 1 by the pitfall gadget $\Psi(y, y')$ and we reach a conflict with the gadget $\Delta(y', z)$. Therefore we can assume that all P_j variables are set to 0, except possibly for $p_{j,1}$, which may be unassigned. Let C be the first clause in $\Pi(y)$ that is not satisfied, which exists because $z = 1$. If $p_{j,1} = 0$ then C is falsified, otherwise C propagates $p_{j,1} = 1$ and we argue as with other P variables.

To prove claim (ii) observe that only easy clauses share variables across blocks, hence setting all the variables in the j-th block does not affect a gadget with variables in the j'-th block. Easy clauses do not unit propagate since at most one variable out of each $Y_{j'}$ set of variables is assigned, at least $k-1$ variables in each easy clause remain unassigned. As for gadgets with variables in the j'-th block, since all j' variables that are assigned are set to 0 by the phase saving heuristic, the same analysis of Lemma 3.10 for (a) states applies. $\qquad\square$

4 Experimental Evaluation

We ran experiments on pitfall formulas using the CaDiCaL 1.0.3 [Bie17], Glucose 4.1 [AS09], and MapleCOMSPS [LOG+16] solvers, representing the VMTF, VSIDS, and CHB and LRB heuristics respectively. We also employed an instrumented version of Glucose that allows choosing a static variable order [EGG+18].

In order to work with smaller instances we slightly modify the formula description to allow different numbers of Y and Z variables. As the base graph for the Tseitin gadget we use a random regular graph of degree 4 and either 45 or 50 vertices. The graphs are chosen so that the Tseitin formula of the smaller graph is solved in a few hundred seconds, while the Tseitin formula of the larger graph cannot be solved in less than one hour. We set the block size of Y and Z variables to 30 and 5 respectively, which we experimentally found to be large enough to make the formulas hard, and we use two amounts of blocks, 6 and 8.

We ran the solvers with their default parameters, except that we disabled pre- and inprocessing since these turn out to be counterproductive, we turned on VMTF and the *unsat* option in CaDiCaL, and we disabled the adaptive strategies of Glucose to obtain more consistent results. We chose the *pure* versions of MapleCOMSPS to avoid interference from other heuristics. The static order decides on Y, X, Z, P, and A variables in this order.

To account for different ways to initialize the variable order we shuffled the variables and clauses (but not the polarities) with 32 different seeds and report the aggregated result, assigning a time of 3600 s to unsolved instances. We do not do any shuffling when running with a fixed order.

We can observe in Table 1 that, as predicted, solvers do not do better than if they had to solve a standalone Tseitin formula, while a fixed variable order is faster by several orders of magnitude.

We also tested the effect of mixing VSIDS with random decisions by running Glucose with a probability of choosing a random variable ranging from 0 to 1. We find that interleaving a moderate amount of random decisions, say with probability 0.2, is helpful, but as a side effect the runtime variance increases. Random decisions decrease the average runtime and for instance the hardest formulas $\Phi(50, 6)$ and $\Phi(50, 8)$ become (barely) solvable.

Table 1: Mean CPU time to solve (s)

Formula	CaDiCaL	Glucose	Maple CHB	Maple LRB	Static
Ц(45)	3331	754	621	424	3600
$\Phi(45, 6)$	2228	1917	600	2598	< 1
$\Phi(45, 8)$	1963	2273	607	2650	< 1
Ц(50)	3600	3600	3600	3600	3600
$\Phi(50, 6)$	3600	3600	3600	3600	< 1
$\Phi(50, 8)$	3600	3600	3600	3600	< 1

5   Concluding Remarks

We proved that CDCL with conflict-rewarding heuristics does not polynomially simulate resolution. This does not mean we should throw away these heuristics, but it does imply that there is room for improvement. Since random choices help, perhaps one could go a step further and actively choose variables that have been long unused.

Speaking of randomness, it would be interesting to determine whether our formula has short proofs using a random decision heuristic. Note that this does not follow immediately from the result of Atserias et al. [AFT11] since the formula contains long clauses.

One annoyance of our result is that in some cases we need to assume a stable data structure. It might be possible to improve our construction and/or analysis to ensure that the scores of all variables in the hard part of the formula are periodically refreshed, which would allow us to get rid of the stability assumption.

It would also be interesting to find smaller formulas that exhibit the same properties, so that size does not become a factor that can influence experiments.

Acknowledgements

References

[AFT11]   Albert Atserias, Johannes Klaus Fichte, and Marc Thurley. Clause-learning algorithms with many restarts and bounded-width resolution. *Journal of Artificial Intelligence Research*, 40:353–373, January 2011. Preliminary version in *SAT '09*.

[AM19]   Albert Atserias and Moritz Müller. Automating resolution is NP-hard. In *Proceedings of the 60th IEEE Symposium on Foundations of Computer Science (FOCS '19)*, November 2019. To appear.

[AS09]   Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern SAT solvers. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI '09)*, pages 399–404, July 2009.

[BBJ14]   María Luisa Bonet, Sam Buss, and Jan Johannsen. Improved separations of regular resolution from clause learning proof systems. *Journal of Artificial Intelligence Research*, 49:669–703, April 2014.

[BEGJ00]   María Luisa Bonet, Juan Luis Esteban, Nicola Galesi, and Jan Johannsen. On the relative complexity of resolution refinements and cutting planes proof systems. *SIAM Journal on Computing*, 30(5):1462–1484, 2000. Preliminary version in *FOCS '98*.

[BF15]   Armin Biere and Andreas Fröhlich. Evaluating CDCL variable scoring schemes. In *Proceedings of the 18th International Conference on Theory and Applications of Satisfiability Testing (SAT '15)*, volume 9340 of *Lecture Notes in Computer Science*, pages 405–422. Springer, September 2015.

[Bie15]   Armin Biere. Lingeling and friends entering the SAT Race 2015. Technical Report 15/2, FMV Report Series, Institute for Formal Models and Verification, Johannes Kepler University, April 2015.

[Bie17]   Armin Biere. CaDiCaL, Lingeling, Plingeling, Treengeling, and YalSAT entering the SAT competition 2017. In *Proceedings of SAT Competition 2017: Solver and Benchmark Descriptions*, pages 14–15, 2017.

[BK14]   Samuel R. Buss and Leszek Kołodziejczyk. Small stone in pool. *Logical Methods in Computer Science*, 10(2):16:1–16:22, June 2014.

[BKS04]   Paul Beame, Henry Kautz, and Ashish Sabharwal. Towards understanding and harnessing the potential of clause learning. *Journal of Artificial Intelligence Research*, 22:319–351, December 2004. Preliminary version in *IJCAI '03*.

[Chv79]   Vasek Chvátal. The tail of the hypergeometric distribution. *Discrete Mathematics*, 25(3):285–287, 1979.

[EB05]   Niklas Eén and Armin Biere. Effective preprocessing in SAT through variable and clause elimination. In *Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing (SAT '05)*, pages 61–75, June 2005.

[EGG⁺18]   Jan Elffers, Jesús Giráldez-Cru, Stephan Gocht, Jakob Nordström, and Laurent Simon. Seeking practical CDCL insights from theoretical SAT benchmarks. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI '18)*, pages 1300–1308, July 2018.

[EJL⁺16]   Jan Elffers, Jan Johannsen, Massimo Lauria, Thomas Magnard, Jakob Nordström, and Marc Vinyals. Trade-offs between time and memory in a tighter model of CDCL SAT solvers. In *Proceedings of the 19th International Conference on Theory and Applications of Satisfiability Testing (SAT '16)*, volume 9710 of *Lecture Notes in Computer Science*, pages 160–176. Springer, July 2016.

[ES04]   Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In *6th International Conference on Theory and Applications of Satisfiability Testing (SAT '03), Selected Revised Papers*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2004.

[Goe93]   Andreas Goerdt. Regular resolution versus unrestricted resolution. *SIAM Journal on Computing*, 22(4):661–683, August 1993.

[HBPV08] Philipp Hertel, Fahiem Bacchus, Toniann Pitassi, and Allen Van Gelder. Clause learning can effectively P-simulate general propositional resolution. In *Proceedings of the 23rd National Conference on Artificial Intelligence (AAAI '08)*, pages 283–290, July 2008.

[JBH10] Matti Järvisalo, Armin Biere, and Marijn Heule. Blocked clause elimination. In *Proceedings of the 16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '10)*, pages 129–144, March 2010.

[JJ09] Matti Järvisalo and Tommi A. Junttila. Limitations of restricted branching in clause learning. *Constraints*, 14(3):325–356, 2009. Preliminary version in *CP '07*.

[KSM11] Hadi Katebi, Karem A. Sakallah, and João P. Marques-Silva. Empirical study of the anatomy of modern SAT solvers. In *Proceedings of the 14th International Conference on Theory and Applications of Satisfiability Testing (SAT '11)*, volume 6695 of *Lecture Notes in Computer Science*, pages 343–356. Springer, June 2011.

[LB01] Daniel Le Berre. Exploiting the real power of unit propagation lookahead. *Electronic Notes in Discrete Mathematics*, 9:59–80, 2001.

[LGPC16a] Jia Hui Liang, Vijay Ganesh, Pascal Poupart, and Krzysztof Czarnecki. Exponential recency weighted average branching heuristic for SAT solvers. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI '16)*, pages 3434–3440, February 2016.

[LGPC16b] Jia Hui Liang, Vijay Ganesh, Pascal Poupart, and Krzysztof Czarnecki. Learning rate based branching heuristic for SAT solvers. In *Proceedings of the 19th International Conference on Theory and Applications of Satisfiability Testing (SAT '16)*, volume 9710 of *Lecture Notes in Computer Science*, pages 123–140. Springer, July 2016.

[LOG+16] Jia Hui Liang, Chanseok Oh, Vijay Ganesh, Krzysztof Czarnecki, and Pascal Poupart. MapleCOMSPS, MapleCOMSPS_LRB, MapleCOMSPS_CHB. In *Proceedings of SAT Competition 2016: Solver and Benchmark Descriptions*, pages 52–53, 2016.

[MMZ+01] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference (DAC '01)*, pages 530–535, June 2001.

[MN14] Mladen Mikša and Jakob Nordström. Long proofs of (seemingly) simple formulas. In *Proceedings of the 17th International Conference on Theory and Applications of Satisfiability Testing (SAT '14)*, volume 8561 of *Lecture Notes in Computer Science*, pages 121–137. Springer, July 2014.

[MSLM09] João P. Marques-Silva, Inês Lynce, and Sharad Malik. Conflict-driven clause learning SAT solvers. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, chapter 4, pages 131–153. IOS Press, 2009.

[PD11] Knot Pipatsrisawat and Adnan Darwiche. On the power of clause-learning SAT solvers as resolution engines. *Artificial Intelligence*, 175(2):512–525, February 2011. Preliminary version in *CP '09*.

[Rya04] Lawrence Ryan. Efficient algorithms for clause-learning SAT solvers. Master's thesis, Simon Fraser University, 2004.

[Urq87] Alasdair Urquhart. Hard examples for resolution. *Journal of the ACM*, 34(1):209–219, January 1987.

[Urq95]   Alasdair Urquhart. The complexity of propositional proofs. *Bulletin of Symbolic Logic*, 1(4):425–467, 1995.

[Van05]   Allen Van Gelder. Pool resolution and its relation to regular resolution and DPLL with clause learning. In *Proceedings of the 12th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR '05)*, volume 3835 of *Lecture Notes in Computer Science*, pages 580–594. Springer, 2005.