

Towards the shortest DRAT proof of the Pigeonhole Principle

Isaac Groszof, Naifeng Zhang, and Marijn J.H. Heule

Carnegie Mellon University, Pittsburgh, Pennsylvania, United States
{igroszof,naifengz,marijn}@cmu.edu

Abstract

The Pigeonhole Principle (PHP) has been heavily studied in automated reasoning, both theoretically and in practice. Most solvers have exponential runtime and proof length, while some specialized techniques achieve polynomial runtime and proof length. Several decades ago, Cook manually constructed $O(n^4)$ extended resolution proofs, where n denotes the number of pigeons. Existing automated techniques only surpass Cook’s proofs in similar proof systems for large n . We construct the shortest known proofs of PHP in the standard proof format of modern SAT solving, DRAT. Using auxiliary variables and by recursively decomposing the original program into smaller sizes, we manually obtain proofs having length $O(n^3)$ and leading coefficient $5/2$.

1 Introduction

Many important SAT instances are known to be unsatisfiable, but are challenging to solve quickly. A natural way to compare different approaches is to compare proof length in a given proof system. This also allows us to directly compare automated and manual solving efforts. Finding shorter and shorter proofs of challenging UNSAT instances can serve as a guide for research into new techniques for efficient SAT solving. If a short proof exists, we can hope to eventually build a fast solver based on the same principle.

The Pigeonhole Principle (PHP), when phrased as a SAT instance, is famously difficult to prove, making it a good challenge problem. Haken [11] proved that any resolution proof of the Pigeonhole Principle must be exponential in size. Likewise, state-of-the-art SAT solvers such as `CaDiCaL` and `Kissat` [5] essentially only search for resolution proofs for PHP instances, thus requiring exponential time and proof size.

As a result, the Pigeonhole Principle has received extensive research focus over the years [1, 2, 4, 6, 9, 11, 16, 17], often employed as a way to evaluate stronger proof systems and new SAT solving tools. It has been known since the 1970s that shorter proofs of unsatisfiability exist, using more powerful proof systems. Cook [8] gave an $O(n^4)$ proof of unsatisfiability, using the extended resolution proof system [18]. His proof consists of n inductively defined PHP formulas, each of $O(n^3)$ size. However, the Pigeonhole Principle can be encoded more efficiently, using only $O(n^2)$ clauses. This gives hope for an $O(n^3)$ size proof along the lines of Cook’s proof. Of course, we must still use the standard inefficient encoding of the input Pigeonhole instance, only using the more efficient encoding for the inductively defined formulas.

More recently, a few specialized automated tools have been created which find shorter proofs of unsatisfiability, empirically scaling as $O(n^3)$, though with larger leading constants. Heule, Kiesel, and Biere [15] describe a novel extension-free proof system called “Propagation Redundancy” (PR), in which they find $O(n^3)$ length proofs for Pigeonhole Principle formulas. Heule and Biere [14] then give a method to convert these PR proofs into $O(n^3)$ length DRAT proofs. In a recently published paper, Bryant, Biere, and Heule [7] describe a solver based on Pseudo-Boolean binary decision diagrams (PGBDD), which outputs DRAT proofs of the Pigeonhole Problem whose clause length empirically scales as $O(n^3)$.

While these solvers’ proofs empirically scale as $O(n^3)$, the leading constant varies dramatically. The PGBDD-based solver’s proofs have a much larger leading constant than Cook’s $O(n^4)$ clause proof, so they are only shorter for $n \geq 128$. The PR-based solver, when its PR proofs converted to DRAT proofs, has a proof length of approximately $3.3n^3$ [14]. This represents the current state-of-the-art in short DRAT proofs for PHP, and it is the benchmark we will attempt to improve upon.

We compare these proofs by examining their proof lengths using the DRAT proof system [19], which has become the standard proof system in SAT solving. For instance, DRAT has been used as the proof system the SAT Competition since 2014 [3]. Proofs in the DRAT format can be automatically converted by the tool `dram-trim` [19] into the LRAT proof format [10], which can be efficiently checked by several formally verified proof checkers [12].

We therefore ask:

What is the shortest DRAT proof of unsatisfiability for the standard PHP(n) formula?

We give the shortest known DRAT proofs of unsatisfiability for PHP(n), both the shortest for concrete small n and for asymptotic n .

In this paper, we provide the following contributions:

- In Section 4, we give the shortest known DRAT proof of the unsatisfiability of PHP(n), and prove that it is a valid DRAT proof. We also give a reference to our implementation of our proof.
- In Section 5, we give an exact formula for the proof length of our proof, demonstrating that our proof length scales as $\frac{5}{2}n^3 + O(n^2)$.
- In Section 6, we empirically compare our proof, Cook’s original proof, and the proofs output by state-of-the-art CDCL-based solvers.

2 Background

2.1 CNF Formulas

We consider propositional formulas in *conjunctive normal form* (CNF), which are defined as follows. A *literal* is either a variable x (a *positive literal*) or the negation \bar{x} of a variable x (a *negative literal*). The *complement* \bar{l} of a literal l is defined as $\bar{l} = \bar{x}$ if $l = x$ and as $\bar{l} = x$ if $l = \bar{x}$. A *clause* is a finite disjunction of the form $(l_1 \vee \dots \vee l_n)$ where l_1, \dots, l_n are literals. A *formula* is a finite conjunction of the form $C_1 \wedge \dots \wedge C_m$ where C_1, \dots, C_m are clauses. For example, $(x \vee \bar{y}) \wedge (z) \wedge (\bar{x} \vee \bar{z})$ is a formula consisting of the clauses $(x \vee \bar{y})$, (z) , and $(\bar{x} \vee \bar{z})$. Formulas can be viewed as sets of clauses, and clauses can be viewed as sets of literals.

A *unit clause* is a clause that contains only one literal. The result of applying the *unit-clause rule* to a formula F is the removal of all clauses that are satisfied by unit clauses and the removal of all literals that are falsified by unit clauses. The iterated application of the unit-clause rule to a formula, until no unit clauses are left, is called *unit propagation*. If unit propagation on a formula F yields the empty clause \perp , we say that it derived a *conflict* or a *contradiction* on F . For example, unit propagation derives a conflict on $F = (\bar{x} \vee y) \wedge (\bar{y}) \wedge (x)$ since the unit clauses remove both literals in the first clause, thereby reducing it to \perp .

2.2 Background on DRAT

We use the DRAT [19] proof system, which has become the standard proof system in SAT solving. In this section, we provide some background on the DRAT proof system.

The DRAT proof system operates by starting with a CNF formula F . Each line of the proof is either a clause addition instruction, or a clause deletion instruction. We define a “working formula” F_i for every proof line i in the DRAT proof. The initial formula F_0 is simply the input formula F .

Given a working formula F_i after i proof lines, the next working formula F_{i+1} is either $F_i \cup \{C\}$, if some clause $C \notin F_i$ is added, or $F_i \setminus \{C'\}$, if some clause $C' \in F$ is deleted. A DRAT proof terminates with the addition of the empty clause, demonstrating a contradiction.

A clause C is valid to add or delete if it satisfies the *Resolution Asymmetric Tautology* (RAT) property for some literal $l \in C$ with respect to the current formula F_i .

A clause C has the RAT property for a literal $l \in C$ with respect to the formula F_i if all resolvents of C on l are implied by F_i via unit propagation. Specifically, consider all clauses $D \in F_i$ such that $\bar{l} \in D$. The resolvent $C \bowtie D$ is defined as:

$$C \bowtie D := (C \setminus l) \cup (D \setminus \bar{l})$$

C has the RAT property on literal l if and only if $F_i \vdash_1 C \bowtie D$ for all such D . That is, F_i implies $C \bowtie D$ via unit propagation. A formula F implies a clause C' via unit propagation if the conjunction of F with the negation of each literal $l' \in C'$ leads to a contradiction via unit propagation.

If a clause $C \notin F$ has the RAT property with respect to F , then F and $F \cup \{C\}$ are satisfiability equivalent [13].

As an example of the RAT property, let $F = (a \vee \bar{b}) \wedge (\bar{a} \vee b) \wedge (b \vee \bar{c}) \wedge (c)$. Let’s examine whether the clause $C = (a)$ has the RAT property with respect to F , on the literal a . The only clause in F which contains \bar{a} is $D = (\bar{a} \vee b)$. The resolvent is $C \bowtie D = (b)$. Now, we must check whether F implies (b) via unit propagation. To do so, we perform unit propagation on $F \cup \{\bar{b}\}$. We find that \bar{c} , followed by a contradiction. As a result, $F \vdash_1 (b)$, implying that C has the RAT property.

2.3 Background on the Pigeonhole Principle

The Pigeonhole Principle states that

It is impossible to put $n + 1$ pigeons in n holes, with at most one pigeon in each hole.

To phrase this as Boolean formula, we will have variables x_{ph} for each pigeon $p \in [0, n]$ and each hole $h \in [1, n]$, where x_{ph} represents whether pigeon p is in hole h . Next, we need to encode the constraints that “each pigeon is in at least one hole” and “each hole contains at most one pigeon”. The standard CNF encoding of this problem, given by Cook [8], encodes “each pigeon is in a hole” as a single clause for each pigeon:

$$(x_{p1} \vee \cdots \vee x_{pn}) \quad \forall 0 \leq p \leq n$$

To encode “each hole contains at most one pigeon”, the standard CNF encoding has a clause for each pair of variables corresponding to the same hole:

$$\bar{x}_{ph} \vee \bar{x}_{qh} \quad \text{for all pigeons } 0 \leq p < q \leq n, \text{ for all holes } 1 \leq h \leq n \quad (1)$$

To denote this standard encoding, we write $\text{PHP}(n)$. Note that the standard encoding uses $O(n^3)$ clauses.

Cook [8] gave a proof that the Pigeonhole Principle is unsatisfiable using the Extended Resolution proof system, using $O(n^4)$ clauses to do so. Cook’s proof proceeds by reducing $\text{PHP}(n)$ to $\text{PHP}(n - 1)$. Cook introduces new variables x'_{ph} for $0 \leq p < n$, $1 \leq h < n$ with definitions as follows:

$$x'_{ph} \leftrightarrow x_{ph} \vee (x_{nh} \wedge x_{pn}) \tag{2}$$

Cook then uses reduction to derive clauses identical to the standard encoding on the x'_{ph} variables, encoding an instance with one fewer pigeon and one fewer hole. Cook then recurses until $n = 1$, at which the instance can be immediately proven unsatisfiable with resolution. Cook’s proof uses $O(n^3)$ clauses in the recursive step from the n to $n - 1$ size instances, for a total of $O(n^4)$ clauses.

In this paper and in our linked generator, we manually construct the shortest known DRAT proofs of the unsatisfiability of the Pigeonhole Problem formula $\text{PHP}(n)$. Our proofs combine the $O(n^3)$ scaling of recent solvers [7, 15] with the immediate generation and small leading constant of Cook’s proof.

3 High-level Overview

Our goal is to give the shortest known DRAT proof of unsatisfiability for the Pigeonhole Principle formula, where proof length is measured by the number of clauses added.

We want to use the same recursive-variable-introduction style of proof that Cook used, introducing new formulas of smaller and smaller size. This is a very effective technique for a short proof, especially for small n , such as $n \leq 100$. Among all such proofs in this style, we want to give the shortest possible proof. Moreover, we want to do so without changing the initial encoding of the pigeonhole formula.

To generate a smaller proof, we change the encoding of the “at most one pigeon per hole” constraint in the recursively introduced formulas, without changing the initial encoding. Both the standard $\text{PHP}(n)$ formula and Cook’s proof use the pairwise encoding of this constraint. There are several more efficient encodings of the at most one constraint, but we choose the most efficient encoding (the smallest sum of the number of clauses plus the number of variables), by recursively removing three literals and adding an auxiliary variable. We describe this more in Section 4.1. As is, these clauses cannot be introduced via DRAT. To overcome this, we introduce a small number of auxiliary clauses – only one per auxiliary variable.

Other than the encoding of the “at most one pigeon per hole” constraint, there is little else to improve, with respect to our goal of minimizing the number of clauses added. The base variables are defined using four clauses each, which is the minimum for any definition other than a simple AND or OR of two literals. Such a simple definition does not seem sufficient for the new formula to have the appropriate structure. Finally, there are the “each pigeon is in a hole” constraints, which are a single clause per pigeon. Other than the small number of auxiliary clauses, there seems to be little room for improvement.

One possibility for further improvement that we did not explore would be to remove multiple pigeons in a single step of variable introduction. It is possible that this could shorten the proof further. Another possibility is that for very large n , another proof style might be shorter, as one of our style’s advantages is its simplicity, which is primarily beneficial at small n , e.g. $6 \leq n \leq 100$. We leave these questions to future research. We believe that our proof is the best

possible proof of unsatisfiability for PHP that uses the recursive-variable-introduction style and removes one pigeon at a time.

4 Our proof of the unsatisfiability of the Pigeonhole Principle formula

In this section, we specify the clauses we add, as well as why adding each clause is a valid RAT step. We describe our proof here in words, but we have also implemented our proof in code. Our proofs are output by the proof generator¹. To generate our proof of PHP(n), run `python3 prove-amo-general.py <n> 3`, where $\langle n \rangle$ is the desired problem size. This outputs our proof in the DRAT format. While our proof generator outputs clause deletions, the deletions are not necessary for our DRAT proof, and we do not include them in our proof length. They are merely added to speed up verification.

For convenience, we have also provided a generator of the standard encoding of the Pigeonhole Problem². To generate the problem, run `python3 generate.py <n>`. Note that this initial encoding of the Pigeonhole Problem uses the same standard encoding as Cook’s proof, making our proof directly comparable to Cook’s proof.

To verify our proof, use a tool such as `drat-trim`³ [19].

4.1 Detailed overview of our solution

Recall from (1) that to encode the constraint that at most one pigeon is in hole h , Cook uses a direct encoding that for each hole h , for all p, q pigeons, $\bar{x}_{ph} \vee \bar{x}_{qh}$, which has a size of $O(n^3)$ clauses. Let us denote this constraint as $\text{AMO}(x_{0h}, x_{1h}, \dots, x_{nh})$, where AMO stands for “At Most One.”

In our encoding, we recursively decompose the AMO constraint by removing the first three literals, namely x_{0h}, x_{1h}, x_{2h} , and add an auxiliary variable y_{0h} . We define y_{0h} to hold if none of the first three literals hold:

$$y_{0h} \leftrightarrow \bar{x}_{0h} \wedge \bar{x}_{1h} \wedge \bar{x}_{2h}$$

We also add pairwise constraints to ensure no pair of the first three literals hold. Finally, we add \bar{y}_{0h} to the rest of literals to form another AMO constraint to be recursively decomposed.

Therefore, for each hole h , $\text{AMO}(x_{0h}, x_{1h}, \dots, x_{nh})$ can be decomposed into

$$\begin{aligned} & \text{AMO}(\bar{y}_{0h}, x_{3h}, x_{4h}, \dots, x_{nh}) \\ & \wedge (y_{0h} \vee x_{0h} \vee x_{1h} \vee x_{2h}) \\ & \wedge (\bar{y}_{0h} \vee \bar{x}_{0h}) \wedge (\bar{y}_{0h} \vee \bar{x}_{1h}) \wedge (\bar{y}_{0h} \vee \bar{x}_{2h}) \\ & \wedge (\bar{x}_{0h} \vee \bar{x}_{1h}) \wedge (\bar{x}_{0h} \vee \bar{x}_{2h}) \wedge (\bar{x}_{1h} \vee \bar{x}_{2h}) \end{aligned} \tag{3}$$

Note that (3) is not necessary to represent the constraint that at most one of $x_{0h}, x_{1h}, \dots, x_{nh}$ is true. Instead, by adding the auxiliary clause (3), we constrain the variables $x_{0h}, x_{1h}, x_{2h}, y_{0h}$ to require that exactly one is true. With this additional constraint we ensure that unit propagation will function smoothly, which is key to short DRAT proofs. If we

¹<https://github.com/isaacg1/pigeonhole/blob/main/prove-amo-general.py>

²<https://github.com/isaacg1/pigeonhole/blob/main/generate.py>

³<https://github.com/marijnheule/drat-trim>

had an “at most one” constraint instead of an “exactly one” constraint, the proof would require case analysis, preventing us from efficiently adding DRAT clauses.

We recursively decompose the latter part, $AMO(\bar{y}_{0h}, x_{3h}, x_{4h}, \dots, x_{nh})$, following the same procedure. In the end, our encoding has a size of $O(n^2)$ clauses, $O(n)$ clauses per hole h . A precise count is given in Section 5.

To reduce from $\text{PHP}(n)$ to $\text{PHP}(n-1)$, we remove pigeon n and hole n , introducing new variables x'_{ph} according to the same definition that Cook [8] used, given in (2). In the same fashion as in Cook’s proof, we repeat this reduction n times until the proof is trivial.

Specifically, our proof consists of the following steps:

- Iterate by k from $n-1$ down to 1,
 - Definitions:
 - * Introduce the x_{ph} variables, for $1 \leq h \leq k, 0 \leq p \leq k$.
 - * Introduce the y_{gh} variables, for $1 \leq h \leq k, 0 \leq g \leq \lfloor k/2 \rfloor$.
 These clauses are extended resolution clauses, a less-powerful subset of RAT.
 - Derivations: Prove pairwise constraints between variables in group g other than y_{gh} . These clauses use the full power of RAT.
 - Finally, for each pigeon $p \in [0, k]$, we derive the “at least one” constraints which specify that each pigeon being in a hole. These clauses are RUP clauses, a less-powerful subset of RAT [19].

Throughout this proof, whenever we write a DRAT clause, we write the resolution variable first. We also **bold** the resolution variable.

In Sections 4.2 to 4.4, we describe the process of variable introduction for the general recursive step where $k < n-1$, in which both the prior formula and the newly introduced formula use our novel encoding. For the case of $k = n-1$, where the prior encoding is the standard encoding $\text{PHP}(n)$ given in (1), the same DRAT introduction clauses suffice, though the proof is significantly simpler.

4.2 Definition Clauses

4.2.1 Introducing Base Variables

Our first kind of clauses introduce the new x'_{ph} variables.

These clauses are defined in the same fashion as in Cook’s proof (2). There are two ways for pigeon p to be in hole h on iteration k :

- If pigeon p was in hole h on iteration $k+1$.
- If pigeon p was in hole $k+1$ on iteration $k+1$ and pigeon $k+1$ was in hole h on iteration $k+1$.

Intuitively, we are deleting pigeon $k+1$ and hole $k+1$, and moving the pigeon from the deleted hole to the opening created by the deleted pigeon. Our proof is flexible enough to allow any pigeon to be deleted – we delete the $k+1$ th pigeon for simplicity of indexing and to mirror Cook’s proof.

Symbolically, we define x'_{ph} as follows:

$$x'_{ph} \leftrightarrow x_{ph} \vee (x_{(k+1)h} \wedge x_{p(k+1)})$$

We implement this definition with the following 4 clauses. In each case, the redundant variable for DRAT purposes is x'_{ph} , the newly introduced variable.

$$\bar{x}'_{ph} \vee x_{ph} \vee x_{p(k+1)} \quad (4)$$

$$\bar{x}'_{ph} \vee x_{ph} \vee x_{(k+1)h} \quad (5)$$

$$x'_{ph} \vee \bar{x}_{ph} \quad (6)$$

$$x'_{ph} \vee \bar{x}_{p(k+1)} \vee \bar{x}_{(k+1)h} \quad (7)$$

One minor optimization that we have made is that if $p = k$, we omit (4) and (5), the two clauses above in which \bar{x}'_{ph} appears. Intuitively, these clauses only prevent pigeons from being added from nowhere in iteration k , which does not harm the proof. They are needed for unit propagation steps in RAT proofs when $p < k$, but because our propagation proceeds towards lower ps , these clauses are unnecessary when $p = k$, the maximum possible value of p .

4.2.2 Introducing Auxiliary Variables

Having introduced the x'_{ph} variables, we now introduce the auxiliary variables y'_{gh} . Each such variable corresponds to a group of x'_{ph} variables. There are 3 types of groups:

- the initial group, group 0
- intermediate groups
- and the final group, group $\lfloor n/2 \rfloor - 1$

Group 0 consists of

$$x'_{0h}, x'_{1h}, x'_{2h}, y'_{0h} \quad (8)$$

Intermediate groups $g \in \{1, 2, \dots, \lfloor n/2 \rfloor - 2\}$ consist of

$$\bar{y}'_{(g-1)h}, x'_{(2g+1)h}, x'_{(2g+2)h}, y'_{gh}$$

The final group consists of $\bar{y}'_{(\lfloor n/2 \rfloor - 2)h}$ and the remaining x' variables. If $n \leq 3$, there is only one group, consisting only of x' variables, which we think of as a final group.

For each group which is not a final group, we have 7 clauses. Of these, 4 are involved in introducing y'_{gh} . Let l_1, l_2, l_3 be the three literals in the group. Symbolically, we define y'_{gh} as

$$y'_{gh} \leftrightarrow \bar{l}_1 \wedge \bar{l}_2 \wedge \bar{l}_3$$

To introduce y'_{gh} , we add the clauses

$$y'_{gh} \vee l_1 \vee l_2 \vee l_3 \quad (9)$$

$$\bar{y}'_{gh} \vee \bar{l}_1$$

$$\bar{y}'_{gh} \vee \bar{l}_2$$

$$\bar{y}'_{gh} \vee \bar{l}_3$$

Note that (9) is not logically required to encode the “at most one pigeon per hole” constraint, but it is necessary to prevent a case analysis that would disrupt the RAT clause addition.

4.3 Derived clauses

In this section, we handle the derived clauses within each group, which are not simply definitions. We describe this for the general case where $k < n - 1$. When $k = n - 1$, we add the same DRAT clauses, but the prior encoding is the standard encoding, so the proof that the added DRAT clauses are valid is slightly different, but similar and simpler.

The three remaining clauses relating to group g are:

$$\overline{x'}_{ph} \vee y'_{(g-1)h} \tag{10}$$

$$\overline{x'}_{(p+1)h} \vee y'_{(g-1)h} \tag{11}$$

$$\overline{x'}_{(p+1)h} \vee \overline{x'}_{ph} \tag{12}$$

where $p = 2g + 1$. Note that for the initial group $g = 0$, $y'_{(g-1)h}$ is replaced by x'_{0h} .

Intuitively, these clauses are valid to add because if we assume their negations, we can perform unit propagation through variables in iteration $k + 1$ to conclude that no other pigeons were present in iteration $k + 1$, then move to iteration k and unit propagate back up to pigeon p , eventually reaching a contradiction.

To see why these hold in more detail, let us look at their resolvents. Let us consider the first clause, involving x'_{ph} , first. The only clauses where x'_{ph} has appeared positively are in the introduction of x'_{ph} , specifically (6) and (7), and the four-literal clause from the introduction of y'_{gh} , (9).

Let us restate these clauses:

$$\begin{aligned} x'_{ph} \vee \overline{x}_{ph} \\ x'_{ph} \vee \overline{x}_{p(k+1)} \vee \overline{x}_{(k+1)h} \\ y'_{gh} \vee \overline{y'}_{(g-1)h} \vee x'_{ph} \vee x'_{(p+1)h} \end{aligned} \tag{13}$$

The resolvent of (13) and (10) contains $y'_{(g-1)h}$ and $\overline{y'}_{(g-1)h}$, so it is an immediate tautology.

We therefore must show that the remaining two clauses, after resolving with (10), are implied by unit propagation:

$$y'_{(g-1)h} \vee \overline{x}_{ph} \tag{14}$$

$$y'_{(g-1)h} \vee \overline{x}_{p(k+1)} \vee \overline{x}_{(k+1)h} \tag{15}$$

For example, let's look at (14). To prove it via unit propagation, we want to show that assuming $\overline{y'}_{(g-1)h} \wedge x_{ph}$ yields a contradiction via unit propagation.

Let us focus on the assumption that x_{ph} holds. Using unit propagation on the clauses introduced in iteration $k + 1$, we can conclude that \overline{x}_{qh} for all $q \neq p$.

Next, using the clauses that introduced the definitions of x'_{qh} in iteration k , we can conclude that $\overline{x'}_{qh}$ for all $q < k$.

Now, we can conclude that y'_{0h} , using the four-literal clause (9) from group 0. We can then derive y'_{1h} from the four-literal clause (9) from group 1, and so on, deriving that y'_{fh} for all $f < g$.

In particular, we conclude that $y'_{(g-1)h}$, using the four-literal clause (9) from group $g - 1$. This produces the desired contradiction.

We can derive a similar contradiction to prove the second desired clause. From the assumption that $x_{p(k+1)}$, we use unit propagation to prove that $\overline{x}_{q(k+1)}$ for all $q \neq p$. From the assumption that $x_{(k+1)h}$, we use unit propagation to prove that \overline{x}_{qh} for all $q < k + 1$.

Now, we can once again conclude that

$$\begin{aligned}\bar{x}'_{qh} & \quad \forall q < p \\ y'_{fh} & \quad \forall f < g\end{aligned}$$

We thereby reach the same contradiction.

We have now given a RAT proof that it is valid to add (10) at this point in the proof. The RAT proofs for the other two clauses, (11) and (12), are essentially identical.

4.4 Deriving “each pigeon is in a hole” clauses

Finally, for each pigeon p , we add the clause

$$x'_{p1} \vee x'_{p2} \vee \dots \vee x'_{pk}.$$

This clause C satisfies the RUP condition, which means that the current formula F implies C itself via unit propagation, without needing to look at any resolvents. To see why, assume that \bar{x}'_{ph} for all $h \in [1, k]$. We previously introduced clauses of the form

$$x'_{ph} \vee \bar{x}_{ph}$$

From unit propagation, we find that \bar{x}_{ph} for all $h \in [1, k]$.

From iteration $k + 1$, we have a clause which says that

$$x_{p1} \vee \dots \vee x_{pk} \vee x_{p(k+1)}$$

As a result, $x_{p(k+1)}$ must hold. But we also previously introduced clauses of the form

$$x'_{ph} \vee \bar{x}_{p(k+1)} \vee \bar{x}_{(k+1)h}$$

We can therefore conclude that $\bar{x}_{(k+1)h}$ must hold, for all $h \in [1, k]$. We therefore conclude that $x_{(k+1)(k+1)}$, by parallel reasoning. Now, we have concluded that both $x_{p(k+1)}$ and $x_{(k+1)(k+1)}$. This is two pigeons in the same hole, so the rest is straightforward. Specifically, we can conclude that the $y_{g(k+1)}$ variable for the group g containing pigeon p must hold, and hence none of the $y_{g'(k+1)}$ for $g' > g$ can hold eventually reaching a contradiction on the final group, which contains pigeon $k + 1$. As a result, this clause is valid to add.

5 Counting Clauses

In this section, we count the exact number of DRAT clauses used in our proof, as well as Cook’s proof [8] as implemented into DRAT by Randy Bryant’s generator⁴.

5.1 Our proof

For a given iteration k , three types of clauses are added.

For each $p \in [0, k]$, $h \in [1, k]$, we have a newly defined variable x_{ph} . To define the variable, four clauses are added. However, for defining the final pigeon $p = k$, we only use two clauses. There are a total of $(4k + 2)k$ such clauses.

⁴<https://github.com/rebryant/pgbdd/blob/master/benchmarks/pigeon-cook.py>

Next, we have the group clauses, which encode the “at most one pigeon per hole” constraint. There are $\lfloor \frac{k}{2} \rfloor$ groups for the $k + 1$ pigeons in a given hole. The first and last groups have up to 3 “ x ” variables, while all intermediate groups have 2. For each group that is not the final group, we have 7 clauses. For the final group, we have 3 clauses if $k \equiv 1 \pmod{2}$, and 6 clauses if $k \equiv 0 \pmod{2}$. Let $f(k)$ be the number of group clauses per hole, which has value:

$$f(k) = \left\lfloor \frac{7}{2}k \right\rfloor - 4$$

As an exception, $f(1) = 1$. In each case, the number of group clauses is $kf(k)$.

Finally, we have an “at least one clause” for each pigeon. There are $k + 1$ such clauses.

In total, the number of clauses added in iteration k , for $k > 1$ is

$$k(4k + 2) + k \left(\left\lfloor \frac{7}{2}k \right\rfloor - 4 \right) + k + 1$$

Such iterations are performed for k from $n - 1$ down to 2, with 9 clauses for $k = 1$ and 1 empty clause to complete the proof.

We can therefore calculate the exact number of clauses used by our proof, for all $n > 1$:

$$\begin{aligned} & \frac{5}{2}n^3 - \frac{35}{8}n^2 + \frac{11}{4}n + 2 \text{ if } n \equiv 0 \pmod{2} \\ & \frac{5}{2}n^3 - \frac{35}{8}n^2 + 3n + \frac{15}{8} \text{ if } n \equiv 1 \pmod{2} \end{aligned}$$

5.2 Cook’s proof

For a given iteration k , three types of clauses are added.

For each $p \in [0, k]$, $h \in [1, k]$, we have a newly defined variable x_{ph} . To define the variable, four clauses are added. There are a total of $4(k + 1)k$ such clauses.

For each pair of new variables x_{ph}, x_{qh} , such that $0 \leq p < q \leq k$, $1 \leq h \leq k$, 2 clauses are added. There are $(k + 1)k^2$ such clauses.

Finally, for each pigeon $p \in [0, k]$, there is an “at least one” constraint. There are $k + 1$ such clauses.

In total, the number of clauses added in iteration k is

$$4(k + 1)k + (k + 1)k^2 + k + 1 = k^3 + 5k^2 + 5k + 1$$

Such iterations are performed for each k from $n - 1$ down to 1, and one final empty clause is added to complete the proof.

The exact number of clauses used by Cook’s proof is:

$$\sum_{k=0}^{n-1} k^3 + 5k^2 + 5k + 1 = \frac{1}{4}n^4 + \frac{7}{6}n^3 + \frac{1}{4}n^2 - \frac{2}{3}n$$

6 Empirical Results

We verified our proof using the `drat-trim` tool [19] for n up to 71, at which point each verification took more than 10 minutes.

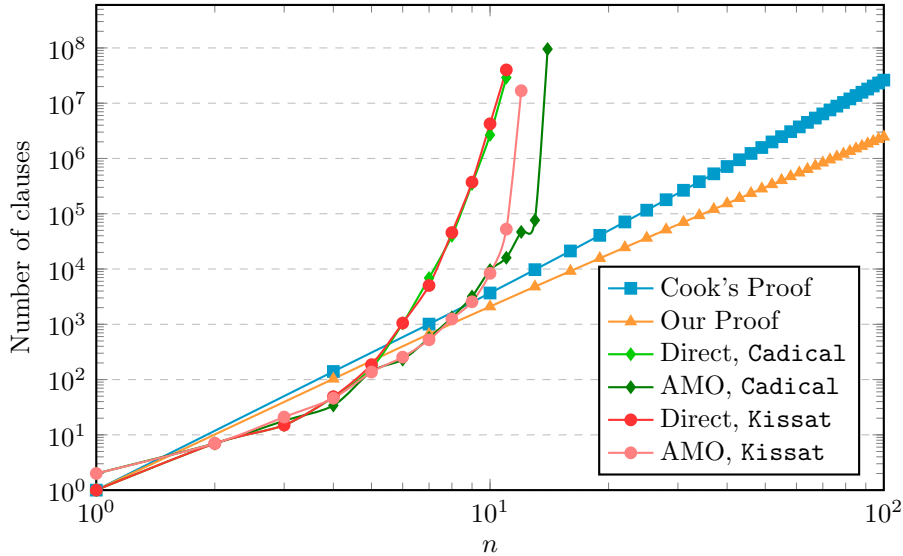


Figure 1: Number of clauses in proofs of pigeonhole problem for n holes.

We compared the length of our proof against Cook’s proof and two state-of-the-art CDCL-based SAT solvers, `CaDiCaL`⁵ and `Kissat`⁶. To each solver, we input both Cook’s direct encoding (the standard encoding) as well as our recursive AMO encoding. We generated Cook’s proof using Randy Bryant’s generator⁷. We measured the number of clauses generated by each approach for proofs from PHP(1) to PHP(100) for Cook’s proof and our proof. For the solvers, we ran all n for which the solver completed in at most 10 minutes.

The number of clauses in the proofs output by `CaDiCaL` and `Kissat` grow exponentially with the instance size n . This can be seen in Fig 1, as the curves are growing superlinearly on the log-log plot. While the proofs are shorter when the AMO encoding is given as input, they still grow in length exponentially, and become far longer than our proof and Cook’s proof once n is greater than 10.

Both our proof and Cook’s proof form straight lines in Fig 1, showing that the number of clauses grows polynomially with n . The clause counts exactly match our formulas from Section 5.

For all $n > 1$, our proof is shorter than Cook’s proof. In the asymptotic limit, the ratio between Cook’s proof length and our proof length converges to $n/10$. This follows from the fact that our leading term is $\frac{5}{2}n^3$, while Cook’s leading term is $\frac{1}{4}n^4$. For example, for $n = 100$, our proof adds 2,456,527 clauses, while Cook’s proof adds 26,169,100 clauses. Our proof is 10.65 times shorter than Cook’s proof for $n = 100$.

7 Optimization for a small number of pigeons

In Fig. 1 in Section 6, we showed that our proof is the shortest known proof for n larger than 7. For inputs $n \leq 7$, using `CaDiCaL` directly, combined with `drat-trim`’s `-O` optimization flag,

⁵<https://github.com/arminbiere/cadical>

⁶<https://github.com/arminbiere/kissat>

⁷<https://github.com/rebryant/pgbdd/blob/master/benchmarks/pigeon-cook.py>

gives the shortest known proof.

This allows an opportunity to shave a few lines off our proof for $n \geq 7$. We use our manual proof to reduce from $\text{PHP}(n)$ to our novel encoding for $k = n - 1$, $k = n - 2$, and so on. However, instead of recursing all the way to $k = 1$, we stop early after $k = 8$, and switch over to a hardcoded proof.

We have implemented this optimization using the flag `--optimized` in our proof generator, for inputs $n > 7$. Using this flag saves 229 steps from our proof, coming even closer to the shortest proof possible.

8 Conclusion

We give the shortest known DRAT proof of the Pigeonhole Principle formula $\text{PHP}(n)$. Our proof size scales as $O(n^3)$, with a leading constant of $\frac{5}{2}$, smaller than any prior proof. Our proof is asymptotically shorter than Cook’s proof [8], the best previously known manually constructed proof valid for all n . Specifically, our proof is shorter by a factor of $\frac{n}{10}$ for large n .

While our proof is the shortest DRAT proof known, there is still room for improvement, giving rise to several open problems. Can one achieve a smaller leading asymptotic term than $\frac{5}{2}n^3$? How much smaller can a proof be for specific n ?

Another important direction is exploring other proof systems or cost models. Our proof makes use of the full power of the RAT proof system. If the proof was restricted to Extended Resolution, could one adapt our proof to still give a $O(n^3)$ proof? Many of our proof steps have $O(n)$ length unit propagation sequences. What if the proof system was changed from DRAT to LRAT, and every clause hint counted as part of the proof size? Could an $O(n^3)$ proof still be achieved?

References

- [1] Gilles Audemard, George Katsirelos, and Laurent Simon. A restriction of extended resolution for clause learning sat solvers. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.
- [2] Paul Beame and Toniann Pitassi. Simplified and improved resolution lower bounds. In *Proceedings of 37th Conference on Foundations of Computer Science*, pages 274–282. IEEE, 1996.
- [3] Anton Belov, Daniel Diepold, Marijn J.H. Heule, and Matti Järvisalo, editors. *Proceedings of SAT Competition 2014*. University of Helsinki, 2014.
- [4] Armin Biere. Picosat essentials. *Journal on Satisfiability, Boolean Modeling and Computation*, 4(2-4):75–97, 2008.
- [5] Armin Biere, Katalin Fazekas, Mathias Fleury, and Maximillian Heisinger. CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In Tomas Balyo, Nils Froleyks, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors, *Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions*, volume B-2020-1 of *Department of Computer Science Report Series B*, pages 51–53. University of Helsinki, 2020.

- [6] Armin Biere, Daniel Le Berre, Emmanuel Lonca, and Norbert Manthey. Detecting cardinality constraints in cnf. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 285–301. Springer, 2014.
- [7] Randal E. Bryant, Armin Biere, and Marijn J. H. Heule. Clausal proofs for pseudo-boolean reasoning. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2022.
- [8] Stephen A Cook. A short proof of the pigeon hole principle using extended resolution. *Acm Sigact News*, 8(4):28–32, 1976.
- [9] Stephen A Cook and Robert A Reckhow. The relative efficiency of propositional proof systems. *The journal of symbolic logic*, 44(1):36–50, 1979.
- [10] Luís Cruz-Filipe, Marijn JH Heule, Warren A Hunt, Matt Kaufmann, and Peter Schneider-Kamp. Efficient certified rat verification. In *International Conference on Automated Deduction*, pages 220–236. Springer, 2017.
- [11] Armin Haken. The intractability of resolution. *Theoretical Computer Science*, 39:297–308, 1985. Third Conference on Foundations of Software Technology and Theoretical Computer Science.
- [12] Marijn Heule, Warren Hunt, Matt Kaufmann, and Nathan Wetzler. Efficient, verified checking of propositional proofs. In *International Conference on Interactive Theorem Proving*, pages 269–284. Springer, 2017.
- [13] Marijn J. H. Heule, Warren A. Hunt, and Nathan Wetzler. Verifying refutations with extended resolution. In Maria Paola Bonacina, editor, *Automated Deduction – CADE-24*, pages 345–359, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.
- [14] Marijn JH Heule and Armin Biere. What a difference a variable makes. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 75–92. Springer, 2018.
- [15] Marijn JH Heule, Benjamin Kiesl, and Armin Biere. Short proofs without new variables. In *International Conference on Automated Deduction*, pages 130–147. Springer, 2017.
- [16] Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. On tackling the limits of resolution in sat solving. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 164–183. Springer, 2017.
- [17] Alexander A Razborov. Proof complexity of pigeonhole principles. In *International Conference on Developments in Language Theory*, pages 100–116. Springer, 2001.
- [18] Grigori S Tseitin. On the complexity of derivation in propositional calculus. In *Automation of reasoning*, pages 466–483. Springer, 1983.
- [19] Nathan Wetzler, Marijn JH Heule, and Warren A Hunt. DRAT-trim: Efficient checking and trimming using expressive clausal proofs. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 422–429. Springer, 2014.