Can Pigeonhole Principle Definitions Be Learned?*

Chad E. Brown and Mikoláš Janota

Czech Technical University in Prague, CIIRC, Czechia

1 Introduction

Pigeonhole principle problems provide challenging problems for SAT solvers. This is inherent since they lead to exponential resolution proofs [4], meaning that the resolution proof of the fact that $n+1$ pigeons cannot be distributed into n holes grows exponentially with n; exponential lower-bounds are also known for other variants [5].

On the other hand, Cook [2] showed how relatively short proofs can be constructed when using *extended resolution* [6], which allows introducing definitions. In essence, when attempting to prove Pigeonhole with $n+1$ pigeons and n holes, one makes definitions and proves lemmas that reduce the problem to Pigeonhole with n pigeons and $n-1$ holes. We describe code that can solve such problems following this approach. We then discuss the possibility that machine learning could be used to automatically suggest the definitions and lemmas.

2 Extended SAT Solving

We have implemented code that can read a SAT problem (a set of input clauses) and a sequence of commands to solve the SAT problem. There are three kinds of commands:

- **Definition:** Make a definition of a new atom q of the form $p \vee p'$ or $p \vee (p' \wedge p'')$ using existing atoms p, p' and p''. This adds new definition clauses to the current clause set.
- **Lemma:** Prove a new clause and then add the clause to the current clause set. The new clause is proven by calling a SAT solver.
- **Delete:** Delete a clause from the current clause set.

After each command has been processed there is a final call to a SAT solver with the current clause set. The system is similar to DRAT [7], but simpler and focusing on extended resolution. Thus, in total, a SAT solver is called $N+1$ times, where N is the number of Lemma commands. The two SAT solvers we use are CaDiCaL 1.3.1 [3] and CaDiCaL 2.1.3 [1].

The formulation of Pigeonhole with n pigeons (and $n-1$ holes) is as in Cook [2]. Let $P_{i,j}$ be the atom meaning Pigeon i is in Hole j, with $0 \leq i < n$ and $0 \leq j < n-1$. The input contains $(n-1)\binom{n}{2}$ 2-literal clauses ensuring that no two pigeons are in the same hole and n $(n-1)$-literal clauses ensuring each pigeon is in at least one hole. CaDiCaL 2.1.3 proves the 10 pigeon case in 3.2s, 11 in 19.6s and 12 in 5m29s. Perhaps surprisingly, CaDiCaL 1.3.1 performs better, proving the 10 pigeon case in 0.4s, 11 in 1.8s, 12 in 11.1s, 13 in 42.5s and 14 in 4m57s. Since neither can prove the case with 15 pigeons within 5 minutes, we take this as the easiest difficult example.

Following Cook's idea, we start with a sequence of definitions, followed by a sequence of lemmas corresponding to the problem with $n-1$ pigeons and $n-2$ holes. After the lemmas are

*The results were supported by the Ministry of Education, Youth and Sports within the dedicated program ERC CZ under the project POSTMAN no. LL1902.

proven, we delete the clauses before proving the lemmas and continue. At this point, we can again make definitions, prove lemmas, and delete previous clauses to again reduce the number of pigeons. Once there are at most 11 pigeons, we simply make a final call to the SAT solver to complete the proof.

Consider the case with 15 pigeons. We begin with the initial 1,485 clauses using atoms $P_{i,j}$. We follow this with 182 definitions: $Q_{i,j}$ is defined as $P_{i,j} \vee (P_{i,13} \wedge P_{14,j})$ with $0 \leq i < 14$ and $0 \leq j < 13$. The idea is that Q now gives an injection from the first 14 pigeons into the first 13 holes. We prove 14 lemmas for each i there is some j such that $Q_{i,j}$ holds. This could be followed by many lemmas ensuring Q never sends two pigeons to the same hole, but we obtain better results if we efficiently prove that if Q sends a pigeon before Pigeon i into Hole j, then it does not send Pigeon i to Hole j. To make this efficient we make 169 definitions allowing us to refer to atoms equivalent to $\bigvee_{i' \in \{0,\ldots,i-1\}} Q_{i',j}$. This is followed by 169 lemmas which essentially say

$$(\bigvee_{i' \in \{0,\ldots,i-1\}} Q_{i',j}) \Rightarrow \neg Q_{i,j}.$$

We can then delete the input clauses and continue. The resulting state essentially has reduced the problem from 15 pigeons to 14 pigeons. Accordingly, we repeat the process of making definitions, proving lemmas and deleting clauses to reduce to 13 pigeons. We continue this until we have reached 11 pigeons at which point we make the final call. In total, we make 1,114 definitions, prove 584 lemmas and delete 3,838 clauses. The entire proof (including all SAT solver calls) takes 3.3s. Using a similar process we can prove the case with 30 pigeons in just over 3 minutes.

3 Can These Recipes Be Learned?

Of course, we are essentially checking an explicitly given proof as we are including the definitions, lemmas and deletions as commands. It would be more exciting to use some machine learning techniques to generate and suggest these commands. In order to do this, the "suggester" would be given the current set of clauses to analyze. After analyzing the set of clauses it could suggest one of the following possibilities:

- Make a definition of the form $p \vee p'$ or $p \vee (p' \wedge p'')$ (also suggesting which p, p' and p'' to use).
- Prove a lemma (explicitly suggesting the lemma clause).
- Delete an existing clause.
- Make the final call to a SAT solver.

Note that while the case with 15 pigeons is only one problem, it would generate thousands of instances of training data. At each step in the proof above, there is the current set of clauses (which changes after each command) and there is the command which follows. Likewise, the case of 30 pigeons would lead to about 70,000 training instances: 15,219 for definitions, 7,809 for lemmas and 51,063 for deleting clauses. One slight issue is that there is only one positive case which corresponds to making the final call to a SAT solver, as this is only done once.

References

[1] Armin Biere, Tobias Faller, Katalin Fazekas, Mathias Fleury, Nils Froleyks, and Florian Pollitt. CaDiCaL 2.0. In Arie Gurfinkel and Vijay Ganesh, editors, *Computer Aided Verification - 36th International Conference, CAV 2024, Montreal, QC, Canada, July 24-27, 2024, Proceedings, Part I*, volume 14681 of *Lecture Notes in Computer Science*, pages 133–152. Springer, 2024.

[2] Stephen A Cook. A short proof of the pigeon hole principle using extended resolution. *ACM SIGACT News*, 8(4):28–32, 1976.

[3] Nils Froleyks and Armin Biere. Single clause assumption without activation literals to speed-up IC3. In Ruzica Piskac and Michael W. Whalen, editors, *Proceedings 21st International Conference on Formal Methods in Computer-Aided Design (FMCAD'21)*, pages 72–76. TU Wien Academic Press, 2021.

[4] Armin Haken. The intractability of resolution. *Theor. Comput. Sci.*, 39:297–308, 1985.

[5] Alexander A. Razborov. Proof complexity of pigeonhole principles. In *Developments in Language Theory*, page 100–116. Springer Berlin Heidelberg, 2002.

[6] G. S. Tseitin. On the complexity of derivations in the propositional calculus. *Studies in Constructive Mathematics and Mathematical Logic*, 1968.

[7] Nathan Wetzler, Marijn J. H. Heule, and Warren A. Hunt. DRAT-trim: Efficient checking and trimming using expressive clausal proofs. In Carsten Sinz and Uwe Egly, editors, *Theory and Applications of Satisfiability Testing – SAT 2014*, pages 422–429, Cham, 2014. Springer International Publishing.