Clausal Proofs for Pseudo-Boolean Reasoning Demonstration Artifact

Randal E. Bryant[1], Armin Biere[2], and Marijn J. H. Heule[1]

[1] Carnegie Mellon University, Pittsburgh, PA, United States
{Randy.Bryant, mheule}@cs.cmu.edu
[2] Albert-Ludwigs University, Freiburg, Germany
biere@cs.uni-freiburg.de

This artifact accompanies the paper "Clausal Proofs for Pseudo-Boolean Reasoning," to be published at TACAS 2022. The artifact includes the code for two Boolean satisfiability solvers: PGPBS, described in the paper, and PGBDD, described in our earlier paper [2]. In addition, the artifact includes the raw data from the experimental results section of the paper, as well as code and data to reproduce either (1) a representative subset, or (2) nearly all of the experimental results. When run, the original data and the reproduced data are compiled into the document `graphs.pdf` in the top-level directory. This file allows a direct comparison between the original and the reproduced results.

1 Description

The PGPBS solver combines pseudo-Boolean (PB) reasoning with binary decision diagrams (BDDs) to generate checkable clausal proofs of unsatisfiability. Included also are two programs that use heuristic methods to detect pseudo-Boolean constraints encoded in the input file:

XOR_EXTRACTOR detects exclusive-or and exclusive-nor parity constraints
CONSTRAINT_EXTRACTOR detects at-most-one, at-least-one, and exactly-one cardinality constraints.

The combination of the extractors and PGPBS provides a fully automated execution flow from a conjunctive normal form (CNF) Boolean formula, encoded in the standard DIMACS format, to a checkable proof, in the standard LRAT format. [4] An LRAT checker is included as part of the artifact.

The normal use of the artifact is to first replicate a subset of the experimental results and then to format this data, as well as the original experimental data into a set of graphs, saved as the file `graphs.pdf` in the top-level directory. A copy of these graphs and their interpretation is included in Appendix A of this document. The total execution time is around 10–15 minutes, depending on the performance of the machine. It can be done within the TACAS22 virtual environment.

It is also possible to recreate almost of all of the results in the paper, subject to the following limitations:

– We have not included the KISSAT SAT solver in our distribution. That prevents generating the results we used to compare our solvers to traditional CDCL solvers. The original data from KISSAT is included.

– For the two sets of Urquhart benchmarks, we used generators that we are not authorized to redistribute. Instead, we have included CNF files for most of these benchmarks, but not for the largest ones, since the file sizes are too large.

Reproducing all of the results requires multiple days running on a well-resourced machine (in terms of physical memory and disk space.) We recommend only attempting the reduced version.

2 Downloading and reproducing results

Both the solver program PGPBS and the two extractors are written entirely in Python. The LRAT checker is written in C. All programs have been tested in the TACAS virtual environment.

The following programs are required to use the artifact:

1. A Python interpreter. The default is to use the local installation of `python3`.
2. A C compiler, The default is to use `gcc`.
3. An installation of the `pdflatex` document preparation program, including an installation of the `tikz` graphics program.

The following sequence of commands will download and run the artifact:

```
> wget http://www.cs.cmu.edu/~bryant/download/pgpbs-artifact.zip
> unzip pgpbs-artifact.zip
> cd pgpbs-artifact
> make reproduce
```

The latter operation will cause the solvers to be run and the results compiled. While running, a lot of data will be printed, but for each benchmark, the lines `UNSAT` and `VERIFIED` should occur. The final summary data will be stored in file `graphs.pdf`. The file `README.txt` contains more documentation about options available with the `make` command.

3 Archival version

The artifact is available on GITHUB as:

```
> git clone https://github.com/rebryant/pgpbs-artifact
```

A Generated graphs

The attached graphs are extracted from the experimental results section of the submitted paper "Clausal Proofs from Pseudo-Boolean Reasoning." For each one, we show the complete data presented in the paper, as well as reduced data generated from the artifact. Although the reproduced results do not demonstrate the full scale of the experimental results, they demonstrate that our solvers far exceed the capabilities of KISSAT, a state-of-the-art CDCL solver.

There are three classes of problems considered. All of the formulas are unsatisfiable, and so the task is to generate a proof of unsatisfiability.

In particular, here is the significance of the reproduced results for each of the four figures in the paper:

Fig. 4 These are for a benchmark based on a family of formulas devised by Urquhart [6], designed to require resolution proofs of exponential size. There are two versions of these formulas, due to Simon and to Li, with Li's versions being more challenging. The formulas scale quadratically with the parameter m, with $m \geq 3$. KISSAT can handle some instances of Simon's benchmarks for $m \in \{3, 4\}$, but none of Li's benchmarks. PGPBS generates a proof of unsatisfiability for $m = 12$ in under 20 seconds.

Fig. 5 These are for the mutilated chessboard problem [1], concerning the tiling of an $n \times n$ chessboard with two corners removed. As the figure shows, KISSAT scales exponentially with n. Our earlier solver PGBDD scales as $O(n^3)$ with careful guidance. The new solver PGPBS can achieve this scaling with full automation. Both programs generate proofs for up to $n = 32$ in under 15 seconds.

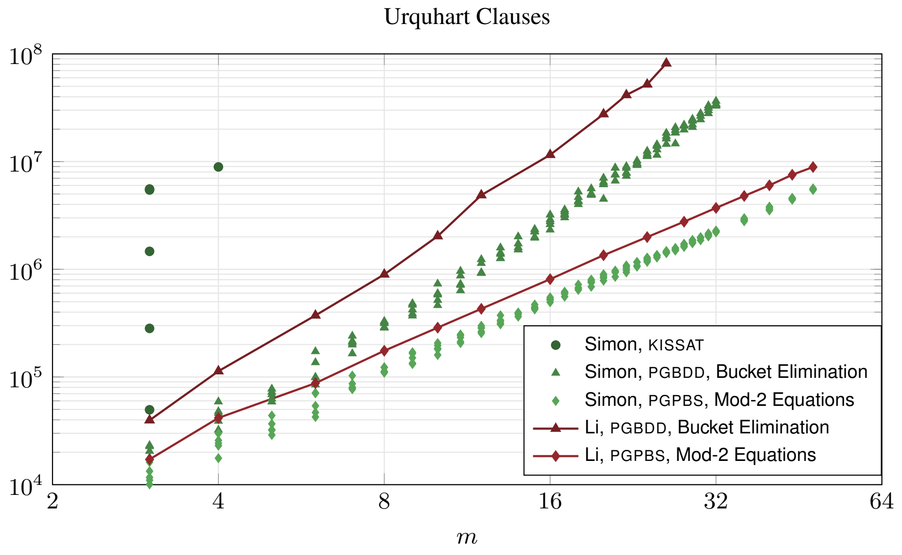
Fig. 6 These further demonstrate the advantages of PGPBS over PGBDD for chessboard tiling problems. PGBDD has exponential scaling when either 1) the input variables are ordered according to a random permutation, or 2) the chess board is converted to a torus by having the ends wrap around. PGPBS maintains the $O(n^3)$ scaling even with these variations. The ability to work efficiently with arbitrary variable orderings is unusual for BDD-based applications. The reproduced results go up to $n = 32$ for the variant problems.

Fig. 7 These are for the pigeonhole problem [5], concerning the impossible task of assigning of $n+1$ pigeons to n holes such that each hole contains at most one pigeon. The at-most-one constraints of pigeons to holes can be encoded either *directly* requiring $O(n^2)$ clauses per constraint, or via an encoding due to Sinz, requiring $O(n)$ clauses. KISSAT scales exponentially for either encoding, as does PGBDD for a direct encoding (using a tree of conjunctions to reduce BDD representations of the clauses to a single BDD). PGBDD can achieve $O(n^3)$ scaling on the Sinz encoding with careful guidance from the user. PGPBS can achieve $O(n^5)$ encoding with either encoding and with no guidance, or $O(n^3)$ performance when the at-most-one constraints are tightened to exactly-one constraints. Finally, Cook [3] showed an inductive approach to proving the unsatisfiability of the problem that scales as $O(n^4)$, but with a very low constant factor. The reproduced results range up to $n = 16$ for versions scaling as $O(n^5)$ and up to $n = 32$ for those achieving $O(n^3)$ or $O(n^4)$ scaling.

References

1. Alekhnovich, M.: Mutilated chessboard problem is exponentially hard for resolution. Theoretical Computer Science **310**(1-3), 513–525 (Jan 2004)
2. Bryant, R.E., Heule, M.J.H.: Generating extended resolution proofs with a BDD-based SAT solver. In: Tools and Algorithms for the Construction and Analysis of Systems (TACAS), Part I. LNCS, vol. 12651, pp. 76–93 (2021)
3. Cook, S.A.: A short proof of the pigeon hole principle using extended resolution. SIGACT News **8**(4), 28–32 (Oct 1976)
4. Cruz-Filipe, L., Heule, M.J.H., Hunt, W.A., Kaufmann, M., Schneider-Kamp, P.: Efficient certified RAT verification. In: Conference on Automated Deduction (CADE). LNCS, vol. 10395, pp. 220–236 (2017)
5. Haken, A.: The intractability of resolution. Theoretical Computer Science **39**, 297–308 (1985)
6. Urquhart, A.: Hard examples for resolution. J.ACM **34**(1), 209–219 (1987)

A) Original results



B) Reproduced results

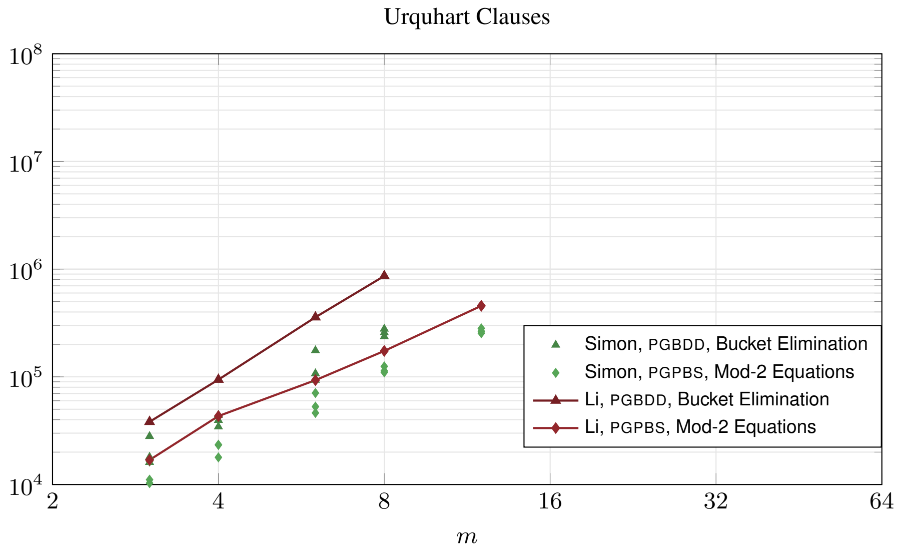
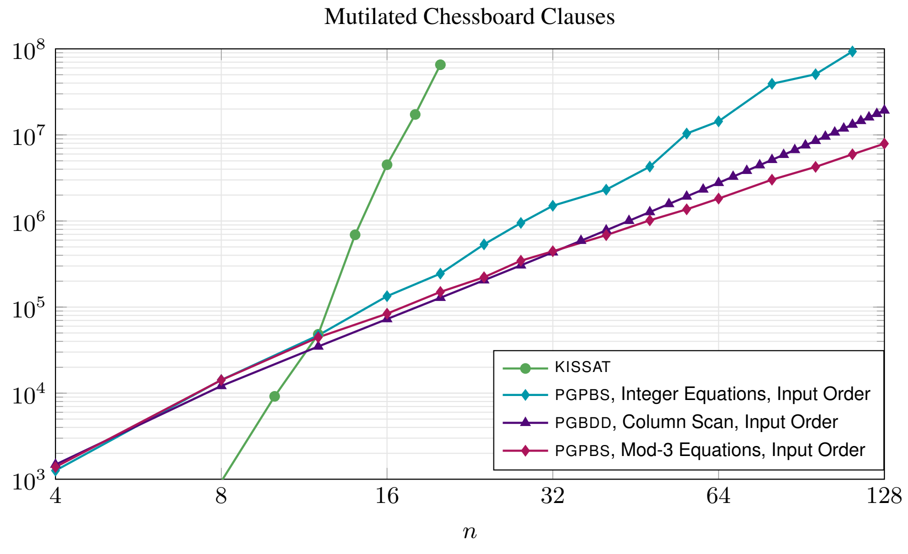


Fig. 4. Total number of clauses in proofs of two sets of Urquhart formulas.

A) Original results



B) Reproduced results

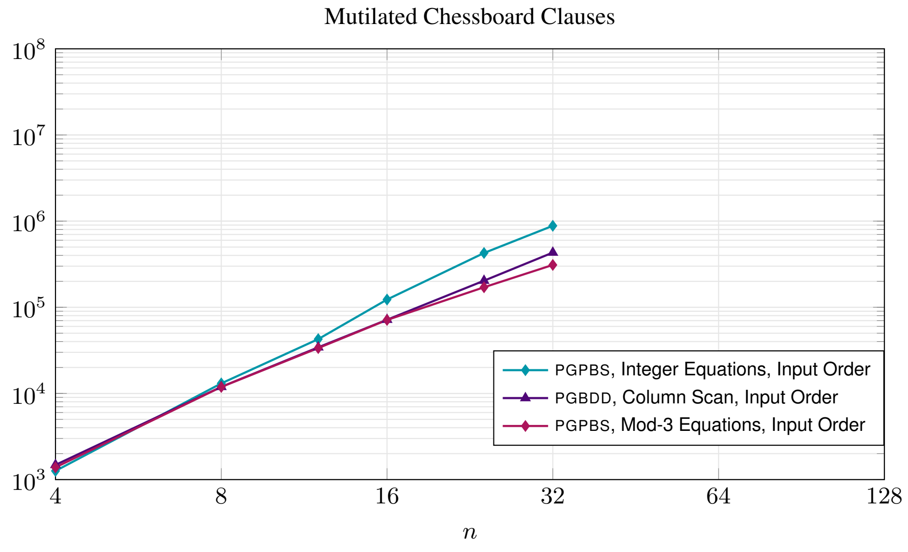
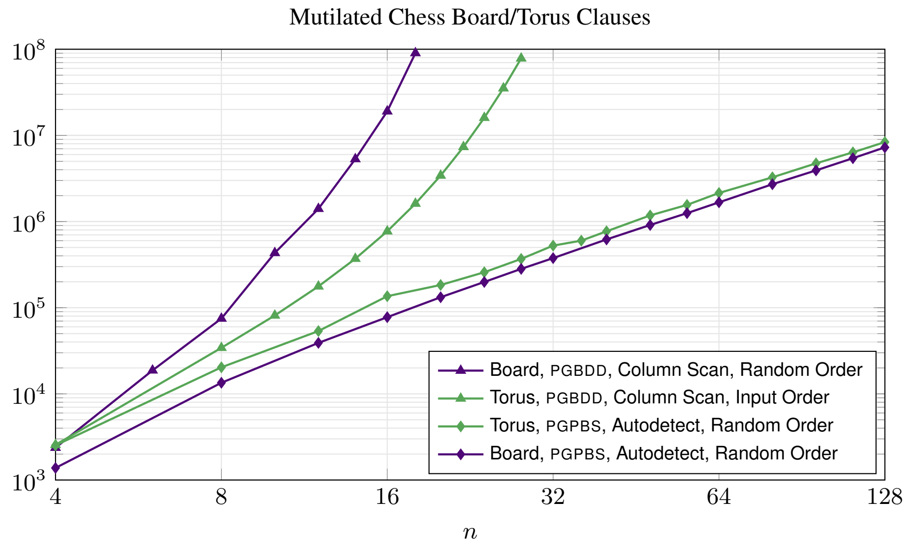


Fig. 5. Total number of clauses in proofs of $n \times n$ mutilated chess board problems.

A) Original results



B) Reproduced results

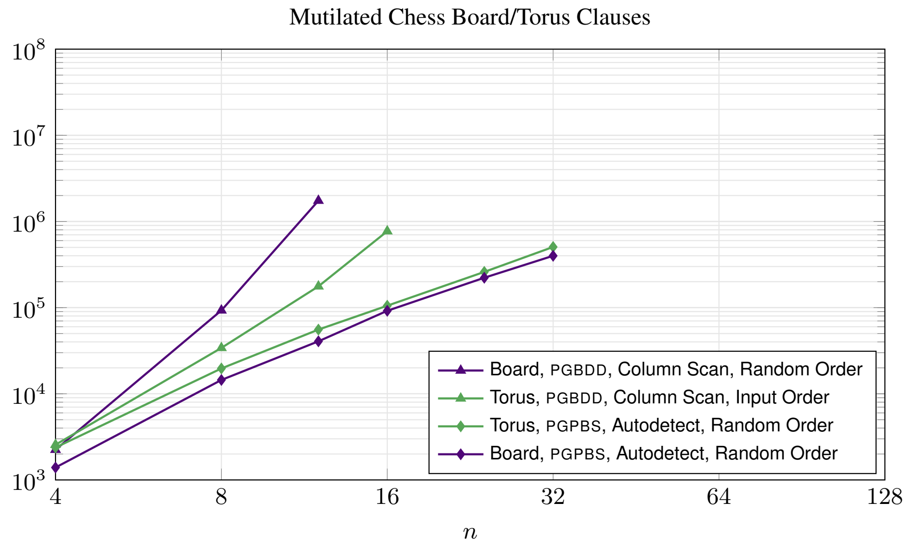
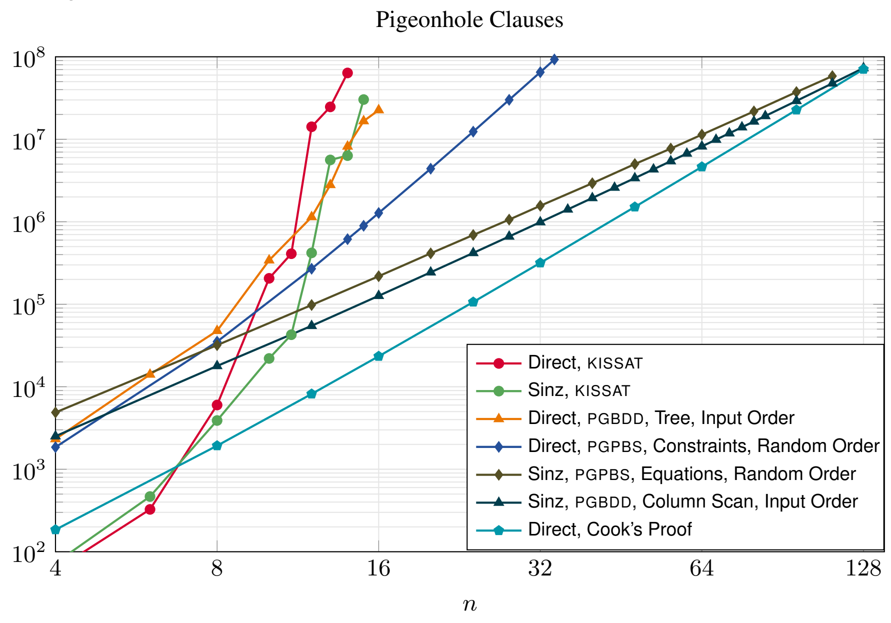


Fig. 6. Stress Testing: Changing the topology and variable ordering for mutilated chess. Autodetection enables the PB solver to use modulo-3 arithmetic.

A) Original results



B) Reproduced results

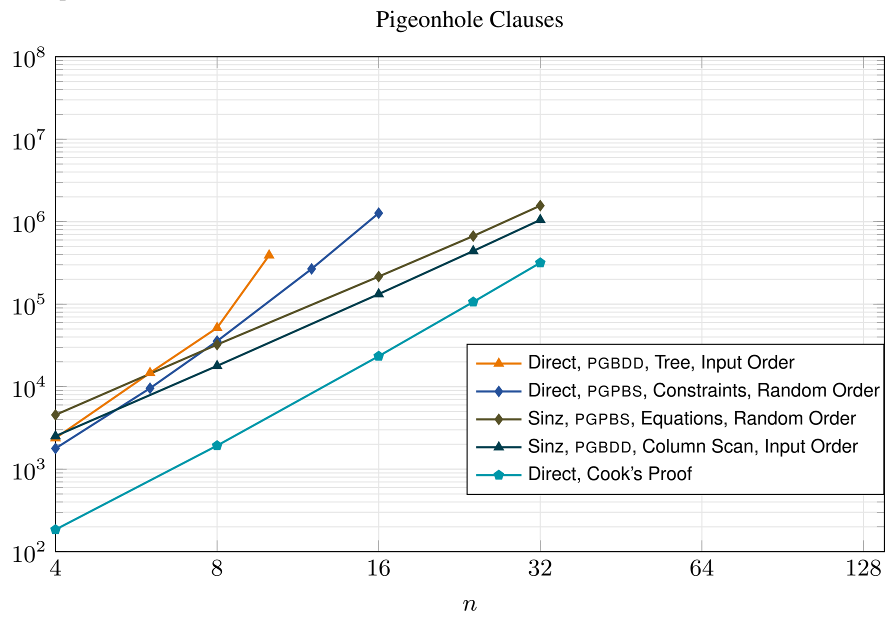


Fig. 7. Total number of clauses in proofs of pigeonhole problem for n holes