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 contains the Boolean satisfiability (SAT) solver PGPBS, as well as code that enables running it on a curated set of benchmarks. The solver combines pseudo-Boolean (PB) reasoning with binary decision diagrams (BDDs) to generate checkable clausal proofs of unsatifiability. Included also are two constraint extraction programs that use heuristic methods to detect either exclusive-or/nor constraints (XOR_EXTRACTOR) or cardinality constraints (CONSTRAINT_EXTRACTOR) encoded in the input file and provide this information to PGPBS. The combination of the extractors and PGPBS provide 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. [5] An LRAT checker is included as part of the artifact.

This artifact is submitted in conjunction with a research paper describing the solver and proof generator, as well as experimental data for which the artifact demonstrates a subset of the results. The artifact does not fully replicate the experimental results of the paper—that would requires hundreds of hours of execution time. Instead, it provides a set of ten unsatisfiable Boolean formulas that PGPBS can easily handle, each requiring less than 250 seconds to execute in the TACAS22 virtual environment and generating a proof with less than 1.5 million proof steps. Four of these can also be handled by our previous SAT solver PGBDD, [3] which is also included in the artifact, but six cannot. The earlier, BDD-based solver EBDDRES [8, 11] would also be able to handle the same four benchmarks as can PGBDD, but PGPBS generates shorter proofs in every case. Beyond these three programs, we know of no other proof-generating SAT solver that can handle *any* of these benchmarks. More documentation about the benchmarks is provided below.

Downloading and running the demonstration

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.

A demonstration of the program can be performed with the following sequence of commands:

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

The latter operation will cause the ten benchmarks to be run. 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 `results-pgpbs.txt`. The file `README.txt` contains more documentation about options available with the `make` command.

Interpreting the results

Table 1 shows the contents of the file `results-pgpbs.txt` when run on the TACAS22 virtual machine, excuting on a 2014 Apple Mac mini with a 3 GHz Intel Core i7 processor. As one might expect, the benchmarks run much faster when executing directly on a state-of-the-art machine. All columns, except the times, should be identical every time the benchmarks are run.

The benchmarks consist of ten files in two different categories, as is described below. As can be seen by the suffix "`shuf`" in the file names, all of the input files were shuffled with the SCRANFILIZE program [2] to randomize numbers assigned to the variables and the ordering of the clauses. This was done to eliminate any possible ability of the extractors or solver to exploit variable or clause ordering in the input file. It also has the effect of randomizing the variable orderings for the BDDs. PGPBS is unusual among BDD-based programs in being largely insensitive to variable ordering.

Besides the file names, the columns in the table indicate the following:

Input Vars. The number of variables in the input CNF file
Input Clauses The number of clauses in the input CNF file
Pseudo-Bool. Constraints The number of pseudo-Boolean constraints that were extracted from the input CNF file
Total Clauses The combined number of clauses in the input file and the proof file. Each proof step consists of a clause.
SAT Time (s) The elapsed time used by PGPBS. (The times for the extractors are not shown, but they typically require only a few seconds.)
Verif. Time (s) The elapsed time used by the LRAT proof checker.
Verif. Status The result of the proof checker, indicating that the generated proofs are all valid.

Benchmark problems

As can be seen in this table, there are two categories of benchmarks: *cardinality* problems concern matchings between different sets, and *parity* problems involving formulas implementing networks of exclusive-or operations. Both categories pose major difficulty for solvers based on conflict-driven clause-learning [9], the dominant approach for state-of-the-art SAT solvers. We evaluated smaller versions of the benchmarks with the 2020 version of KISSAT, winner of that year's SAT competition. We limited KISSAT

Benchmark File	Input Vars.	Input Clauses	Pseudo-Bool. Constraints	Total Clauses	SAT Time (s)	Verif. Time (s)	Verif. Status
cardinality/mchess-board-24_shuf.cnf	1100	3736	574	220000	31.00	0.96	VERIFIED
cardinality/mchess-torus-24_shuf.cnf	1100	3736	574	220000	31.22	1.00	VERIFIED
cardinality/pigeon-direct-16_shuf.cnf	272	2193	33	1425588	268.46	7.91	VERIFIED
cardinality/pigeon-sinz-16_shuf.cnf	528	769	33	1237868	237.13	7.17	VERIFIED
cardinality/randomG-B-mix-n19-d05_shuf.cnf	580	1216	39	97233	11.00	0.31	VERIFIED
cardinality/randomG-n20-d05_shuf.cnf	420	1051	41	511213	90.81	3.10	VERIFIED
parity/tseitingrid7x165_shuf.cnf	2310	9240	1155	410484	46.34	0.37	VERIFIED
parity/tseitingrid7x185_shuf.cnf	2590	10360	1295	460545	63.15	0.44	VERIFIED
parity/urquhart-li-03_shuf.cnf	153	408	102	17518	1.90	0.04	VERIFIED
parity/urquhart-li-12_shuf.cnf	2880	7680	1920	465129	64.00	0.48	VERIFIED

Table 1. Results when run on the TACAS22 virtual machine

to problems for which it could generate proofs with at most 100 million proof steps. That is well beyond the numbers seen in the table.

The cardinality benchmarks consist of six files covering three different categories:

- The mchess benchmarks encode variants of the *mutilated chessboard* problem, concerning an $N \times N$ board (in this case $N = 24$) with the corners on the upper left and the lower right removed. It attempts to tile the board with dominos, with each domino covering two squares. Since the two removed squares had the same color, and each domino covers one white and one black square, no tiling is possible. This problem has been well studied in the context of resolution proofs, for which it can be shown that any proof must be of exponential size [1].
  The two variants concern the topology of the board. The board variant has a standard square board. The torus variant has a board that wraps around on both the horizontal and the vertical boundaries. A non-shuffled version of the board variant could be handled by our earlier solver PGBDD, but it required careful guidance from the user. The torus variant, on the other hand, scales exponentially with N for PGBDD, even with user guidance. As Table 1 shows, PGPBS is largely unaffected by the topology. It can also operate with a random variable ordering and with no user guidance.
  KISSAT exceeds the 100-million step limit beyond $N = 20$. Its performance scales exponentially on this problem, and so reaching $N = 24$ would be much more challenging. As can be seen, PGPBS can handle this problem easily. In the paper, we show results for both variants up to $N = 128$.
- The pigeon benchmarks encode the *pigeonhole* problem, attempting to assign $N+1$ pigeons into N holes (in this case $N = 16$) such that 1) each pigeon is assigned to some hole and 2) each hole contains at most one pigeon. This is impossible, of course, but any resolution proof for this must be of exponential length [7]. Here we consider two different ways of encoding the at-most-one constraints for the holes. A direct encoding uses $N\,(N+1)/2$ clauses per hole, while the Sinz encoding [10] uses $3N-1$ clauses per hole, but it requires introducing $N-1$ additional variables. A non-shuffled version of the sinz variant could be handled by PGBDD with careful guidance from the user, but PGBDD scales exponentially for the direct variant. KISSAT also scales exponentially and is limited to $N = 14$ for the direct variant and $N = 15$ for the sinz variant. By contrast, PGPBS scales as $O(N^5)$. In the paper, we show results up to $N = 34$.
- The two randomG benchmarks are drawn from a set of benchmarks created by Codel et al. [4] and included in the 2021 SAT competition. None of the entrants could generate a proof of unsatisfiability for either benchmark within the 5000 second time limit. These benchmarks encode different perfect matching problems on bipartite graphs where one node partition is of size N (in these cases $N \in \{19, 20\}$), while the other is of size $N + 1$, and therefore no perfect matching is possible. As can be seen, PGPBS easily handles these problems.

The parity benchmarks consist of four files covering two different categories. As mentioned earlier, all of these can be handled by the programs PGPBS, PGBDD, and EBDDRES, but PGPBS will generate the shortest proofs. None of them can be handled by any other known solver.

– The `tseitingrid` benchmarks were created in 2016 by Elffers [6], as a category that would be challenging but scale polynomially. The two files included, defined over grid graphs of size 7×165 and 7×185, were included in the 2020 SAT competition. None of the entrants could generate proofs for either benchmark within the 5000 second time limit. Both can easily be handled by PGPBS.
– The `urquhart-li` benchmarks were generated by a program written by Li implementing a class of formulas devised by Urquhart [12]. These formulas are designed to be very compact and yet require very long resolution proofs. They are defined over a class of graphs having $2M^2$ nodes (in this case, $M \in \{3, 12\}$). KISSAT cannot handle even the smallest of these. PGPBS requires 1–2 seconds for $M = 3$ and readily handles the other one, as well. In the paper, we show results up to $M = 48$.

Archival version

The artifact is also available on GITHUB as:

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

We intend to maintain this repository and continue to make it publicly available. If accepted, we will also store a version on Zenodo.

References

1. Alekhnovich, M.: Mutilated chessboard problem is exponentially hard for resolution. Theoretical Computer Science **310**(1-3), 513–525 (Jan 2004)
2. Biere, A., Heule, M.J.H.: The effect of scrambling CNFs. In: Pragmatics of SAT (2018)
3. 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)
4. Codel, C., Reeves, J., Heule, M.J.H., Bryant, R.E.: Bipartite perfect matching benchmarks. In: Pragmatics of SAT (2021)
5. 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)
6. Ellfers, J., Nordström, J.: Documentation of some combinatorial benchmarks. In: Proceedings of the SAT Competition 2016 (2016)
7. Haken, A.: The intractability of resolution. Theoretical Computer Science **39**, 297–308 (1985)
8. Jussila, T., Sinz, C., Biere, A.: Extended resolution proofs for symbolic SAT solving with quantification. In: Theory and Applications of Satisfiability Testing (SAT). LNCS, vol. 4121, pp. 54–60 (2006)
9. Marques-Silva, J., Lynce, I., Malik, S.: Conflict-driven clause learning SAT solvers. In: Handbook of Satisfiability, pp. 131–153. IOS Press (2009)
10. Sinz, C.: Towards an optimal CNF encoding of Boolean cardinality constraints. In: Principles and Practice of Constraint Programming (CP). LNCS, vol. 3709, pp. 827–831 (2005)
11. Sinz, C., Biere, A.: Extended resolution proofs for conjoining BDDs. In: Computer Science Symposium in Russia (CSR). LNCS, vol. 3967, pp. 600–611 (2006)
12. Urquhart, A.: The complexity of propositional proofs. The Bulletin of Symbolic Logic **1**(4), 425–467 (1995)