Practically Feasible Proof Logging for Pseudo-Boolean Optimization (Extended Abstract)

Wietze Koops
Lund University, Sweden
University of Copenhagen, Denmark

Daniel Le Berre
Univ. Artois, CNRS, UMR 8188 CRIL, France

Magnus O. Myreen
Chalmers University of Technology, Sweden
University of Gothenburg, Sweden

Jakob Nordström
University of Copenhagen, Denmark
Lund University, Sweden

Andy Oertel
Lund University, Sweden
University of Copenhagen, Denmark

Yong Kiam Tan
Institute for Infocomm Research (I^{2}R), A*STAR, Singapore
Nanyang Technological University, Singapore

Marc Vinyals
University of Auckland, New Zealand

This extended abstract presented at the CP/SAT Doctoral Program summarizes and as such builds heavily on the corresponding CP paper [14]. We refer to the CP paper for declaration of funding, acknowledgements, and references to supplementary material. In addition, the pseudo-Boolean solvers *RoundingSat* and *Sat4j*, for which we developed proof logging in this paper, have been submitted to the Pseudo-Boolean Competition 2025 presented at SAT 2025.

Introduction. Combinatorial optimization is a major success story in computer science. Astonishing progress over the past decades in the performance of combinatorial solvers allows these solvers to successfully solve real-world problems in model checking [2], cryptanalysis [16], planning [20], and many more application domains. These increases in performance have come at the cost of increasing the complexity of the algorithm and the solver software, however. As a result, even mature solvers have bugs, and sometimes incorrectly claim optimality or infeasibility, or even return "solutions" that actually do not satisfy all constraints [4, 9]. Such errors preclude the application of solvers to domains where correctness is crucial.

The Boolean satisfiability (SAT) community has pioneered the use of *certifying solvers* to address this problem. Certifying solvers use *proof logging* to output a machine-verifiable proof that the answer that the solver produced is correct. In the now de facto standard *DRAT* [22] format, such a proof essentially consists of just the clauses that the solver has learned. As a result, the overhead of proof generation is generally at most 10% of the solving time, while proof checking can be done within a factor 10 of the solving time. Proof checker also come with a formally verified backend, which means that correctness is certified by the strongest guarantees offered by formal methods [21].

The most successful approach to port these successes to more expressive paradigms is *pseudo-Boolean (PB) proof logging*, which uses 0–1 linear constraints, and has been applied in SAT-based optimization (MaxSAT) [1, 13], subgraph solving [10, 11], and constraint programming [17, 18], among others. In particular, enabling proof logging can increase solver runtimes by more than a factor 10, and proof checking can be roughly a factor 1,000 slower than solving. These overheads are orders of magnitude worse than SAT proof logging.

In this work, we present—to the best of our knowledge, for the first time—fast and practically feasible certified solving for a combinatorial optimization problem. We show how to provide proof logging for the state-of-the-art pseudo-Boolean solvers *RoundingSat* [6, 7, 8] and *Sat4j* [15], covering all techniques used in these solvers, and implement this in the pseudo-Boolean proof checker *VeriPB* and the formally verified backend *CakePB*. As our

Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

	logging			*VeriPB*				*VeriPB + CakePB*			
solver	max	95%	med	max	95%	med	RL	max	95%	med	RL
RoundingSat	1.463	1.204	1.027	16.58	7.10	1.34	8	19.17	9.22	1.43	10
Sat4j Cutting Planes	1.517	1.142	1.026	3.41	1.39	0.50	0	3.83	1.60	0.54	0
Sat4j Resolution	1.705	1.417	1.097	336.43	18.99	1.14	3	338.31	19.35	1.28	3

Table 1 Summary of the experimental results. 'max' stands for the maximum overhead, '95%' for the overhead within which 95% of the instances could be logged/checked, 'med' for the median overhead, and 'RL' (Resource Limit) for the number of instances that met a resource limit (10h checking time (3 instances for *Sat4j* Resolution), 14GB of memory (9 instances for *RoundingSat*), or 100GB of disk space (1 instance for *RoundingSat*)).

main result, we can provide formally verified conclusions within a factor 20 of the solving time, getting close to the overhead factor of 9 traditionally required in the SAT competitions.

To achieve this, we develop proof logging for a number of more advanced techniques in *RoundingSat* for which it is much less obvious how to express the reasoning in terms of pseudo-Boolean proof rules, including linear programming integration [6] and core-guided optimization [7]. In addition, we optimize various aspects of both the proof logging and checking, in order to reduce the overhead to an acceptable level.

Preliminaries. We now provide a very brief summary of the *VeriPB* proof system [3, 12], which is based on the cutting planes proof system [5]. The *VeriPB* proof system operates on a database of 0–1 linear inequalities $\sum_i a_i\ell_i \geq A$. Given two such inequalities, the cutting planes proof system allows adding them. For a positive integer c, we can multiply $\sum_i a_i\ell_i \geq A$ by c to obtain $\sum_i (ca_i)\ell_i \geq cA$, or divide by c and round to obtain $\sum_i \left\lceil \frac{a_i}{c} \right\rceil \ell_i \geq \left\lceil \frac{A}{c} \right\rceil$. Finally, we can *saturate* a constraint $\sum_i a_i\ell_i \geq A$ to obtain $\sum_i \min\{a_i, A\}\ell_i \geq A$. In the *VeriPB* proof syntax, these operations are implemented using reverse Polish notation (`pol`).

These rules can only derive implied constraints. In addition, *VeriPB* supports the so-called *redundance-based strengthening* (or *redundance* for short). This rule can, among other things, be used to define new variables.

Linear Programming Integration. *RoundingSat* is tightly integrated with a linear programming (LP) solver [6]. Most aspects of this integration only take (positive) linear combinations of constraints and can therefore be logged directly with `pol` lines. However, *mixed-integer rounding* (MIR) cut generation is an exception for two reasons. First, the MIR cut is not natively supported by *VeriPB*. Second, the cut generation procedure turns inequalities into equalities by introducing non-Boolean *integral slack variables*. Since the proof system only supports 0–1 valued variables, these slack variables cannot be directly encoded. Instead, we use a proof by contradiction to log a MIR cut.

Core-Guided Optimization. *RoundingSat* solves optimization problems using linear search, core-guided search, or a hybrid combination of the two [7]. In core-guided optimization, we iteratively derive constraints (called *core constraints*) showing that the objective of the given minimization problem cannot attain its current lower bound. After that, we introduce (Boolean) counter variables in the derived core constraint. These counter variables indicate that the core is actually stronger (i.e., has a larger right hand side), and effectively turn the core constraint into an equality. This equality is then used to rewrite the objective. As a part of the logging procedure, we make heavy use of the redundance rule to define the counter variables for the core constraints.

Implementation. We now discuss some optimizations implemented in the solvers and the checkers. In *RoundingSat*, we optimized logging of so-called *unit constraints* that state that a variable must take some fixed value, as well as simplifications of constraints using such unit constraints. In the proof checker *VeriPB*, we optimized solution checking. Finally, in *CakePB* two major optimizations were done. Firstly, constraint simplifications are optimized by merging multiple consecutive simplications before applying them to the constraint. Secondly, the efficiency of the map keeping track of occurrences of variables in constraints was improved.

Experiments. In Table 1 we summarize some experimental results, run on the instances from the Pseudo-Boolean Competition 2024 [19]. Overall, *RoundingSat* solved 555 instances, *Sat4j* Cutting Planes solved 322 instances, and *Sat4j* Resolution solved 366 instances. Out of these, proof checking using *VeriPB* and *CakePB* was successful on all but 13 instances, while the remaining instances could not be checked due to our resource limits: a memory limit on 9 instances solved by *RoundingSat*, a disk space limit on one instance solved by *RoundingSat*, and a timeout on 3 instances solved by *Sat4j* Cutting Planes. All in all, these results show that proof logging for pseudo-Boolean solving is now practically feasible.

References

1. Jeremias Berg, Bart Bogaerts, Jakob Nordström, Andy Oertel, Tobias Paxian, and Dieter Vandesande. Certifying without loss of generality reasoning in solution-improving maximum satisfiability. In *CP '24*, volume 307 of *LIPIcs*, pages 4:1–4:28, September 2024.
2. Armin Biere and Daniel Kröning. SAT-based model checking. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*, chapter 10, pages 277–303. Springer, December 2018.
3. Bart Bogaerts, Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. Certified dominance and symmetry breaking for combinatorial optimisation. *Journal of Artificial Intelligence Research*, 77:1539–1589, August 2023. Preliminary version in *AAAI '22*.
4. Robert Brummayer, Florian Lonsing, and Armin Biere. Automated testing and debugging of SAT and QBF solvers. In *SAT '10*, volume 6175 of *LNCS*, pages 44–57. Springer, July 2010.
5. William Cook, Collette Rene Coullard, and György Turán. On the complexity of cutting-plane proofs. *Discrete Applied Mathematics*, 18(1):25–38, November 1987.
6. Jo Devriendt, Ambros Gleixner, and Jakob Nordström. Learn to relax: Integrating 0-1 integer linear programming with pseudo-Boolean conflict-driven search. *Constraints*, 26(1–4):26–55, October 2021. Preliminary version in *CPAIOR '20*.
7. Jo Devriendt, Stephan Gocht, Emir Demirović, Jakob Nordström, and Peter Stuckey. Cutting to the core of pseudo-Boolean optimization: Combining core-guided search with cutting planes reasoning. In *AAAI '21*, pages 3750–3758, February 2021.
8. Jan Elffers and Jakob Nordström. Divide and conquer: Towards faster pseudo-Boolean solving. In *IJCAI '18*, pages 1291–1299, July 2018.
9. Xavier Gillard, Pierre Schaus, and Yves Deville. SolverCheck: Declarative testing of constraints. In *CP '19*, volume 11802 of *LNCS*, pages 565–582. Springer, October 2019.
10. Stephan Gocht, Ciaran McCreesh, Magnus O. Myreen, Jakob Nordström, Andy Oertel, and Yong Kiam Tan. End-to-end verification for subgraph solving. In *Proceedings of the 38th AAAI Conference on Artificial Intelligence (AAAI '24)*, pages 8038–8047, February 2024.
11. Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. Subgraph isomorphism meets cutting planes: Solving with certified solutions. In *IJCAI '20*, pages 1134–1140, July 2020.
12. Stephan Gocht and Jakob Nordström. Certifying parity reasoning efficiently using pseudo-Boolean proofs. In *AAAI '21*, pages 3768–3777, February 2021.

13 Hannes Ihalainen, Andy Oertel, Yong Kiam Tan, Jeremias Berg, Matti Järvisalo, Magnus O. Myreen, and Jakob Nordström. Certified MaxSAT preprocessing. In *IJCAR '24*, volume 14739 of *LNCS*, pages 396–418. Springer, July 2024.

14 Wietze Koops, Daniel Le Berre, Magnus O. Myreen, Jakob Nordström, Andy Oertel, Yong Kiam Tan, and Marc Vinyals. Practically feasible proof logging for pseudo-Boolean optimization. In *Proceedings of the 31st International Conference on Principles and Practice of Constraint Programming (CP '25)*, August 2025. To appear.

15 Daniel Le Berre and Anne Parrain. The Sat4j library, release 2.2. *Journal on Satisfiability, Boolean Modeling and Computation*, 7:59–64, July 2010.

16 Fabio Massacci and Laura Marraro. Logical cryptanalysis as a SAT problem. *Journal of Automated Reasoning*, 24:165–203, February 2000.

17 Matthew McIlree and Ciaran McCreesh. Proof logging for smart extensional constraints. In *CP '23*, volume 280 of *LIPIcs*, pages 26:1–26:17, August 2023.

18 Matthew McIlree, Ciaran McCreesh, and Jakob Nordström. Proof logging for the circuit constraint. In *CPAIOR '24*, volume 14743 of *LNCS*, pages 38–55. Springer, May 2024.

19 Pseudo-Boolean competition 2024. `https://www.cril.univ-artois.fr/PB24/`, August 2024.

20 Dominik Schreiber. Lilotane: A lifted SAT-based approach to hierarchical planning. *Journal of Artificial Intelligence Research*, 70:1117–1181, March 2021.

21 Yong Kiam Tan, Marijn J. H. Heule, and Magnus O. Myreen. cake_lpr: Verified propagation redundancy checking in CakeML. In *TACAS '21*, volume 12652 of *LNCS*, pages 223–241. Springer, March-April 2021.

22 Nathan Wetzler, Marijn J. H. Heule, and Warren A. Hunt Jr. DRAT-trim: Efficient checking and trimming using expressive clausal proofs. In *SAT '14*, volume 8561 of *LNCS*, pages 422–429. Springer, July 2014.