Certified Implicit Hitting Set Solving with Local Search for Pseudo-Boolean Optimization

Benjamin Bogø
University of Copenhagen, Copenhagen, Denmark

Xiamin Chen
Shanghai University of Finance and Economics, Shanghai, China

Wietze Koops
Lund University, Lund, Sweden
University of Copenhagen, Copenhagen, Denmark

Pinyan Lu
Shanghai University of Finance and Economics, Shanghai, China
Huawei Taylor Lab, Shanghai, China

Jakob Nordström
University of Copenhagen, Copenhagen, Denmark
Lund University, Lund, Sweden

Marc Vinyals
University of Auckland, Auckland, New Zealand

Qingzhao Wu
Shanghai University of Finance and Economics, Shanghai, China

Abstract

Implicit Hitting Set (IHS) Solving has been a successful approach in state-of-the-art solvers for Maximum Satisfiability (MaxSAT). Similar to solution-improving search and core-guided search, IHS has also recently been ported from MaxSAT to Pseudo-Boolean Optimization. Unlike the other techniques, there are currently no certified IHS solvers according to our knowledge. Traditionally, the IHS problem has been solved using an integer linear programming solver which does not have proof logging. We propose to use the certifying pseudo-Boolean solver RoundingSat with VeriPB proof logging to solve the IHS problem which allows us to build a reusable and certifying IHS solver. We also tightly integrate local search into the solver. Our work in progress implementation shows that proofs for solved instances are verified by VeriPB and the formally verified checker CakePB.

2012 ACM Subject Classification Theory of computation → Logic and verification; Mathematics of computing → Combinatorial optimization

Keywords and phrases implicit hitting set solving, pseudo-Boolean optimization, local search, proof logging, certifying algorithms, 0–1 integer linear programming

Digital Object Identifier 10.4230/LIPIcs...

1 Introduction

Pseudo-Boolean solving and optimization use 0–1 integer linear inequalities to express the constraints. Pseudo-Boolean solving is a generalization of Boolean Satisfiability (SAT) since SAT-CNF clauses can be expressed in terms of 0–1 integer linear inequalities. At the same time, pseudo-Boolean optimization is a restriction of integer linear programming since the values of the variables are restricted to 0 or 1.

Pseudo-Boolean optimization can be seen as a generalization of Maximum Satisfiability (MaxSAT) that is the optimization version of SAT. The MaxSAT paradigm includes algorithms

© Benjamin Bogø, Xiamin Chen, Wietze Koops, Pinyan Lu, Jakob Nordström, Marc Vinyals and Qingzhao Wu;
licensed under Creative Commons License CC-BY 4.0

such as solution-improving search [9], core-guided search [11] and implicit hitting set (IHS) search [7] that all recently been successfully ported to pseudo-Boolean optimization [8, 19].

Certifying algorithms [1, 16] have for a long time been the standard in SAT solving [14, 13]. However, certification is often harder in more expressive paradigms such as MaxSAT and pseudo-Boolean optimization. VeriPB proof logging [3, 12] has in more recent times allowed to certifying algorithms for solution-improving search [20] and core-guided search [2] in MaxSAT. However, there is no certifying algorithms for IHS search according to our knowledge.

Our current main contribution is a certifying implementation of implicit hitting set solving for pseudo-Boolean optimization with local search [6] and a reusable pseudo-Boolean solver to solve the IHS problem. The certified implicit hitting set solver has been implemented using the certified pseudo-Boolean solver RoundingSat [10, 15] and some extra proof logging of IHS that is verified by VeriPB and the formally verified checker CakePB [4].

2 Preliminaries

2.1 Pseudo-Boolean Optimization

A variable x_i in pseudo-Boolean optimization is a Boolean variable that can take the value 0 or 1. We denote the positive version of the variable x_i as x_i and its negation as $\overline{x_i} = 1 - x_i$. A literal ℓ_i is either x_i or $\overline{x_i}$. A pseudo-Boolean constraint C is a linear inequality of the form $C \doteq \sum_i a_i \ell_i \geq A$, where a_i are non-negative integer coefficients, ℓ_i are literals and A is the non-negative integer stating the degree of falsity. We say that C is satisfied under the assignment α, that maps variables to 0 or 1, if the inequity C holds when substitution the value of the variables in α.

A pseudo-Boolean satisfaction problem F is a set of pseudo-Boolean constraints and is a pseudo-Boolean optimization problem if it also contains an objective $O \doteq \sum_i w_i \ell_i$ that should be minimized, where w_i (without loss of generality) are non-negative integers.

2.2 Implicit Hitting Set Search

Implicit hitting set solving splits a pseudo-Boolean optimization problem (F, O) into a pseudo-Boolean satisfaction problem F, which will be called the *decision problem*, and a pseudo-Boolean optimization problem (K, O), which will be called the *IHS problem*, where K is a set of core constraints (initial empty) only over objective variables that grows during search. The IHS problem will at all times only consists of constraints implied by F such that any lower bound for the IHS problem is also a lower bound for the original problem. The idea is then to solve the simpler IHS problem to optimality to find an optimistic solution α and check if it the solution can be extended to a full solution β (containing all variables) for the decision problem. If yes, then the optimal solution has been found since α gives a lower bound, β gives an upper bound and the objective values of α and β are the same. If the solution cannot be extended, then a core constraint D can be extracted that describes why α was too optimistic such that is not found again. D is added to K before optimizing the IHS problem again.

2.3 Local Search

Local search is an incomplete method, i.e. it is not guaranteed to return an optimal solution. It starts with a full (not necessarily satisfiable) variable assignment β and keeps track of the current best solution β_{best} found. In every iteration, it flips one variable based on clause-weighting techniques that are used to find the most promising variables to flip. This yields a new variable assignment β' that replaces the current best solution β_{best} if β' satisfies all constraints in F and has a better objective value than β_{best}.

2.4 VeriPB Proofs

VeriPB proofs consist of a sequence of proof lines that each check if a given constraint can be derived in the way specified in the proof line. The relevant rules and their purpose for certifying IHS will very briefly be described. RoundingSat already produce proofs for conflict analysis using the `pol`-rule that allows to do cutting planes derivations (using addition, multiplication, division, saturation, and weakening). The `red`-rule, that is a generalization of the RAT-rule [18,8] used in SAT proof logging, can be used to introduce a new variable y with the same meaning as a constraint $C \doteq \sum_i a_i \ell_i \geq A$ via *reification*, i.e. $y \Leftrightarrow C$. This corresponds to the two pseudo-Boolean constraints $(y \Rightarrow C) \doteq A\overline{y} + \sum_i a_i \ell_i \geq A$ and $(y \Leftarrow C) \doteq \left(\sum_i a_i - A + 1\right) y + \sum_i a_i \overline{\ell_i} \geq \left(\sum_i a_i - A + 1\right)$. Finally, the `soli`-rule allows to log a solution β to introduce a solution-improving constraint $\sum_i w_i \ell_i \leq O(\beta) - 1$, where $O(\beta)$ is the objective value of β, stating that a better solution should exists.

3 Pseudo-Boolean Implicit Hitting Set Search

```
1  Function IHS(F, O)
2      β_best ← LocalSearch(F, O);
3      if β_best = ⊥ then β_best ← Solve(F, ∅);
4      if β_best = ⊥ then return ⊥;
5      lb ← 0;
6      ub ← O(β_best);
7      K ← Seeding(F, O);
8      while true do
9          lb, {α_1, ..., α_m} ← Optimize(K, O, lb, ub);
10         if lb ≥ ub then return β_best;
11         ub, β_best, K ← ProcessSolutions(F, ub, β_best, K, {α_1, ..., α_m});
12         if lb ≥ ub then return β_best;
13     end
```

Algorithm 1 Overview of our IHS algorithm implementation from [5].

3.1 Initialization

Algorithm 1 shows an overview of our implementation of IHS that is based on the description in [19]. It takes a pseudo-Boolean optimization problem (F, O) as input. First (line 2), local search is run on the original problem to try to find an initial solution. If it fails (line 3), then the complete pseudo-Boolean solver is run on the original problem without the objective to check if F is satisfiable or not. In case no solution is found ($\beta_{\text{best}} = \bot$, line 4), then the algorithm reports that there is no solution. In case there is a solution, the lower bound (line 5) is set to 0 and the upper bound (line 6) is set to the objective value of the found solution.

The set of core constraints K (line 7) is initialized with core constraints already present in the original problem F. This technique is called *constraint seeding* [19]. The idea is to take all constraints from F that are only over variables present in the objective and put them directly into K. We extend this technique by also trying to weaken non-objective variables from constraints to obtain non-trivial cores. For instance, if x_1 and x_2 are in the objective, then x_3 can be weakened in $x_1 + x_2 + x_3 \geq 2$ to obtain the core constraint $x_1 + x_2 \geq 1$. However, weakening x_3 in $x_1 + x_2 + x_3 \geq 1$ would yield the trivial constraint $x_1 + x_2 \geq 0$ and thus is not added. All seeded constraints are either present in the formula or can be derived using the `pol`-rule with weakening steps.

The main loop (line 8-13) runs until the lower and upper bounds meet (line 10 and 12). Note that this is slightly different from the description in section 2.2 where only the lower bound would increase because the IHS problem (K, O) was always solved to optimality in that description. In practice, it is very expensive to always solve to optimality, so an alternative approach is to instead guarantee to always make progress by either improving the upper bound or find a core constraint to avoid finding the same solution again [19]. Therefore, the IHS solver could stop as soon as it has a solution better than the current upper bound.

3.2 Solving the IHS Problem

The IHS problem (K, O) is solved (line 9) by either an incremental local search solver or a reusable complete pseudo-Boolean solver that both are initialized with the core constraints. The incremental aspect is that it is possible to add new core constraints to the solvers without having to start over from scratch every time a new constraint is added. It is currently work in progress how often local search should be used compared the complete solver and how often it makes sense to solve to optimality.

3.2.1 Local Search

The local search solver can either run until it finds a solution better than the current upper bound or until it reaches its iteration/time limit. In the first case, it finds a single solution while it might find zero or more solutions in the latter case which is denoted as $\{\alpha_1, \ldots, \alpha_m\}$ in Algorithm 1. In case local search does not find any solutions then the complete pseudo-Boolean solver is run to find solutions. Otherwise, the solutions from local search are processed as described in section 3.3. Note that the lower bound is not updated when local search is used since it is not known if the solution is optimal. The local search algorithm does not have to be certifying since it is enough to check that the found solutions are valid as a proof that they are valid solutions.

3.2.2 Reusable Complete Pseudo-Boolean Solver

The complete pseudo-Boolean solver can either run until it finds a solution better than the current upper bound or to optimality meaning that it finds at least one solution. In case it is solved to optimality, the lower bound is updated (line 9). The solver uses solution-improving search which normally introduces unconditionally solution-improving constraints whenever a solution is found. These constraints could be used to learn constraints that would not be valid in the next call. In order to be able to reuse these learned constraints, we instead introduce reified solution-improving constraints such that these are only turned on when they are valid. For instance, if the solver finds a solution of value 9 then it would introduce a new variable $y_9 \Leftrightarrow (O \leq 8)$ with the exact meaning that a solution with value 9 has been found. This can be justified by the `red`-rule as described in section 2.4. The solver then continues its search while assuming y_9 such that $O \leq 8$ is enforced. When the solver then finds a better solution with value 4 then it introduces $y_4 \Leftrightarrow (O \leq 3)$ as well as the clause $y_4 \Rightarrow y_9 \doteq \overline{y_4} + y_9 \geq 1$ because $O \leq 3$ clearly implies $O \leq 8$ and we still want the constraints for y_9 to be valid when we now instead assume y_4. This constraint is justified by adding $y_4 \Leftarrow (O \leq 3)$ and $y_9 \Rightarrow (O \leq 8)$ with the `pol`-rule and divide it to a clause. When the solver has to solve the updated IHS problem, it just starts with no assumptions and then introduce reified solution-improving constraints unless they already exist.

In case the (global) upper bound ub is updated (line 11), then an unconditional solution-improving constraint is added to the IHS problem and justified using the `soli`-rule. This would propagate all reified variables representing solution-improving constraints with values that are at least ub to 1 as units. Hence, these reified variables would vanish from all constraints as if the constraints were derived with the unconditional solution-improving constraint. When $lb = ub$ then the IHS solver is used to show optimality. At this point, it is possible to derive $O \geq lb$ from the IHS problem, and the unconditional solution-improving constraint $O \leq ub - 1$ is also added to the IHS problem. Hence, adding these constraints using the `pol`-rule would yield contradiction $0 \geq lb - ub + 1 \geq 1$.

3.3 Processing Solutions

Standard IHS solving only returns one solution at a time but there is no reason to throw away a solution unless it does not guarantee progress. In case multiple solutions were found then the solution α with the smallest objective value is considered. Note that the solutions are found with strictly decreasing objective value, so no sorting is needed. α is given to the decision problem where it is either used for single, independent or weight-aware core extraction [19]. In case the solution α can be extended to a full solution β then the upper bound is updated if the objective value of β is better that the current upper bound. This is always the case if no cores are found but might not be the case if independent or weight-aware core extraction removed some of the assumptions. If the upper bound is improved, then all unprocessed solutions that does not have a better objective value are discarded. If the decision solver returns core constraints, then all solutions that falsifies at least one of the core constraints are discarded. Hence, there is progress in both cases. Note that the current processed solution α will be discarded in both cases. The process continues until there are no solutions left.

4 Experimental Evaluation

The algorithm and techniques have been implemented in the complete pseudo-Boolean solver RoundingSat [10] with integration of local search [6]. The solver was run with different configurations on the optimization (OPT-LIN) instances of the pseudo-Boolean competition 2024 [17]. Currently, the performance of the implementation is still work in progress. However, the generated proofs of the solved instances were all successfully checked by VeriPB and the formally verified checker CakePB [4].

An early certifying version of the solver (not using local search) was submitted to the pseudo-Boolean competition 2025 [18] where results can be found when they are published.

5 Concluding Remarks

We have implemented a certifying implicit hitting set optimizer for pseudo-Boolean optimization problems that is verified by VeriPB and the formally verified checker CakePB. Local search is integrated such that it can be used in an incremental manner and the complete pseudo-Boolean solver is reusable such that they do not have to start from scratch when they solve the implicit hitting set problem. However, the performance of the current implementation still needs polishing to improve performance to compete with the current state-of-the-art IHS solvers.

References

1. Eyad Alkassar, Sascha Böhme, Kurt Mehlhorn, Christine Rizkallah, and Pascal Schweitzer. An introduction to certifying algorithms. *it - Information Technology Methoden und innovative Anwendungen der Informatik und Informationstechnik*, 53(6):287–293, December 2011.
2. Jeremias Berg, Bart Bogaerts, Jakob Nordström, Andy Oertel, and Dieter Vandesande. Certified core-guided MaxSAT solving. In *Proceedings of the 29th International Conference on Automated Deduction (CADE-29)*, volume 14132 of *Lecture Notes in Computer Science*, pages 1–22. Springer, July 2023.
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. Bart Bogaerts, Ciaran McCreesh, Magnus O. Myreen, Jakob Nordström, Andy Oertel, and Yong Kiam Tan. Documentation of VeriPB and CakePB for the SAT competition 2023. Available at `https://satcompetition.github.io/2023/checkers.html`, March 2023.
5. Benjamin Bogø, Xiamin Chen, Wietze Koops, Pinyan Lu, Jakob Nordström, Marc Vinyals, and Qingzhao Wu. Certified implicit hitting set solving with local search for pseudo-boolean optimization. Accepted as work-in-progress for Pragmatics of SAT 2025 (PoS25), June 2025.
6. Xiamin Chen, Zhendong Lei, and Pinyan Lu. Deep cooperation of local search and unit propagation techniques. In Paul Shaw, editor, *30th International Conference on Principles and Practice of Constraint Programming, CP 2024, September 2-6, 2024, Girona, Spain*, volume 307 of *LIPIcs*, pages 6:1–6:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. URL: `https://doi.org/10.4230/LIPIcs.CP.2024.6`, `doi:10.4230/LIPICS.CP.2024.6`.
7. Jessica Davies and Fahiem Bacchus. Solving MAXSAT by solving a sequence of simpler SAT instances. In *Proceedings of the 17th International Conference on Principles and Practice of Constraint Programming (CP '11)*, volume 6876 of *Lecture Notes in Computer Science*, pages 225–239. Springer, September 2011.
8. 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 *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI '21)*, pages 3750–3758, February 2021.
9. Niklas Eén and Niklas Sörensson. Translating pseudo-Boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2(1-4):1–26, March 2006.
10. Jan Elffers and Jakob Nordström. Divide and conquer: Towards faster pseudo-Boolean solving. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI '18)*, pages 1291–1299, July 2018.
11. Zhaohui Fu and Sharad Malik. On solving the partial MAX-SAT problem. In *Proceedings of the 9th International Conference on Theory and Applications of Satisfiability Testing (SAT '06)*, volume 4121 of *Lecture Notes in Computer Science*, pages 252–265. Springer, August 2006.
12. Stephan Gocht and Jakob Nordström. Certifying parity reasoning efficiently using pseudo-Boolean proofs. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI '21)*, pages 3768–3777, February 2021.
13. Marijn J. H. Heule, Warren A. Hunt Jr., and Nathan Wetzler. Trimming while checking clausal proofs. In *Proceedings of the 13th International Conference on Formal Methods in Computer-Aided Design (FMCAD '13)*, pages 181–188, October 2013.
14. Marijn J. H. Heule, Warren A. Hunt Jr., and Nathan Wetzler. Verifying refutations with extended resolution. In *Proceedings of the 24th International Conference on Automated

Deduction (CADE-24), volume 7898 of *Lecture Notes in Computer Science*, pages 345–359. Springer, June 2013.

15 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.

16 Ross M. McConnell, Kurt Mehlhorn, Stefan Näher, and Pascal Schweitzer. Certifying algorithms. *Computer Science Review*, 5(2):119–161, May 2011.

17 `https://www.cril.univ-artois.fr/PB24/`, August 2024.

18 `https://www.cril.univ-artois.fr/PB25/`, August 2025.

19 Pavel Smirnov, Jeremias Berg, and Matti Järvisalo. Pseudo-Boolean optimization by implicit hitting sets. In *Proceedings of the 27th International Conference on Principles and Practice of Constraint Programming (CP '21)*, volume 210 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 51:1–51:20, October 2021.

20 Dieter Vandesande, Wolf De Wulf, and Bart Bogaerts. QMaxSATpb: A certified MaxSAT solver. In *Proceedings of the 16th International Conference on Logic Programming and Non-monotonic Reasoning (LPNMR '22)*, volume 13416 of *Lecture Notes in Computer Science*, pages 429–442. Springer, September 2022.