Improving Watched Pseudo-Boolean Propagation with Significant Literals

Mia Müßig[1,2], Jan Johannsen[1]

[1]Institut für Informatik, Ludwig-Maximilians-Universität München, Germany
[2]Department of Statistical Learning, ScaDS.AI Leipzig, Germany

Abstract

A key challenge in pseudo-boolean solving is the efficient detection and processing of unit literals. In SAT solving this is done by using the watched literal scheme, but for general pseudo-boolean constraints there is no dominant method. This paper introduces the significant literal framework to generalize existing watched literal schemes for pseudo-boolean solvers, which is implemented in a modification of the RoundingSAT solver. For this modification, small improvements can be observed on the instances from the decision and optimization tracks of the 2024 Pseudo-Boolean Competition, and a substantial improvement in instances with large coefficients like Knapsack.

Keywords

Pseudo-Boolean Solving, Unit Propagation, Watched Literals

1. Introduction

Contemporary satisfiability (SAT) solvers achieve efficiency by conflict driven clause learning (CDCL) and unit propagation, making them a powerful tool for boolean problems like model checking in chip development, formal software verification and scheduling problems. On the other hand, Integer Linear Programming (ILP) solvers use integer relaxation and the branch-and-cut procedure to dominate in industrial applications including production planning, network optimization and portfolio selection. Pseudo-boolean solving has emerged as a promising middle ground between the two approaches, aiming to combine the advantages of both methods. SAT methods like unit propagation and conflict analysis are now applied to the powerful cutting plane system underlying ILP solvers.

However, neither unit propagation nor conflict analysis can be generalized to arbitrary pseudo-boolean constraints without careful considerations. The latter received significant attention, with various papers discussing different methods of constructing conflict constraints [1, 2, 3]. For unit propagation, the focus is on adapting the watched literal scheme introduced in the SAT solver Chaff [4], which is part of almost all modern CDCL solvers. However, many teams developing competitive pseudo-boolean solvers observed none to only minimal improvement compared to the simpler counting method [1, 3, 5]. This changed when Devriendt [6] obtained a significant performance increase for his watched literal scheme, which replaces the computation of the maximum coefficient of unassigned literals used to determine the watched literal set by a fixed upper bound. This method has been implemented in the RoundingSAT solver, which is currently considered the fastest pseudo-boolean solver [7]. Recently it has been observed that the performance improvement of watched literals can be amplified by caching optimizations and a hybrid unit propagation approach [8].

This paper aims to demonstrate that the watched literal scheme of RoundingSAT can be further improved by introducing significant literals, which are especially effective for constraints with large coefficient sizes. Our approach uses a tighter upper bound by finding the maximum coefficient of unassigned *significant* literals, but keeps the fixed upper bound as in Devriendt's scheme [6] for constraints without significant literals. The significance criterion for literals is determined in such a

Joint Proceedings of the 23rd International Workshop on Satisfiability Modulo Theories and of the 16th Pragmatics of SAT International Workshop

M.Muessig@campus.lmu.de (M. Müßig); jan.johannsen@ifi.lmu.de (J. Johannsen)
0000-0003-0249-6831 (M. Müßig)

way that we strike a balance between time savings from having fewer watched literals and the higher cost of updating the new watched literal scheme. Additionally, we attempt to explain a connection of the performance differences between the various watched literal schemes and the size distributions of the constraint coefficients.

We give a detailed description of the algorithm behind the significant literal approach in section 3, and then in section 4 present performance comparisons with the current RoundingSAT implementation across various datasets.

2. Preliminaries

A *pseudo-boolean problem* consists of variables $x_i \in \{0,1\}$, with *literals* l_i representing either x_i or $\overline{x}_i := 1 - x_i$ and *pseudo-boolean constraints* C of the form $\sum_{i=1}^{n} a_i l_i \geq b$ for $a_i, b \in \mathbb{N}$.

Without loss of generality we will assume that the coefficients are in descending order, so $\forall i : a_i \geq a_{i+1}$. A *cardinality constraint* is a pseudo-boolean constraint with $\forall i \ : \ a_i = 1$. If additionally $b = 1$, the constraint is a *clause*.

We denote the current (partial) *assignment* of the variables l_i with ρ, identified with the set of literals set to 1. The *slack* of a pseudo-boolean constraint with the current assignment ρ is defined as $slack(C, \rho) = -b + \sum_{\overline{l}_i \notin \rho} a_i$.

Informally, the slack represents the amount by which the left-hand side can exceed the right-hand side without unassigning literals. Thus, if $slack(C, \rho) < 0$, the constraint C can not be satisfied with ρ, and we need to backtrack.

A literal l_i is called a *unit literal*, if $slack(C, \rho) < a_i$. This means that l_i must be set to 1 in order to satisfy the constraint C, as otherwise $slack(C, \rho \cup \{\overline{l_i}\}) < 0$.

For a pseudo-boolean constraint C we denote its watched literals with $W(C)$, a subset of its non-falsified literals, i.e. $W(C) \subseteq \{l_i \in C : \overline{l_i} \notin \rho\}$. Note that $W(C)$ depends on the current assignment ρ. The *maximum coefficient* a_{max} of a constraint C is defined as $a_{max} := \max\{a_i \in C : l_i, \overline{l_i} \notin \rho\}$. If all literals of C are assigned, we define it as $a_{max} = 0$.

The *watch slack* is the slack restricted to $W(C)$: $wslack(C, \rho) = -b + \sum_{l_i \in W(C)} a_i$. By definition $wslack(C, \rho) \leq slack(C, \rho)$ holds.

Lemma 1. *C contains a unit literal if and only if no watched literals set $W(C)$ exists such that $wslack(C, \rho) \geq a_{max}$.*

Proof. If C contains a unit literal, then by definition we have an unassigned literal l_i with $slack(C, \rho) < a_i$ and $a_{max} \geq a_i$. So we arrive at $a_{max} \geq a_i > slack(C, \rho) \geq wslack(C, \rho)$. Thus, no watched literal set $W(C)$ with $wslack(C, \rho) \geq a_{max}$ can exist.

If for all sets of literals $W(C)$ we have $wslack(C, \rho) < a_{max}$, this also holds true for $W(C) = \{l_i \mid \overline{l_i} \notin \rho\}$, i.e. if we watch all non-falsified literals. In that case we have $wslack(C, \rho) = slack(C, \rho)$, leading us to $slack(C, \rho) < a_{max}$, and therefore the literal corresponding to a_{max} must be unit. □

If we replace a_{max} with an upper bound, a weaker version of Lemma 1 still holds:

Lemma 2. *Let a_{up} be an upper bound, i.e. $a_{up} \geq a_{max}$. If C contains a unit literal, then no watched literals set $W(C)$ exists such that $wslack(C, \rho) \geq a_{up}$.*

Proof. By Lemma 1 we know that if C contains a unit literal, then we can not choose watched literals $W(C)$ such that $wslack(C, \rho) \geq a_{max}$. Since $a_{up} \geq a_{max}$, this immediately implies that we also can not find $W(C)$ with $wslack(C, \rho) \geq a_{up}$. □

The problem of using Lemma 1 as a criterion for finding unit literals is that a_{max} needs to be known for all constraints. So every time a variable x_i is assigned or unassigned, we have to update a_{max} in all constraints that contain x_i or $\overline{x}_i$. This procedure can take up to two thirds of the total unit propagation runtime [3], and defeats the original purpose of watched literals.

RoundingSAT instead uses a criterion based on Lemma 2 with $a_{up} = a_1$ [6]. As a_1 is constant, no updates are necessary to preserve $a_{up} \geq a_{max}$. One downside of this approach is that now our unit propagation methods can encounter false positives, as the converse of Lemma 2 does not hold true. Additionally, it often leads to a larger $W(C)$, increasing the work necessary for maintaining the watched literal scheme.

3. Watched Literal Propagation with Significant Literals

The idea behind significant literals is finding a middle ground between always updating a_{max} and replacing it with a constant upper bound. For that we choose an arbitrary criterion determining if l_i is significant for a constraint C. We denote the set of all significant literals for C with $S(C)$. Since the criterion does not depend on ρ, $S(C)$ is constant.

Now we define $a_{smax} := \max\{a_i \in C : l_i, \overline{l_i} \notin \rho \vee l_i \notin S(C)\}$, i.e. a_{smax} is the largest coefficient of a literal which is either unassigned or not significant. As with a_{max}, if all literals are assigned and significant, we simply define $a_{smax} = 0$. By definition $a_{smax} \geq a_{max}$, so we can apply Lemma 2 with $a_{up} = a_{smax}$.

This framework generalizes the two previously mentioned methods. If we choose a significance criterion which declares all literals as significant, we have $a_{smax} = a_{max}$ and recover the method which calculates a_{max} for all constraints. If we instead declare no literal to be significant for any constraint, we have $a_{smax} = a_1$ and recover the current method in RoundingSAT [6].

To elaborate what significance criteria are useful, we look at the following "worst case" constraint for the current RoundingSAT unit propagation system: $100y + \sum_{i=1}^{100} x_i \geq 10$.

Starting with $\rho = \emptyset$, we have $a_1 = a_{max} = 100$ and $slack(C, \rho) = 190$. Then the implementation iteratively adds watched literals until $wslack(C, \rho) \geq 100$, which results in $W(C) = \{y, x_1, \dots x_{10}\}$. If the next decision of the solver is $\rho = \{\overline{y}\}$, all 100 x_i literals will be added to $W(C)$ without achieving $wslack(C, \rho) \geq 100$. This means that until the solver reverses the assignment of y, the watched set $W(C)$ will be unnecessarily large, and no unit literal can exist until at least 90 of the x_i variables are assigned.

When we instead use significant literals and a criterion which declares y to be significant for the constraint, we only need $wslack(C, \rho) \geq 1$ after $\rho = \{\overline{y}\}$. This allows for the much smaller watched set $W(C) = \{x_1, \dots x_{11}\}$, which reduces the workload for updating the watched literal scheme in further assignments.

Here the difference between the two watched literal schemes is exaggerated, as the example constraint is deliberately chosen to amplify this problem. Simply choosing the equivalent constraint $10y + \sum_{i=1}^{100} x_i \geq 10$ allows the current RoundingSAT implementation to watch only roughly twice as many literals as the significant literal approach. However, it still illustrates that the difference between the two watched literal schemes is mainly dependent on the size of the constraint coefficients. This will also be supported by empirical evidence in the following section.

3.1. Algorithm

Algorithm setSmax demonstrates how to set $C.a_{smax}$ to the value of a_{smax}. Here $C.l[k]$ is a reference to a literal l_k occurring in C, $C.a[k]$ denotes its corresponding coefficient, and size(C) denotes the number of literals in C. We also count the number of backjumps which occurred so far with $bkjmps$. In the while loop we search for an unassigned or non-significant literal, and finish immediately if we found one. Since the literals in C are sorted by descending order of their coefficients, the first found literal is guaranteed to have the largest coefficient, and we have found a_{smax}.

To further optimize this procedure, we remember the index k at which we left the loop in the last call to setSmax. If no backtracking occurred in the meantime, no literals have been unassigned and so all literals before k are still assigned and significant. Thus, the first condition in algorithm setSmax ensures that we only restart the search from the beginning if a backjump occurred. This idea is directly inspired by the algorithm propagate first presented by Devriendt [6]. Each time the algorithm propagates the

Algorithm 1 setSmax (constraint C)

```
1:  Input: Constraint C
2:  if C.lastset < bkjmps then
3:      C.k ← 1
4:      C.lastset ← bkjmps
5:  while C.k ≤ size(C) do
6:      l_k ← C.l[k]
7:      if l_k, l̄_k ∉ ρ or l_i ∉ S(C) then
8:          C.a_smax ← C.a[k]
9:          return
10:     C.k ← C.k + 1
11: C.a_smax ← 0
```

literal l_i in the constraint C it calls PROPAGATE with the constraint and idx being set to i to indicate the index of the propagated literal in C.

When we now assign a literal we check in each constraint in which it is significant, if its corresponding coefficient is equal to a_{smax} of that constraint. Only if that it is the case, we need to call algorithm SETSMAX to update the value. For unassigning we only need to check for each significant constraint if its corresponding coefficient is bigger than a_{smax} and if so set a_{smax} to its coefficient.

Now we need to integrate the new $C.a_{smax}$ into the unit propagation of RoundingSAT, which consists of PROPAGATEOPT, PROCESSWATCHES and BACKJUMP [6]. Luckily only the first routine needs to be modified into algorithm PROPAGATE, since it is the only one dependent on the choice of a_{up}. This independence of a_{up} also holds for proof of the watch slack invariant [6], which means that all routines correctly preserve $C.\text{wslk} = wslack(C, \rho)$.

However, one has to be careful which optimizations of PROPAGATEOPT are still logically sound for a non-constant a_{up}. The first optimization can be kept, as the following lemma shows.

Lemma 3. *If no backtracking occurred, the while loops in line 7 and 19 can be restarted at the index where they left off in the last call, without changing the behaviour of the algorithm.*

Proof. Without backtracking any literal assigned in ρ at the last call to function PROPAGATE remain assigned. Thus, if i is the index, where the while loop of line 7 left off, the literals $l_1, \dots l_{i-1}$ either remain false or are already watched, and we can safely skip them.

Similarly, if the while loop of line 19 terminates with index j, then all literals $l_1, \dots l_{j-1}$ are already assigned. Without backtracking this still holds true now, and we can restart the search with index j. □

The second optimization, which skips the search for new watched literals if $C.\text{wslk} + C.a[idx] \geq C.a_{up}$, does not work for non-constant a_{up}. The idea behind it is that if at some assignment the while loop of line 7 terminates because of the condition $C.\text{wslk} < C.a_{smax}$, all non-falsified literals are watched. Until we have backtracked so far that this specific assignment is reversed, all non-falsified literals remain watched, and any further search for new literals to watch will be unnecessary. However, with a non-constant a_{up} further assignments can make it possible to fulfil $wslack(C, \rho) \geq a_{up}$, even if that was not possible at some past call. Thus, the optimization can only be applied if a_{up} is constant in the constraint. In our case we simply check this when $S(C) = \emptyset$, since we have $a_{smax} = a_1$ if no literals are significant for a constraint.

3.2. Significance Criteria

We experiment with various significance criteria, which aim to identify literals where the increased cost of updating a_{smax} is outweighed by allowing for a smaller watched literal set $W(C)$. Let $c \in \mathbb{N}$ be a fixed cut-off value, and $s \in \mathbb{R}$ a fixed scaling factor . We consider the following criteria, where the literal l_i with coefficient a_i is significant for a constraint C if:

Algorithm 2 PROPAGATE (constraint C, integer idx), modification of PROPAGATEOPT [6]

Input: assignment ρ, literal l, constraint C
if C.lastprop $< bkjmps$ **then**
  $C.i \leftarrow 1$
  $C.j \leftarrow 1$
  C.lastprop $\leftarrow bkjmps$
if C.wslk $+ C.a[idx] \geq C.a_{smax} \vee S(C) \neq \emptyset$ **then**
  while $C.i \leq$ size(C) and C.wslk $< C.a_{smax}$ **do**
    if $\overline{C.l}[i] \notin \rho$ and $C.l[i] \notin W(C)$ **then**
      ▷ Add $C.l[i]$ to watched literals
      $W(C) \leftarrow W(C) \cup \{C.l[i]\}$
      C.wslk $\leftarrow C$.wslk $+ C.a[i]$
    $C.i \leftarrow C.i + 1$
if C.wslk $\geq C.a_{smax}$ **then**
  ▷ Enough watched literals, unwatch propagated literal
  $W(C) \leftarrow W(C) \setminus \{C.l[idx]\}$
  return OK
if C.wslk < 0 **then return** CONFLICT
▷ A unit literal could exist
while $C.j \leq$ size(C) and C.wslk $< C.a[j]$ **do**
  $l_j \leftarrow C.l[j]$
  if $l_j, \bar{l}_j \notin \rho$ **then**
    ▷ Enqueue new unit literal
    $\rho \leftarrow \rho \cup \{l_j\}$
  $C.j \leftarrow C.j + 1$
return OK

- (Absolute size) $a_i > c$
- (Absolute size of maximum coefficient) $a_1 > c$
- (Relative size) $a_1 > s \sum_{j=2}^{n} a_j$

It should be noted that for the latter two criterion always either all or none of the literals are declared to be significant, since the definition does not depend on i. Additionally, each of the three criteria can also be restricted to only input constraints, which we will mark as "C input $\wedge$ criterion" in the legends of runtime plots. If no criterion is specified, we simply define all literals in input constraints to be significant.

4. Experimental Evaluation

To perform the evaluation we use commit d34b6bed of the RoundingSAT solver, which is the latest version as of December 2024 [7]. We extended RoundingSAT to implement the unit propagation with significant literals. The implementation and the data presented in the following plots are publicly available [9]. It should be noted that - as in the current RoundingSAT implementation - the respective watched literal scheme is only applied to "true" pseudo-boolean constraints, while clauses and cardinality constraints are treated separately. Since $\forall i : a_i = 1$ applies to clauses and cardinality constraints by definition, one can simply use the standard watched literal approach from SAT solving with $b + 1$ watched literals for each constraint.

All runtime measurements are done without proof logging, although we verified all certificates of an independent run using the VeriPB verifier [10]. The benchmark was run on an AMD Ryzen 5950X CPU with a timeout of $3600s$.

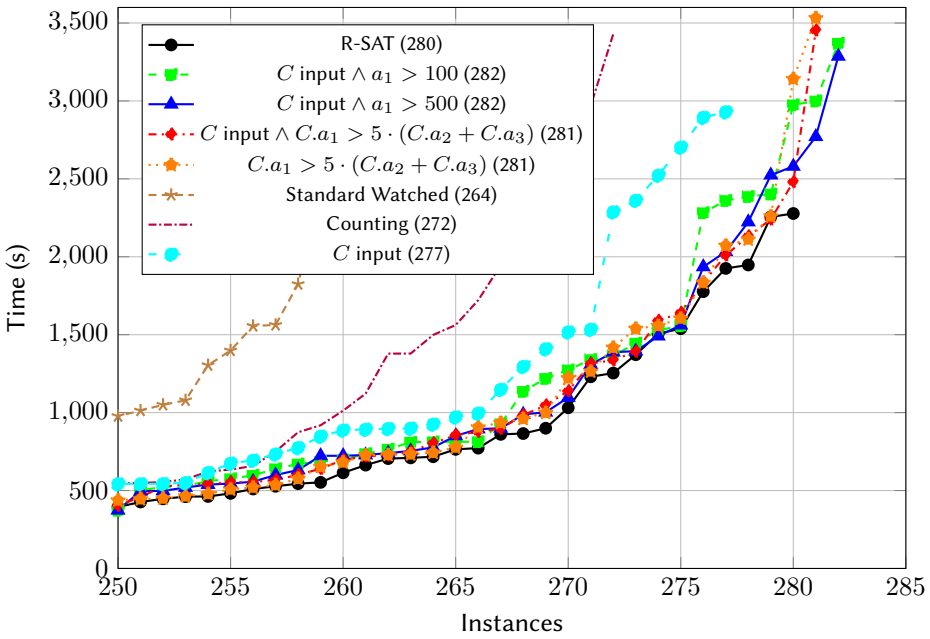

Figure 1: Runtime comparison on the DEC-LIN track. The number of instances solved is given in parentheses.

Our first dataset will consist of the 398 selected instances of the DEC-LIN track in the Pseudo-Boolean Competition 2024 [11], which aim to be a representative sample of pseudo-boolean decision problems. We evaluate the unmodified RoundingSAT solver (R-SAT) without the optional SoPlex Linear Programming integration and without the hybrid mode suggested by Robert Nieuwenhuis et al. [8], as both actually reduce the number of instances solved for this specific dataset. Additionally, we evaluate the counting method - which explicitly calculates the slack of each constraint - by disabling watched propagation in RoundingSAT. Finally we present the best performing significance criteria and "Standard Watched", which uses a significance criterion declaring every literal to be significant for all its constraints, in order to represent the traditional watched literal scheme with $a_{up} = a_{max}$.

Figure 1 demonstrates that the counting method is less efficient than the unit propagation by Devriendt [6], although faster than the naive way of implementing watched literals. We also observe that some significance criteria already gain a small advantage over the existing RoundingSAT implementation on the DEC-LIN instances. An extensive evaluation of different significance criteria on these instances is given in appendix B.

As shown in appendix A, the most successful criteria for DEC-LIN also perform well on the OPT-LIN track in the Pseudo Boolean Competition 2024, which comprises a collection of 487 optimization problems. Here again, a small improvement compared to RoundingSAT can be observed.

Across both datasets we notice that unit propagation with significant literals works best when it is only applied to the input constraints and not to the learned constraints. Our explanation for this behaviour is the difference in the distribution of coefficient sizes as shown in Figures 2 and 3.

To produce Figure 2 we have collected all constraints in the selected instances of the DEC-LIN track, which are not clauses or cardinality constraints and group all their coefficients by absolute values in buckets. For Figure 3 we did the same for all learned constraints RoundingSAT on the dataset, but excluded instances which are not solved within the $3600s$ timeout.

We observe that coefficients in input constraints are far more unevenly distributed. Thus, a_{smax} can often be far smaller than a_1, leading to a smaller and therefore less expensive watched literal set. On learned constraints with a more even distribution, a_{smax} remains similar to a_1, thus the cost of updating a_{smax} outweighs the benefit of only marginally smaller watched literal sets.

While significant literals already yield a small improvement on general instances, they are substantially better if the instances mainly consist of constraints with large coefficients. One example are the

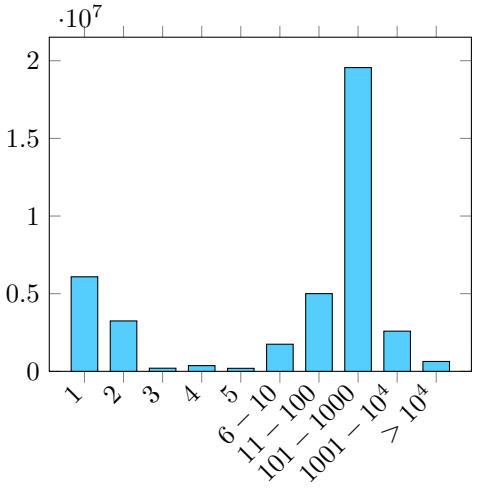

Figure 2: Distribution in input constraints.

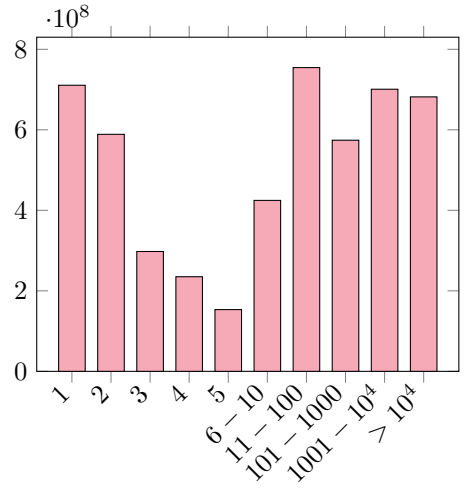

Figure 3: Distribution in learned constraints.

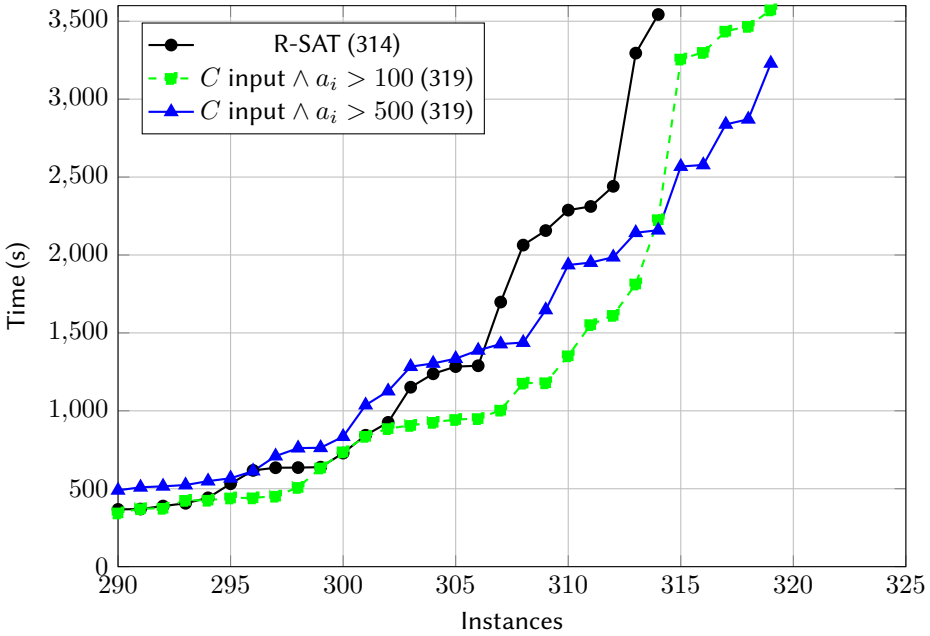

Figure 4: Runtime comparison of significant literal schemes on Knapsack instances. The number of instances solved is given in parentheses.

783 Knapsack instances submitted to the OPT-LIN track of the Pseudo Boolean Competition 2024, as Figure 4 shows.

The plot demonstrates that the two criteria clearly outperform the unmodified RoundingSAT solver, both in the number of instances solved and the general runtime of hard instances.

We further support this observation by looking at another application with large coefficient in the benchmark dataset of the Pseudo Boolean Competition, the "Virtual Machine Consolidation" problem [12]. The dataset includes 27 decision and 27 optimization problems submitted as benchmarks for the Pseudo Boolean Competition 2015 and is thus significantly smaller than the other datasets. However we can still observe in Figure 5, that the significance criteria, which performed best for the Knapsack instances, also outperform the current RoundingSAT implementation here.

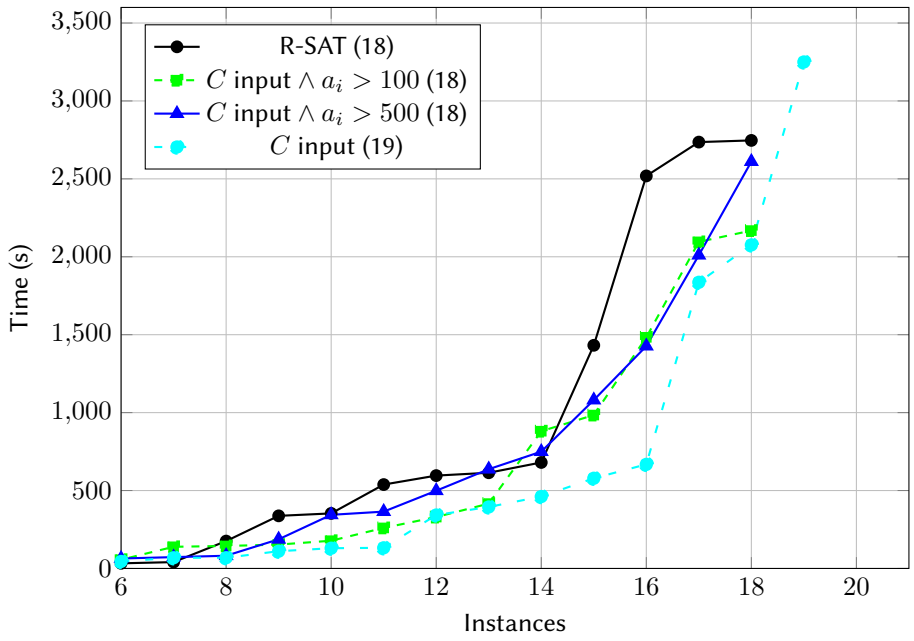

Figure 5: Runtime comparison of significant literal schemes on the PBFVMC instances.

5. Conclusion

It has been successfully demonstrated how significant literals can speedup unit propagation for pseudo-boolean constraints. Especially for applications with a large average coefficient size we have observed a substantial improvement compared to the current RoundingSAT watched propagation scheme.

In addition, an avenue for future improvements is indicated by the observation that the performance differences between the watched literal schemes is mainly determined by the distribution of coefficients. One possibility could be to select the significance criterion itself during the preprocessing step based on the coefficients of the individual instance. Another improvement could be the adaptation of the logging method developed by Robert Nieuwenhuis et al. [8], which ensures that the search behavior of the solver is completely independent of the exact unit propagation scheme chosen. This would make it easier to identify small performance improvements from different significant literal definitions.

Acknowledgments

This paper is based on the first author's Master's Thesis [13] supervised by the second author. The authors would like to thank the anonymous reviewers of SAT 2025 and the Pragmatics of SAT 2025 Workshop for their careful reading and helpful suggestions.

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

[1] D. Chai, A. Kuehlmann, A fast pseudo-boolean constraint solver, in: Proceedings of the 40th Annual Design Automation Conference, DAC '03, 2003, p. 830–835. doi:10.1145/775832.776041.

[2] J. Elffers, J. Nordström, Divide and conquer: Towards faster pseudo-boolean solving, in: International Joint Conference on Artificial Intelligence, 2018.

[3] H. M. Sheini, K. A. Sakallah, Pueblo: a modern pseudo-boolean sat solver, in: Design, Automation and Test in Europe, volume 2, 2005, pp. 684–685. doi:10.1109/DATE.2005.246.
[4] M. W. Moskewicz, L. Zhang, C. F. Madigan, S. Malik, Y. Zhao, Chaff: Engineering an Efficient SAT Solver , in: Design Automation Conference, 2001, pp. 530–535. doi:10.1109/DAC.2001.935565.
[5] F. A. Aloul, A. Ramani, I. L. Markov, K. A. Sakallah, Generic ilp versus specialized 0-1 ilp: an update, in: Proceedings of the 2002 IEEE/ACM International Conference on Computer-Aided Design (ICCAD '02), ACM, New York, NY, USA, 2002, pp. 450–457. doi:10.1145/774572.774638.
[6] J. Devriendt, Watched Propagation of 0-1 Integer Linear Constraints, in: Principles and Practice of Constraint Programming, 2020, pp. 160–176.
[7] J. Elffers, J. Devriendt, S. Gocht, J. Nordström, RoundingSat - the pseudo-boolean solver powered by proof complexity!, 2024. URL: https://gitlab.com/MIAOresearch/software/roundingsat.
[8] R. Nieuwenhuis, A. Oliveras, E. Rodríguez-Carbonell, R. Zhao, Speeding up Pseudo-Boolean Propagation, in: 27th International Conference on Theory and Applications of Satisfiability Testing (SAT 2024), volume 305 of *Leibniz International Proceedings in Informatics (LIPIcs)*, 2024, pp. 22:1–22:18. doi:10.4230/LIPIcs.SAT.2024.22.
[9] M. Müßig, RoundingSAT with significant literals, 2025. URL: https://gitlab2.cip.ifi.lmu.de/muessig/roundingsat.
[10] S. Gocht, C. McCreesh, J. Nordström, VeriPB: the easy way to make your combinatorial search algorithm trustworthy, in: From Constraint Programming to Trustworthy AI, workshop at the 26th International Conference on Principles and Practice of Constraint Programming (CP '20), 2020.
[11] Pseudo-Boolean Competition, Pseudo-Boolean Competition 2024, 2024. URL: https://www.cril.univ-artois.fr/PB24/, organized by Olivier Roussel, CRIL, Université d'Artois, France.
[12] B. C. Ribas, R. M. Suguimoto, R. A. Montaño, F. Silva, M. Castilho, PBFVMC: A new pseudo-boolean formulation to virtual-machine consolidation, in: 2013 Brazilian Conference on Intelligent Systems, 2013, pp. 201–206. doi:10.1109/BRACIS.2013.41.
[13] M. Müßig, Investigating Watched Literal Schemes for Pseudo-Boolean Solvers, Master's thesis, LMU Munich, 2025.

A. Comparison with OPT-LIN Instances

The comparison for the OPT-LIN track of the Pseudo-Boolean Competition 2024 is shown in figure 6.

B. Comparison of different Significance Definitions

In figures 7 through 12, we compare the runtime of different significance criteria on the DEC-LIN instances of the Pseudo-Boolean Competition 2024.

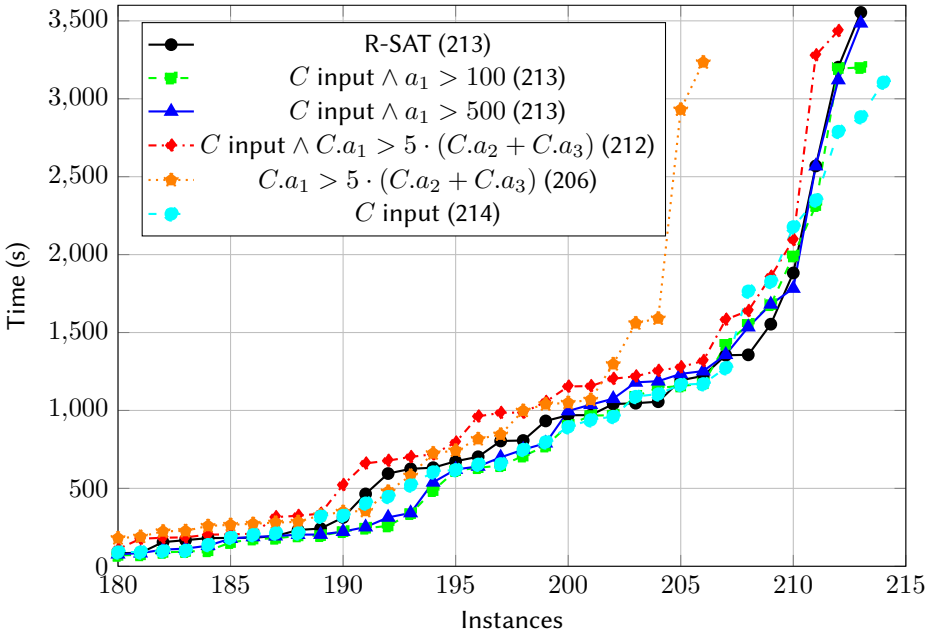

Figure 6: Runtime comparison of significant literal schemes on the OPT-LIN Track.

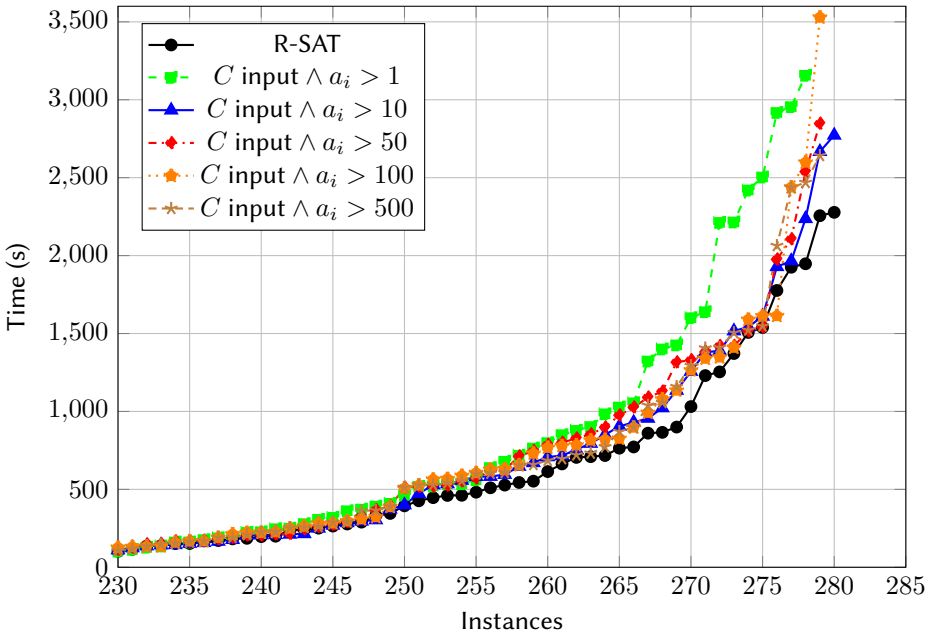

Figure 7: Cut-off value comparison for absolute size criterion on input constraints on the DEC-LIN Track.

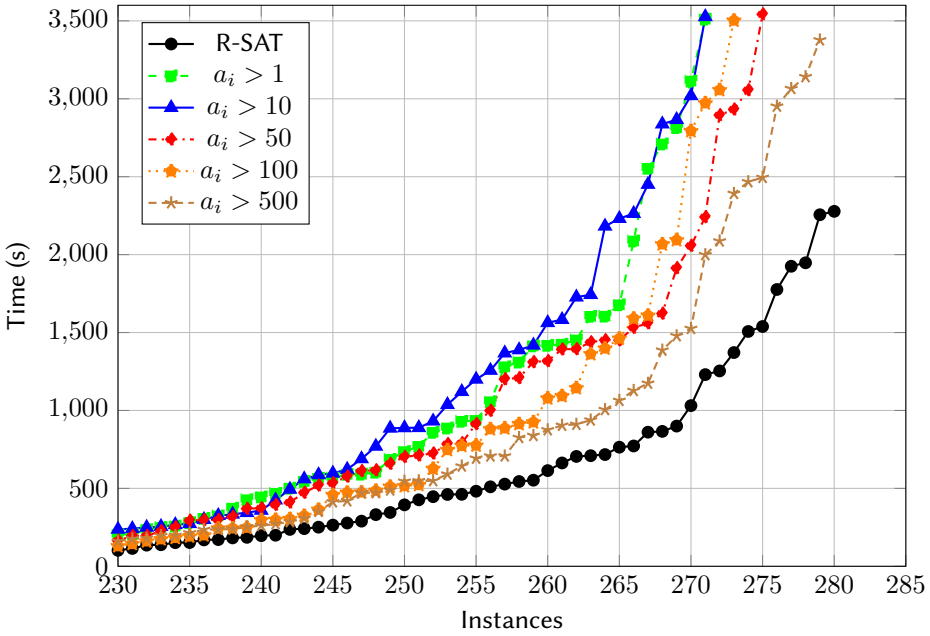

Figure 8: Cut-off value comparison for absolute size criterion on the DEC-LIN Track.

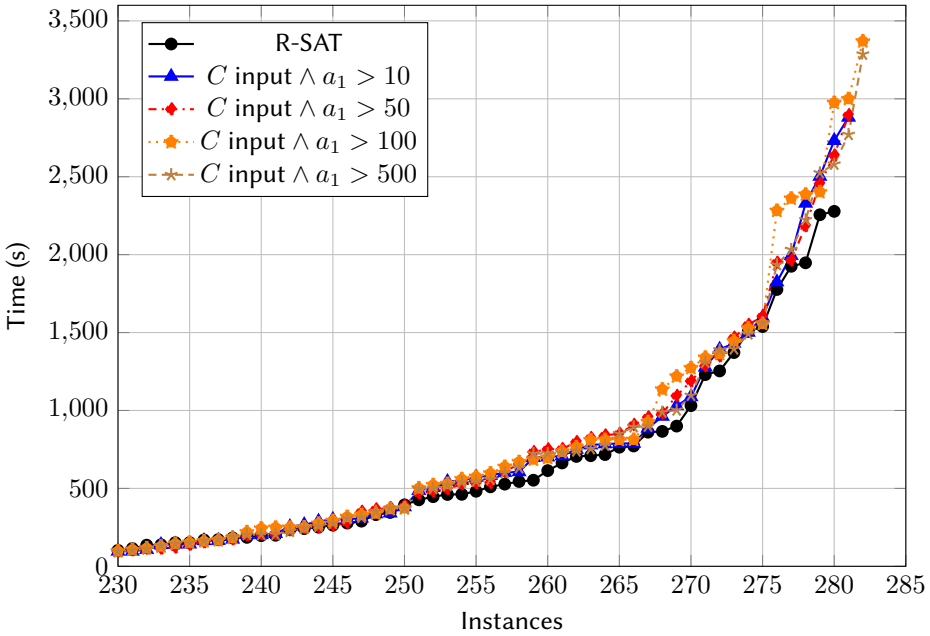

Figure 9: Cut-off value comparison for absolute size of maximum coefficient criterion on input constraints on the DEC-LIN Track. $c = 1$ is redundant since it would be equivalent to standard watched literal scheme.

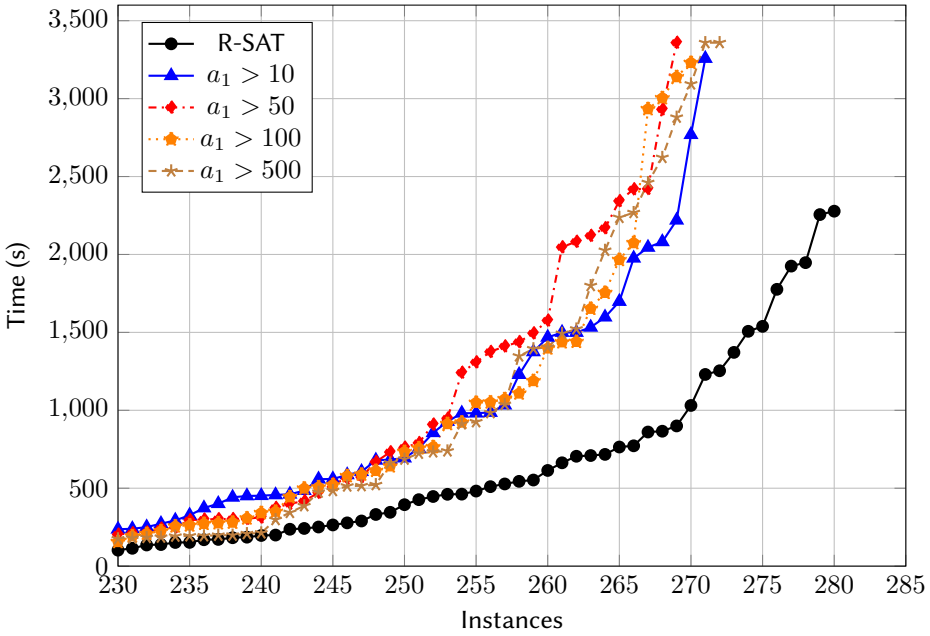

Figure 10: Cut-off value comparison for absolute size of maximum coefficient criterion on the DEC-LIN Track. $c = 1$ is redundant since it would be equivalent to the standard watched literal scheme.

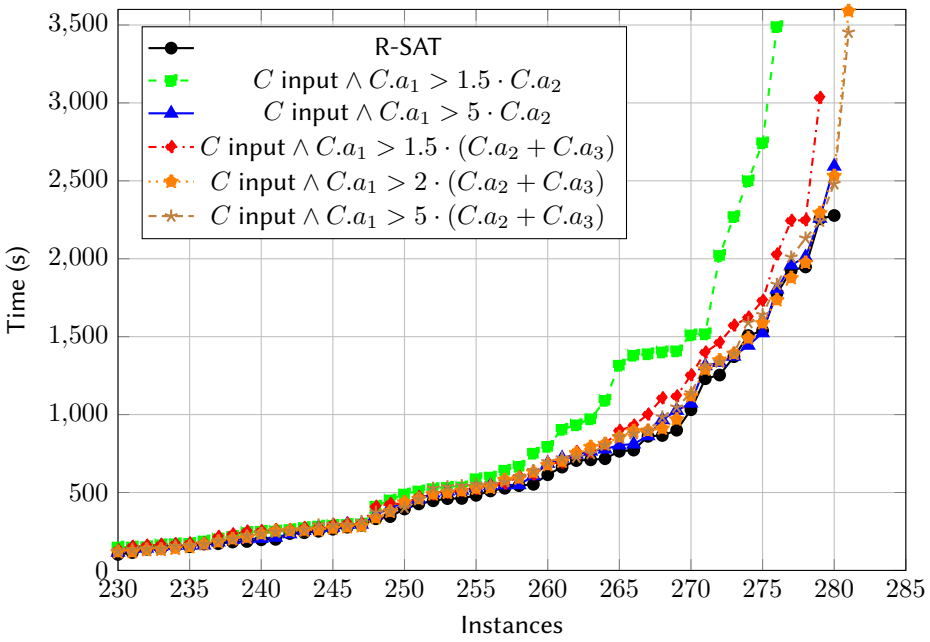

Figure 11: Cut-off value comparison for relative size criterion on input constraints on the DEC-LIN Track.

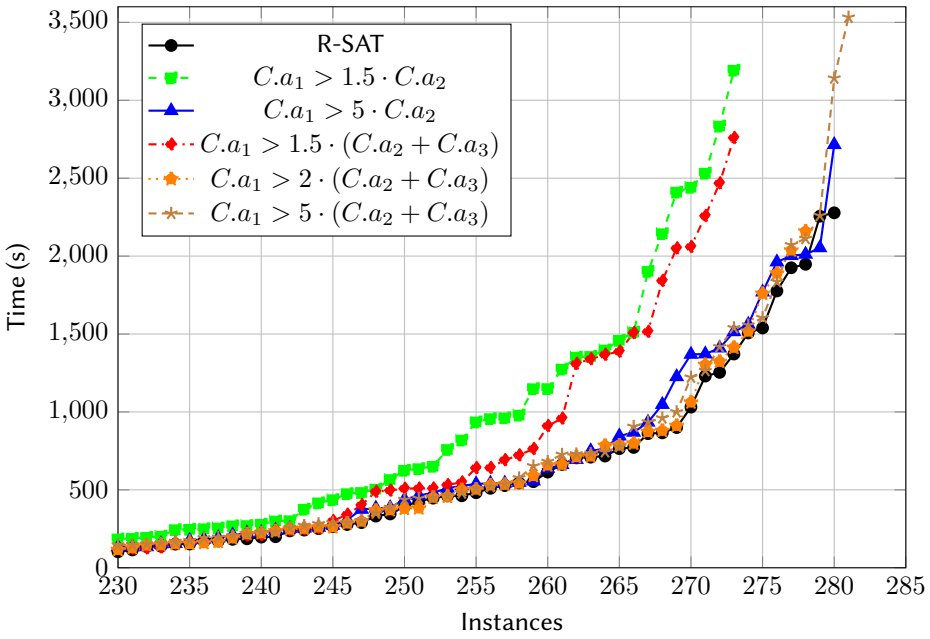

Figure 12: Cut-off value comparison for relative size criterion on the DEC-LIN Track.