Speeding Up Pseudo-Boolean Propagation

1st International Workshop on
Solving Linear Optimization Problems for Pseudo-Booleans and Yonder

Lund, Sweden

Robert Nieuwenhuis, Albert Oliveras, Enric Rodríguez-Carbonell, Rui Zhao

Technical University of Catalonia

Conflict-Driven PB Solving

- **SAT** technology has huge impact in diverse areas, but has some limits:
  1. No polynomial proofs for some well-known problems.
  2. No natural encoding of some constraints.
- Pseudo-Boolean(**PB**) solving is a remarkable alternative to SAT with:
  1. Exponentially stronger underlying proof system.
  2. More involved reasoning procedures.
- The dominant algorithm for PB solving is Conflict-Driven Pseudo-Boolean solving, where unit propagation is very time consuming.
- **GOAL OF THIS PAPER:** improve the performance of unit propagation by a more careful implementation (essentially no novel algorithms).

Propagation of Clauses

Two-watched literal scheme:

If two non-false literals exist, no conflict or propagation is possible.

Assignment (trail) ρ: $\bar{p}$

F	U	U	U	U
p	$\bar{q}$	r	$\bar{s}$	t
	$\uparrow$	$\uparrow$		

$p \vee \bar{q} \vee r \vee \bar{s} \vee t$

Watch List (ℓ): a list of clauses where ℓ is being watched.

Whenever ℓ becomes false, traverse its watch list checking for propagation.

Propagation of Clauses

Two-watched literal scheme:

If two non-false literals exists, no conflict or propagation is possible.

Assignment (trail) ρ: $\bar{p}$ q

F	F	U	U	U
p	$\bar{q}$	r	$\bar{s}$	t
		↑		↑

$p \vee \bar{q} \vee r \vee \bar{s} \vee t$

Watch List (ℓ): a list of clauses where ℓ is being watched.

Whenever ℓ becomes false, traverse its watch list checking for propagation.

Propagation of Clauses

Two-watched literal scheme:

If two non-false literals exists, no conflict or propagation is possible.

Assignment (trail) ρ: $\bar{p}$ q $\bar{r}$

F	F	F	U	U
p	$\bar{q}$	r	$\bar{s}$	t
			↑	↑

$p \vee \bar{q} \vee r \vee \bar{s} \vee t$

Watch List (ℓ): a list of clauses where ℓ is being watched.

Whenever ℓ becomes false, traverse its watch list checking for propagation.

Propagation of Clauses

Two-watched literal scheme:

If two non-false literals exists, no conflict or propagation is possible.

Assignment (trail) ρ: $\bar{p}\ q\ \bar{r}\ \bar{t}$

F	F	F	U	F
p	$\bar{q}$	r	$\bar{s}$	t
			↑	↑

Cannot find literals to watch ⇒ propagation of $\bar{s}$

$p \lor \bar{q} \lor r \lor \bar{s} \lor t$

Watch List (ℓ): a list of clauses where ℓ is being watched.

Whenever ℓ becomes false, traverse its watch list checking for propagation.

Propagation of Cardinality Constraints

For a constraint of the form $\ell_1 + \ell_2 + \cdots + \ell_n \geq k$, we have to watch $k + 1$ non-false literals.

Assignment (trail) ρ: s $\bar{t}$

$p + \bar{q} + r + \bar{s} + t \geq 3$ $\rightarrow$ *propagation not detected* !!

As for clauses:

- If watched r is assigned false, solver tries to watch a replacement.
- Unwatched literals cause no work.
- Possible propagation when not enough watches.

Propagation of PB Constraints

$4p + 3\bar{q} + 2r + 2\bar{s} + t \geq 10$

For a constraint of the form $C = \sum_i c_i \ell_i \geq d$,

The *slack* is the maximum number we can obtain from the $(lhs - rhs)$.

$$slack(C, \rho) = \left(\sum_{\ell_i \text{ not falsified by } \rho} c_i \right) - d$$

- Conflict found iff $slack < 0$.
- Undefined $c_i \ell_i$ will be propagated iff $slack - c_i < 0$.
- C is neither conflicting nor propagating if $slack \geq maxUndefCoeff(C)$.
  - To avoid updating $maxUndefCoeff(C)$, we use

    $slack \geq maxCoeff(C)$.

    This is less precise, but more efficient.

Propagation Example

$$slack(C,\rho) = \Big(\sum_{\ell_i \text{ not falsified by } \rho} c_i\Big) - d$$

$4p + 3\bar{q} + 2\mathbf{r} + 2\bar{s} + t \geq 10$

trail (ρ)	*slack*	*results*
$\rho = \emptyset$	2	$p, \bar{q}$ are propagated (more than one!)
⋮		
$\rho = p, \bar{q}, s, \bar{t}$	-1	conflicting (even if some lits are undefined)

For keeping **slack** up to date, we need to watch all literals in every constraint: **counter-based propagation**

RoundingSat PB Solver

RoundingSat is probably the fastest PB solver [Dev20]:

- Rigorous experimental evaluations of propagation mechanisms.
- Strong evidence of design decision.
- Proposal of a hybrid approach.

Counter-based Unit Propagation

Counter-based propagation in **RoundingSat**:

```
Function Counter-Propagation-in-RoundingSat(Watch w):
//                                        <ctrPtr,idx>
    Constraint ctr := w.ctrPtr

    if  ctr.isDeleted          then return      // always executed
    if  ctr.type ≠ PB-counter  then return      // always executed

    slack  :=  ctr.slack
    slack  -=  ctr[w.idx].coef                  // decrease slack

    if slack < 0        then return CONFLICT

    if slack < ctr.maxCoef  then      // possible propagation
        i := 0
        while i < ctr.size and slack < ctr[i].coef do
            if isUndef(ctr[i].lit) then propagate(ctr[i].lit)
            i := i + 1

    return OK
```

Watched-based Propagation of PB Constraints

As for clauses, the goal is to watch a hopefully small set of non-false lits that guarantee that no propagation/conflict exists.

For a constraint of the form $C = \sum_i c_i \ell_i \geq d$,

The *watchslack* is: the number we can obtain from the *watches(C) − rhs*.

$$\mathit{watchslack}(C, \rho) = \Big(\sum_{\substack{\ell_i \text{ not falsified by } \rho, \\ \ell_i \in \mathit{watches}(C)}} c_i \Big) - d$$

C is neither conflicting nor propagating if
$\mathit{watchslack} >= \mathit{maxUndefCoeff}(C)$ (or $\mathit{maxCoeff}(C)$ for simplicity)

Only when not enough watches:

- Conflict found iff $\mathit{watchslack} < 0$.
- Undefined $c_i \ell_i$ will be propagated iff $\mathit{watchslack} - c_i < 0$.

Watched-based Unit Propagation

```
Function Watched-Propagation-in-RoundingSat(Watch w):
    Constraint ctr := w.ctrPtr
    if ctr.isDeleted or ctr.isNot-PBWatched  then return

    wslack  := ctr.wslack
    wslack  -= ctr[w.idx].coef          // decrease watchslack

    i := 0
    while i < ctr.size and wslack < ctr.maxCoef do
        Lit lit := ctr[i].lit
        if ( lit_is_not_False and not_watched ) then
            watch_ith_lit()             // watch more literals
        i := i + 1

    if wslack ≥ ctr.maxCoef then
        unwatch_w.idxth_lit()      // unwatch current literal
        return OK

    if wslack < 0     then return CONFLICT
    i := 0 and check_for_propagation()   // possible propag.
```

Speeding Up Pseudo-Boolean Propagation

Precise Evaluation Methodology

Problem: even subtle changes in unit propagation have huge impact on search space traversal

- This might blur the improvements on unit propagation implementation.

Seed	Time(s)	Decs	Confs
0	174.3	3.7 *M*	314 *K*
1	247.6	2.1 *M*	473 *K*
2	166.5	3.9 *M*	300 *K*
3	162.9	2.6 *M*	271 *K*

Seed	Time(s)	Decs	Confs
4	182.2	1.4 *M*	270 *K*
5	224.7	2.5 *M*	423 *K*
6	248.5	2.6 *M*	463 *K*
7	148.7	1.8 *M*	230 *K*

- **Solution**: force solvers to explore the same search space by providing additional information in a **log** file, containing
  - Decision literals.
  - Lemmas to be learned.
  - Next cleanup/restart time, among others.

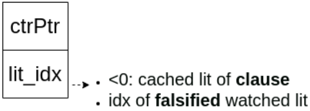
- **Experimental setting**: run ***RoundingSat*** solver on logs for around 100 benchmarks selected from *OPT-SMALLINT-LIN* category in the PB Competition 2016.

Speeding Up Pseudo-Boolean Propagation

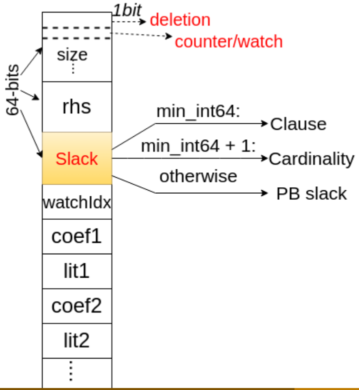
(1) Minimizing the Number of Constraint Loads

CaDiCaL: *"the cache line with the clause data is forced to be loaded here and thus this first memory access below is the real hot-spot of the solver"*.

Watch list element in ***RoundingSat****:*



Constraint:

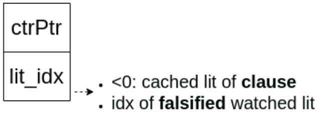


Counter-based Unit Propagation

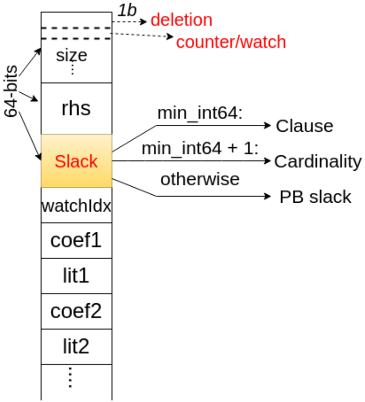
```
Function Counter-Propagation-in-RoundingSat(Watch w):
//                                          <ctrPtr,idx>
    Constraint ctr := w.ctrPtr

    if ctr.isDeleted              then return      // always executed
    if ctr.type ≠ PB-counter      then return      // always executed

    slack  :=  ctr.slack
    slack  -=  ctr[w.idx].coef               // decrease slack

    if slack < 0   then return CONFLICT

    if slack < ctr.maxCoef   then      // possible propagation
        i := 0
        while i < ctr.size and slack < ctr[i].coef do
            if isUndef(ctr[i].lit) then propagate(ctr[i].lit)
            i := i + 1

    return OK
```

(1) Minimizing the Number of Constraint Loads

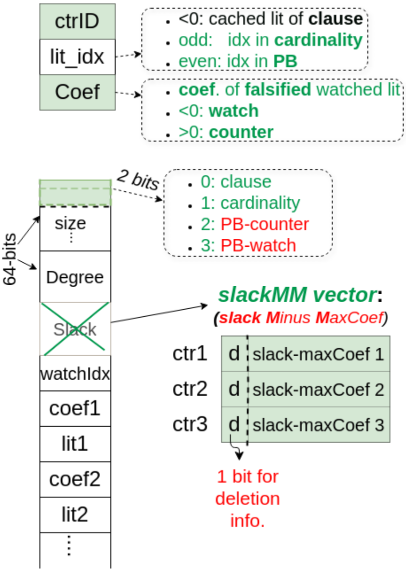
*Watch list element in **RoundingSat**:*



Constraint:



Our proposal:

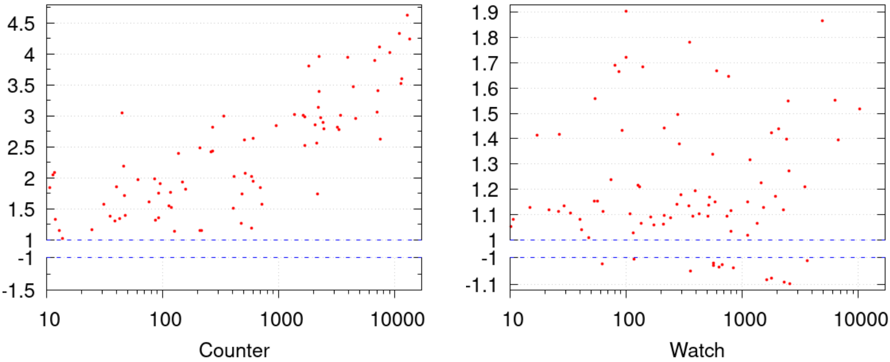


Improved Counter-based Propagation

```
Function Improved-Counter-based-Propagation(Watch w):
    id := w.identifier()
    if  slackMM[id].isDeleted       then return
    if  w.type() ≠ PB-counter       then return
    slackMC  :=  slackMM[id].slack
    slackMC  -=  w.coef                   // decrease slack
    if  slackMC < 0    then               // possible propagation
        Constraint ctr := constraints[id]   // loading constraint
        slack = slackMC + ctr.maxCoef
        if  slack < 0   then return CONFLICT
        i := 0
        while i < ctr.size and slack < ctr[i].coef do
            if isUndef(ctr[i].lit) then propagate(ctr[i].lit)
            i := i + 1
    return OK
```

For counters, the % of watch list elements that require loading the constraint is on average (6.29%), and median (1.26%). (less impact for watch.)

Impact of Constraint Loads



Step 1: Constraint loads vs. original procedure.

- Caption is in the form of "***Enhancement*** vs. ***Baseline***".
- Ratio is obained by "***large_time*** / ***small_time***".
- **Positive**: (enhancement is better), **Negative**: (baseline is better).

(2) Garbage Collection Frequency

Cleanup phase in RoundingSat:

1. Some constraints are marked as deleted, but they are **not yet removed** from the constraints database or the watch lists.
2. Apply **garbage collection** only if the **wasted memory** is large enough:
   - Constraints reallocation.
   - Rebuilding watch lists.

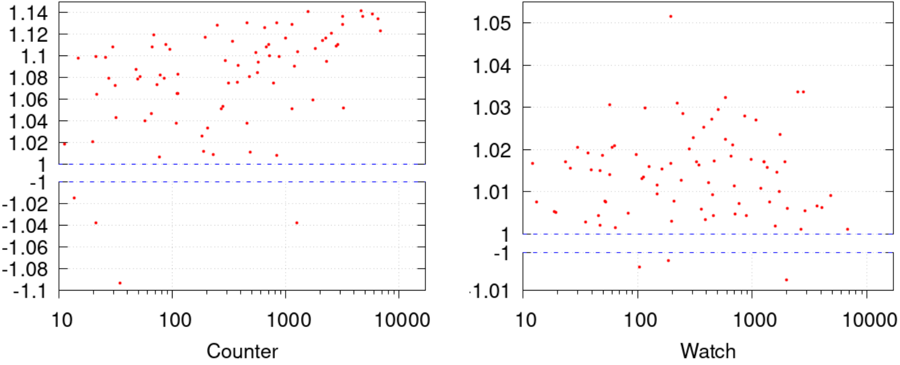
Suggestion: apply **garbage collection** in every cleanup phase.

- More compact constraint database and watch lists.
- No need the 1-bit for deletion in *slackMM* vector.
- No need to check deletion in propagation procedure.

Impact of Garbage Collection Frequency

Note that we accumulate the improvements in our implementation.

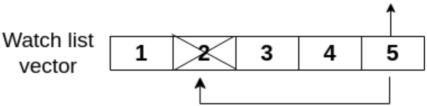
That is, we now compare constr. load vs constr. load + garbage collection.



Step 2: Garbage fixed vs. constraint loads.

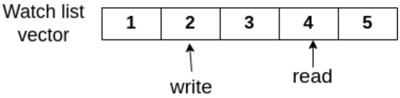
(3) Watchlists Elements Deletion

- Watch list elem. removal in RoundingSat done by moving last element. (needed for counter, due to the co-existence of clauses, cardinalities.)



Far locations, only one write.
better for watch

- In state-of-art SAT solvers (e.g. CaDiCaL, Kissat, MiniSAT), two pointers are kept, representing the reading and writing position, respectively.

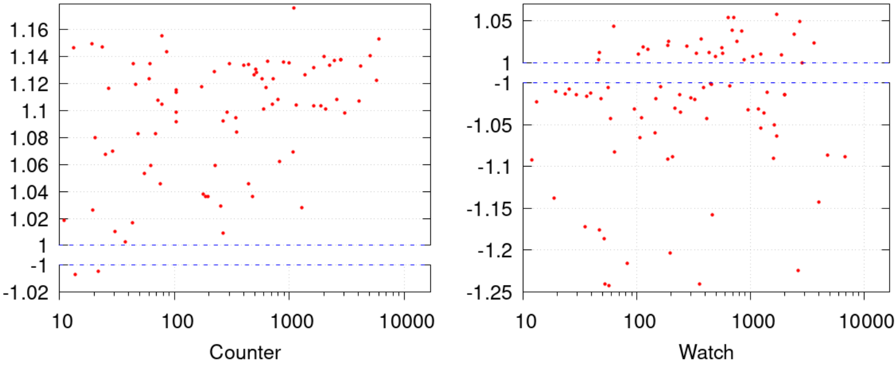


Close locations, but more writes.
better for counter

Table: Watch list length

Scheme	Average	Median
Counter	2365	564
Watched	203	71

Impact of WL Element Deletion

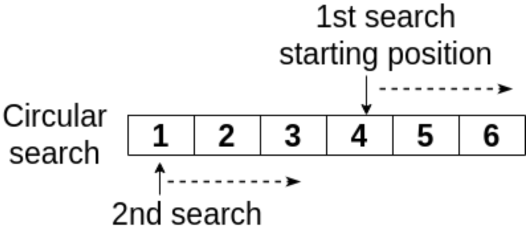


Step 3: WL element deletion vs. garbage fixed.

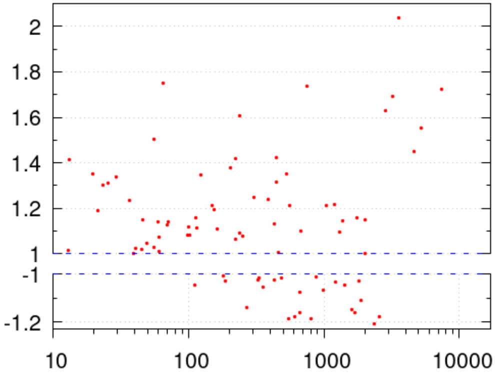
- Compared with bad points for watch, the improvement for counter is more.

(4) Circular Search For Watched Literals

- When a watched literal becomes false, we may need to watch more literals to guarantee *watchslack* $\geq$ *maxCoef*.
- Watching an **inactive literal** (rarely becomes false), saves a lot of work in propagation.
- Always searching for watched literals from the beginning makes it difficult for inactive literals at the end of a constraint to be watched.
- A solution is to search in a circular way, storing the last position tried and searching from the next position in the next time.
  (A 2nd search is required only if there is a backjump in between.)

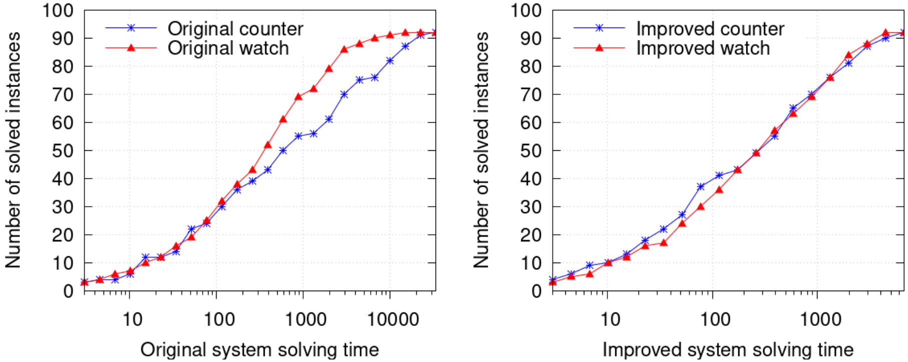


Impact of Circular Search



Step 4: Circular search vs. WL element deletion

Improved System vs. Original System



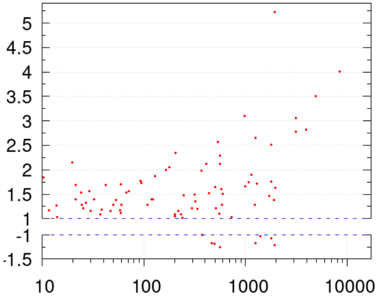
- In the end, the two approaches are very competitive, mainly because the number of constraints **to be loaded** is super similar.

Idea: Decide whether to use counter/watches **for each** constraint.

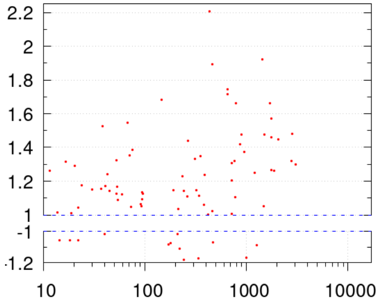
- A set of lits is watched to guarantee *watchslack* $\geq$ *maxCoef*
- **A threshold** $\alpha \in$ [0...1] is set. If the % of watched lits is smaller than α, we use watches (otherwise counters).

Our results:

1. Best threshold: 0.2 (original), 0.1 (improved).
2. Median of % watched constraints: 72% (original), 56% (improved).



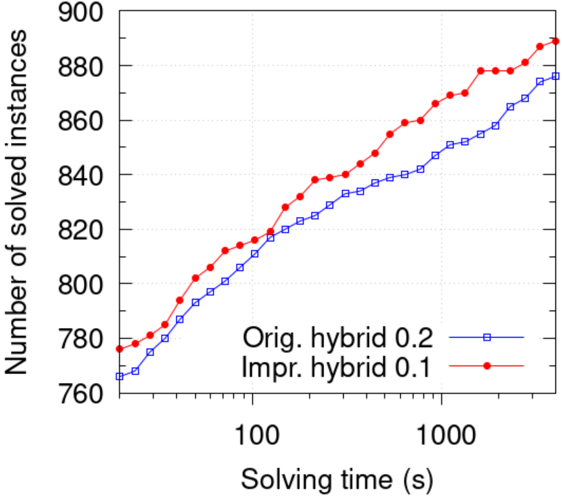
Improved hybrid (0.1) vs. original hybrid (0.2)



Improved hybrid (0.1) vs. improved VBS

Improvements in Overall Runtime

Experiments: **do not use logs**. Run *RoundingSat* on 1600 benchmarks in the category OPT-SMALLINT-LIN. (time limit: 3600s)



Speeding Up Pseudo-Boolean Propagation

Conclusions and Future Work

Conclusions:

- The novel methodology allow to precisely evaluate the propagation mechanisms.
- A more careful implementation has improved the propagation procedures used in RoundingSat.

Future work:

- Precisely analyze the impact of maintaining (an upper bound on) the maximum coefficient of undefined literals.
- Compute slacks with respect to the whole assignment. (Instead of only the current propagated trail)
- Enhance the hybrid method by dynamically analyzing the literal activities in a constraint.

Questions!

Improvements Step by Step