# Pseudo-Boolean proof trimming : early results and Why everything is (maybe) useless ?

**Arthur GONTIER**

May 24, 2024

## Table of Contents

1 Why trim

2 How to trim

3 Early results and improvements

4 Proof analysis ?

5 Conclusion

## Proof logging and trimming

| opb | pbp |
|---|---|
| $a + b \geq 2$ | `p 1 2 +` |
| $a + c \geq 1$ | `p 2 5 3 * +` |
| $c + d \geq 1$ | `u 0` $\geq 1$ |
| $\neg a \geq 1$ | |

Which part of the proof is actually useful ?

## Proof logging and trimming

| opb | pbp |
|---|---|
| $a + b \geq 2$ | `p 1 2 +` |
| $a + c \geq 1$ | `p 2 5 3 * +` |
| $c + d \geq 1$ | `u 0` $\geq 1$ |
| $\neg a \geq 1$ | |

Which part of the proof is actually useful ?

| opb | pbp |
|---|---|
| $a + b \geq 2$ | `u 0` $\geq 1$ |
| $\neg a \geq 1$ | |

Is it true in practice ?

How do we find that and at which cost ?

## Context

- Instances : newSIPbenchmarks/biochemicalReactions (http://perso.citi-lab.fr/csolnon/SIP.html)
- Solver : Glasgow Subgraph Solver
- Trimmer (dumb)
- Average constraint removed : 75% (mean over 2137 nontrivial instances)

## Table of Contents

1 Why trim

2 How to trim

3 Early results and improvements

4 Proof analysis ?

5 Conclusion

## Antecedent cone from contradiction

- *front* : equations with no antecedents yet
- *cone* : proof from *front* to contradiction

**Algorithm 1:** makesmol

```
1 $front \leftarrow$ firstUnitPropag or pol
2 while $front \neq \emptyset$ do
3     $eq \leftarrow$ pop $front$
4     if $eq \notin cone$ then
5         $cone \leftarrow cone \cup \{eq\}$
6         if $eq \in pbp$ then
7             $front \leftarrow front \cup$ antecedents of $eq$
8 return $cone$
```

## Pol & ia antecedents : read

| pbp | antecedants |
|---|---|
| `p 1 2 +` | $\{1, 2\}$ |
| `p 2 5 3 * +` | $\{2, 5\}$ |
| `ia 5 :` $a \geq 1$ | $\{5\}$ |

## Rup antecedents : we have to rup

| opb | pbp |
|---|---|
| $a + b \geq 2$ | `p 1 2 +` |
| $a + c \geq 1$ | `p 2 5 3 * +` |
| $c + d \geq 1$ | `u` $0 \geq 1$ |
| $\neg a \geq 1$ | |

## Rup antecedents : we have to rup

| opb | pbp |
|---|---|
| $a + b \geq 2$ | $2a + b + c \geq 3$ |
| $a + c \geq 1$ | $5a + b + 4c \geq 6$ |
| $c + d \geq 1$ | `u` $0 \geq 1$ |
| $\neg a \geq 1$ | |

## RUP ANTECEDENTS : WE HAVE TO RUP

| `opb` | `slack` | `pbp` | `slack` |
|---|---|---|---|
| $a + b \geq 2$ | $0$ | $2a + b + c \geq 3$ | $1$ |
| $a + c \geq 1$ | $1$ | $5a + b + 4c \geq 6$ | $4$ |
| $c + d \geq 1$ | $1$ | `u` $0 \geq 1$ | $-1$ |
| $\neg a \geq 1$ | $0$ | | |

## Rup antecedents : we have to rup

| opb | slack | pbp | slack |
|---|---|---|---|
| $a + b \geq 2$ | 0 | $2a + b + c \geq 3$ | 1 |
| $a + c \geq 1$ | 1 | $5a + b + 4c \geq 6$ | 4 |
| $c + d \geq 1$ | 1 | u $0 \geq 1$ | $-1$ |
| $\neg a \geq 1$ | 0 | | |

There are 3 ways to get $a \geq 1$ by rup here : $\{1, 4\}$ or $\{4, 5\}$ or $\{4, 6\}$

## Rup antecedents : we have to rup

| opb | slack | pbp | slack |
|---|---|---|---|
| $a + b \geq 2$ | $0$ | $2a + b + c \geq 3$ | $1$ |
| $a + c \geq 1$ | $1$ | $5a + b + 4c \geq 6$ | $4$ |
| $c + d \geq 1$ | $1$ | `u` $0 \geq 1$ | $-1$ |
| $\neg a \geq 1$ | $0$ | | |

There are 3 ways to get $a \geq 1$ by rup here : $\{1, 4\}$ or $\{4, 5\}$ or $\{4, 6\}$

Simple solution : compute slack from top to bot (proof size x2 if reverse)

## Solution antecedents

$\texttt{sol}\ \neg c\ d$

## Solution antecedents

$$\texttt{sol}\ \neg c\ d$$

Solutions have no antecedents.

**Safety measure**

Complete solution (full assignment)

$$\texttt{sol}\ \ a\ b\ \neg c\ d$$

## Reduction & dominance antecedents

## Reduction & dominance antecedents

Not yet

## Table of Contents

1 Why trim

2 How to trim

3 Early results and improvements

4 Proof analysis ?

5 Conclusion

## Efficiency : examples dumb RUP

| Instance | size | | time (s) | | trimmer | writer | parser |
|---|---|---|---|---|---|---|---|
| 7 | 6.438 MB | 693.9 KB | 2.158 | 0.267 | 0.436 | 0.375 | 3.93 |
| 8 | 3.116 MB | 406.8 KB | 1.309 | 0.245 | 0.346 | 0.078 | 2.155 |
| 10 | 3.525 MB | 565.4 KB | 1.499 | 0.345 | 0.686 | 0.094 | 2.484 |
| 17 | 10.32 MB | 576.8 KB | 5.11 | 0.376 | 0.514 | 0.302 | 10.45 |
| 21 | 6.354 MB | 704.1 KB | 2.569 | 2.533 | 20.26 | 0.147 | 4.388 |
| 25 | 4.968 MB | 1.364 MB | 3.097 | 0.83 | 0.995 | 0.228 | 5.484 |
| 26 | 10.27 MB | 1.081 MB | 5.329 | 0.601 | 1.126 | 0.293 | 9.986 |
| 27 | 3.423 MB | 553.1 KB | 2.223 | 0.375 | 0.235 | 0.088 | 4.178 |
| 29 | 10.17 MB | 582.8 KB | 6.596 | 0.392 | 0.68 | 0.242 | 13.93 |
| 31 | 3.178 MB | 480.0 KB | 2.0 | 0.346 | 0.178 | 0.126 | 3.601 |
| 35 | 5.298 MB | 757.7 KB | 2.076 | 0.37 | 0.679 | 0.207 | 3.216 |
| 37 | 3.554 MB | 596.1 KB | 1.594 | 12.28 | 65.52 | 0.149 | 2.53 |
| 41 | 9.438 MB | 965.9 KB | 5.444 | 0.545 | 1.028 | 0.25 | 9.723 |
| 44 | 2.525 MB | 402.8 KB | 1.2 | 0.241 | 0.452 | 0.109 | 1.819 |
| 46 | 7.714 MB | 863.2 KB | 3.42 | 0.402 | 0.81 | 0.325 | 6.565 |

## Deletions

- Use Deletions from original proof ? maybe sub-optimal
- We have the antecedent cone

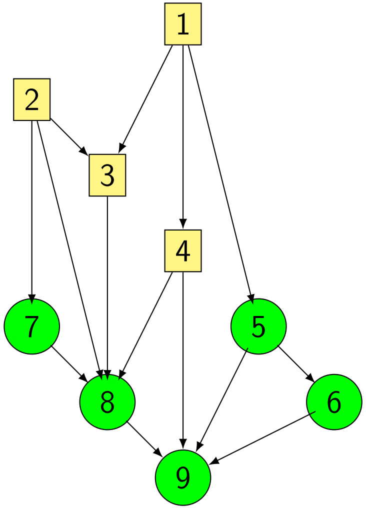
### Our idea

Delete the antecedents of c if c is their last child

### Results

- Proof are a little bit bigger

Can they be faster ?



## Deletions

- Use Deletions from original proof ? maybe sub-optimal
- We have the antecedent cone

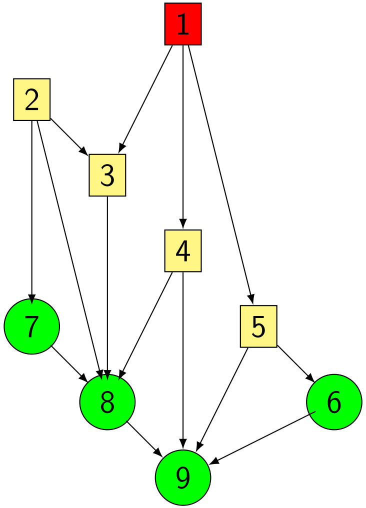
**Our idea**

Delete the antecedents of c if c is their last child

**results**

- Proof are a little bit bigger

Can they be faster ?



## Deletions

- Use Deletions from original proof ? maybe sub-optimal
- We have the antecedent cone

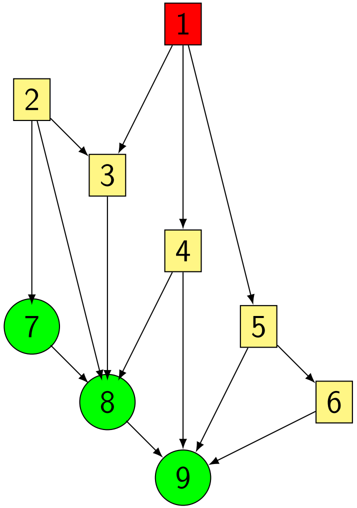
### Our idea

Delete the antecedents of c if c is their last child

### Results

- Proof are a little bit bigger

Can they be faster ?



## Deletions

- Use Deletions from original proof ? maybe sub-optimal
- We have the antecedent cone

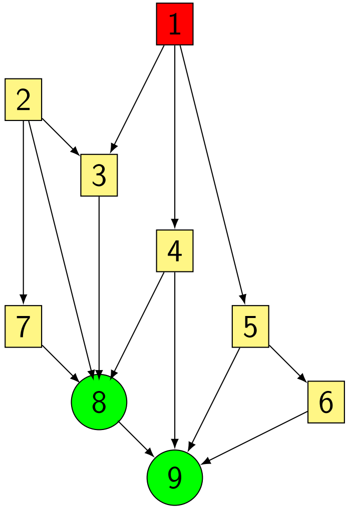
**Our idea**

Delete the antecedents of c if c is their last child

**RESULTS**

- Proof are a little bit bigger

Can they be faster ?



## Deletions

- Use Deletions from original proof ? maybe sub-optimal
- We have the antecedent cone

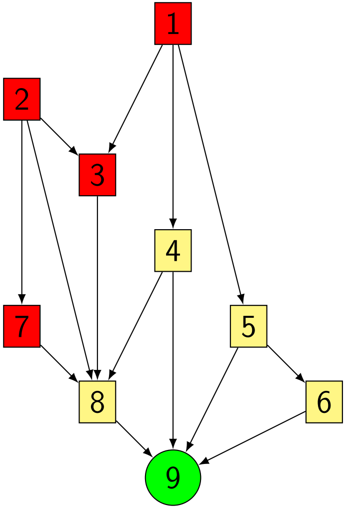
### Our idea

Delete the antecedents of c if c is their last child

### Results

- Proof are a little bit bigger

Can they be faster ?



## Efficiency : examples dumb RUP with deletions

| Instance | size | | time (s) | | trimmer | writer | parser |
|---|---|---|---|---|---|---|---|
| 7 | 6.438 MB | 817.1 KB | 2.153 | 0.373 | 0.433 | 0.397 | 3.912 |
| 8 | 3.116 MB | 474.5 KB | 1.323 | 0.277 | 0.341 | 0.101 | 2.14 |
| 10 | 3.525 MB | 662.5 KB | 1.457 | 0.373 | 0.686 | 0.333 | 2.366 |
| 17 | 10.32 MB | 654.1 KB | 5.103 | 0.45 | 0.53 | 0.337 | 10.22 |
| 21 | 6.354 MB | 814.0 KB | 2.374 | 1.387 | 18.5 | 0.374 | 4.152 |
| 25 | 4.968 MB | 1.63 MB | 2.7 | 1.054 | 0.857 | 0.319 | 4.93 |
| 26 | 10.27 MB | 1.254 MB | 4.935 | 0.724 | 1.1 | 0.659 | 9.523 |
| 27 | 3.423 MB | 643.4 KB | 2.059 | 0.469 | 0.233 | 0.168 | 3.906 |
| 29 | 10.17 MB | 664.6 KB | 6.317 | 0.457 | 0.664 | 0.32 | 13.23 |
| 31 | 3.178 MB | 556.3 KB | 1.851 | 0.401 | 0.184 | 0.295 | 3.54 |
| 35 | 5.298 MB | 887.0 KB | 1.955 | 0.479 | 0.667 | 0.325 | 3.069 |
| 37 | 3.554 MB | 669.4 KB | 1.481 | 2.137 | 61.25 | 1.551 | 2.46 |
| 41 | 9.438 MB | 1.123 MB | 4.593 | 0.619 | 0.968 | 0.35 | 9.165 |
| 44 | 2.525 MB | 472.5 KB | 1.1 | 0.286 | 0.427 | 0.082 | 1.711 |
| 46 | 7.714 MB | 1.017 MB | 3.097 | 0.494 | 0.765 | 0.404 | 5.437 |

## Efficiency : What is taking time

- Parser : Pol

  (All pol must be computed to compute slack) $\Longrightarrow$ lazy pol generation ?
- Trimmer : Rup

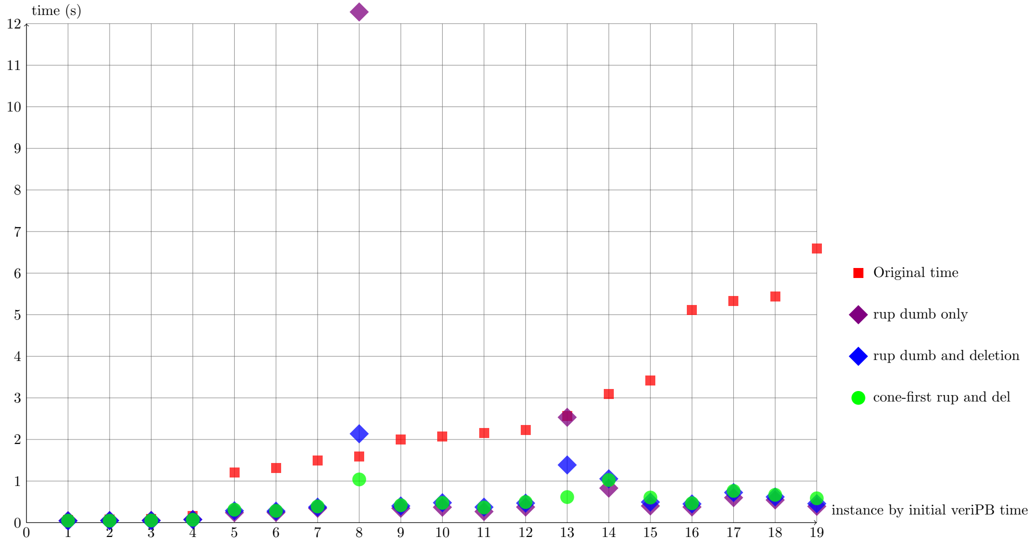
  (worst case : recompute all slacks)

$\Longrightarrow$ Try to reuse constraints already in the cone first ?

## Efficiency : examples first cone rup

| Instance | size | | time (s) | | trimmer | writer | parser |
|---|---|---|---|---|---|---|---|
| 7 | 6.438 MB | 817.1 KB | 2.216 | 0.364 | 0.451 | 0.295 | 4.061 |
| 8 | 3.116 MB | 472.0 KB | 1.355 | 0.283 | 0.172 | 0.109 | 2.167 |
| 10 | 3.525 MB | 667.5 KB | 1.5 | 0.39 | 0.32 | 0.202 | 2.521 |
| 17 | 10.32 MB | 654.1 KB | 5.141 | 0.467 | 0.523 | 0.331 | 10.29 |
| 21 | 6.354 MB | 749.0 KB | 2.477 | 0.615 | 2.297 | 0.151 | 4.436 |
| 25 | 4.968 MB | 1.63 MB | 2.686 | 1.025 | 0.853 | 0.368 | 4.993 |
| 26 | 10.27 MB | 1.254 MB | 4.951 | 0.763 | 1.145 | 0.711 | 9.649 |
| 27 | 3.423 MB | 643.4 KB | 2.084 | 0.496 | 0.234 | 0.189 | 4.054 |
| 29 | 10.17 MB | 664.6 KB | 6.911 | 0.585 | 0.692 | 0.724 | 13.73 |
| 31 | 3.178 MB | 556.3 KB | 1.951 | 0.419 | 0.177 | 0.171 | 3.51 |
| 35 | 5.298 MB | 876.3 KB | 2.021 | 0.475 | 0.456 | 0.2 | 3.214 |
| 37 | 3.554 MB | 497.5 KB | 1.534 | 1.039 | 11.28 | 0.829 | 2.494 |
| 41 | 9.438 MB | 1.123 MB | 5.316 | 0.669 | 0.981 | 0.376 | 9.628 |
| 44 | 2.525 MB | 460.0 KB | 1.171 | 0.311 | 0.214 | 0.115 | 1.839 |
| 46 | 7.714 MB | 1.017 MB | 3.536 | 0.604 | 0.786 | 0.219 | 5.786 |

## Early results summary



## Smarter Rup

### What we do now

- Use ctrs with small id first.
- Maximize ctr reuse (first cone rup like DRATtrim)

## Smarter Rup

### What we do now

- Use ctrs with small id first.
- Maximize ctr reuse (first cone rup like DRATtrim)

### Ideas

- Prioritize ctrs that are close together ?
- Prioritize ctrs that are nearly closed ?
- Maybe use rupCheckWithHints ?
- Any more ideas ?

## Parallel problems

| Instance | size | | time (s) x4 | | trimmer x2 | writer x4 | parser x4 |
|---|---|---|---|---|---|---|---|
| 7 | 6.438 MB | 817.1 KB | 11.7 | 2.385 | 1.798 | 1.418 | 16.77 |
| 8 | 3.116 MB | 472.0 KB | 5.989 | 1.134 | 0.516 | 0.773 | 9.404 |
| 10 | 3.525 MB | 667.5 KB | 6.968 | 1.305 | 0.907 | 1.136 | 9.195 |
| 17 | 10.32 MB | 654.1 KB | 26.44 | 1.271 | 1.112 | 1.497 | 42.15 |
| 21 | 6.354 MB | 749.0 KB | 13.13 | 1.916 | 6.719 | 1.256 | 19.14 |
| 25 | 4.968 MB | 1.63 MB | 13.78 | 4.155 | 3.021 | 3.14 | 22.89 |
| 26 | 10.27 MB | 1.254 MB | 19.71 | 1.02 | 1.365 | 0.7 | 19.34 |
| 27 | 3.423 MB | 643.4 KB | 2.871 | 0.635 | 0.29 | 0.198 | 5.27 |
| 29 | 10.17 MB | 664.6 KB | 31.66 | 0.867 | 1.077 | 1.464 | 45.75 |
| 31 | 3.178 MB | 556.3 KB | 9.678 | 2.303 | 0.582 | 1.196 | 15.37 |
| 35 | 5.298 MB | 876.3 KB | 10.19 | 2.728 | 1.571 | 1.396 | 13.47 |
| 37 | 3.554 MB | 497.5 KB | 7.714 | 2.136 | 25.08 | 1.732 | 10.14 |
| 41 | 9.438 MB | 1.123 MB | 24.65 | 2.635 | 2.882 | 2.084 | 35.83 |
| 44 | 2.525 MB | 460.0 KB | 5.73 | 1.412 | 0.635 | 1.619 | 7.504 |
| 46 | 7.714 MB | 1.017 MB | 15.48 | 2.179 | 2.522 | 2.414 | 21.59 |

## Table of Contents

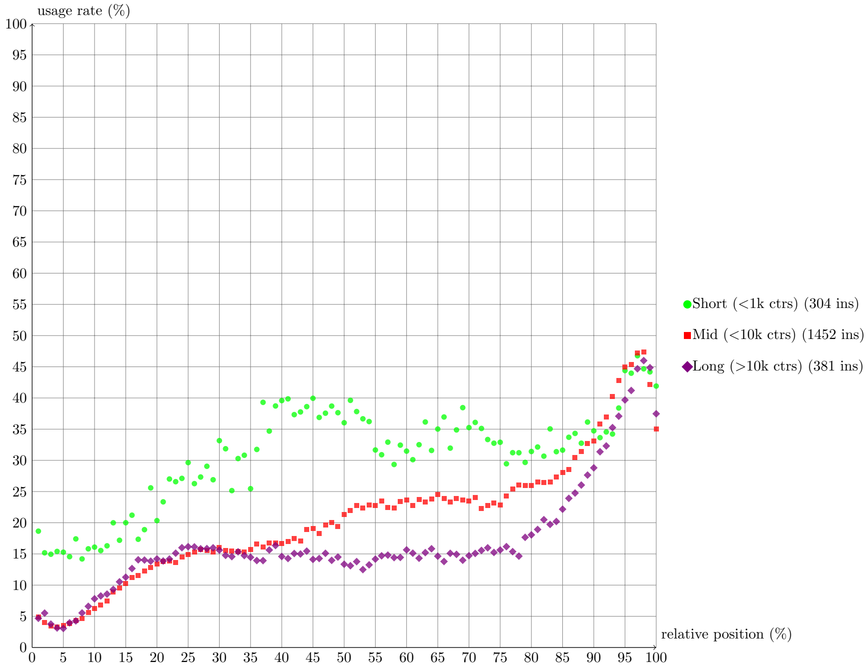
1 Why trim

2 How to trim

3 Early results and improvements

4 Proof analysis ?

5 Conclusion


usage rate (%)
relative position (%)
0
5
10
15
20
25
30
35
40
45
50
55
60
65
70
75
80
85
90
95
100
Short (<1k ctrs) (304 ins)
Mid (<10k ctrs) (1452 ins)
Long (>10k ctrs) (381 ins)

## Table of Contents

1 Why trim

2 How to trim

3 Early results and improvements

4 Proof analysis ?

5 Conclusion

## Conclusion

### What we have

- A dumb trimmer for the Glasgow subgraph solver proofs
- Early rup and del comparisons
- Early proof analysis

## Conclusion

### What we have

- A dumb trimmer for the Glasgow subgraph solver proofs
- Early rup and del comparisons
- Early proof analysis

### What is next ?

- Find and compare more rup algorithms
- Try to identify which constraints are useful in the solver (tags ?)
- Tests on more problems
- Think about red and dom
- Any more ideas ?

## Question from a friend of mine who hate making parsers

Grammar ?