Efficient All-UIP Learned Clause Minimization

http://fmv.jku.at/sat_shrinking

Mathias Fleury and Armin Biere

SAT 2021

JOHANNES KEPLER UNIVERSITÄT LINZ
LIT AI Laboratory
FMV
FWF
Der Wissenschaftsfonds.

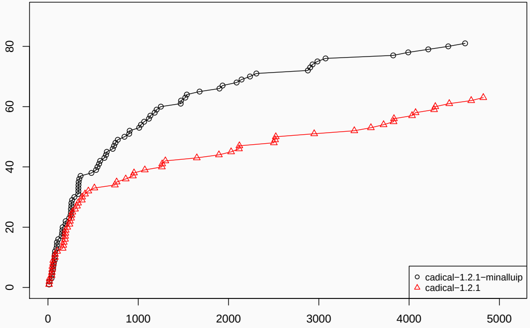
Introduction

Introduction

- SAT solvers analyze conflicts to derive the deduced clause C...
- ... that can be shortened by the standard minimization algorithm $C' \subseteq C$
- ... or even more $|C''| \leq |C|$ (all-UIP technique)

SAT Competition 2020

[F&B'20] and [Hickey et al. SAT Comp'20] won the planning track.

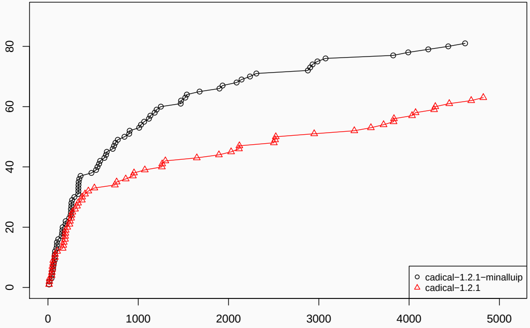


Cactus plot of CaDiCaL on the planning track

Don't worsen the LBD score!

SAT Competition 2020

[F&B'20] and [Hickey et al. SAT Comp'20] won the planning track.



Cactus plot of CaDiCaL on the planning track

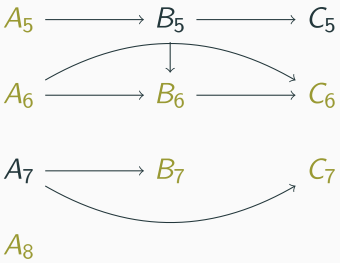
Don't worsen the LBD score!

Introduction

- SAT solvers analyze conflicts to derive the deduced clause C...
- ... that can be shortened by the standard minimization algorithm $C' \subseteq C$
  Contribution: completeness of the algorithm, earlier breaking conditions
- ... or even more $|C''| \leq |C|$ (all-UIP technique)
  Contribution: simpler unconditional implementation, use 1-UIP

Minimization

Implication graph



Implication graph

Deduced clause: $\neg A_5 \vee \neg A_6 \vee \neg B_6 \vee \neg C_6 \vee \neg B_7 \vee \neg C_7 \vee \neg A_8$

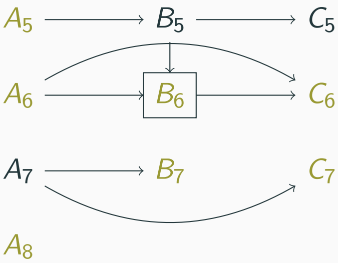
Standard Minimization Algorithm

Algorithm [Sörensson&Biere, SAT'09]: for any literal

1. replace the literals by all the incoming arrows, recursively
2. if final clause is shorter: literal is redundant

Shorten as much as possible.

Minimization Example

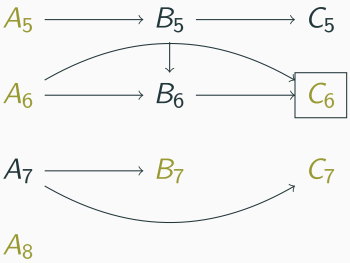


Implication graph

Deduced clause:

$\neg A_5 \vee \neg A_6 \vee \neg B_6 \vee \neg C_6 \vee \neg B_7 \vee \neg C_7 \vee \neg A_8$

Minimization Example

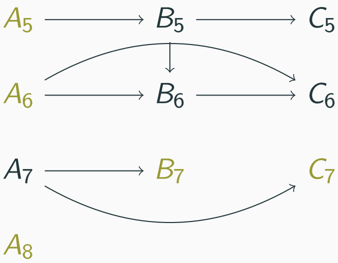


Implication graph

Deduced clause:

$\neg A_5 \vee \neg A_6 \vee \neg \cancel{B_6} \vee \neg C_6 \vee \neg B_7 \vee \neg C_7 \vee \neg A_8$

Minimization Example



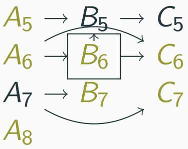
Implication graph

Deduced clause:

$\neg A_5 \vee \neg A_6 \vee \neg \cancel{B_6} \vee \neg \cancel{C_6} \vee \neg B_7 \vee \neg C_7 \vee \neg A_8$

Definition

M = trail = nodes in graph
N = reasons = edges in graph
C = conflict = brown literals



Definition (Trail redundancy)

Given $\neg L \in M$, $M \vDash \neg C$, N the set of all reasons in the trail
L is trail redundant iff $N \cup \neg C \setminus \{L\} \vDash \neg L$.

Theorem (Completeness)

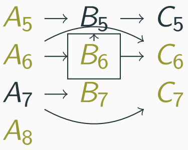
All trail redundant literals of C can be removed

Definition

M = trail = nodes in graph

N = reasons = edges in graph

C = conflict = brown literals



Definition (Trail redundancy)

Given $\neg L \in M$, $M \models \neg C$, N the set of all reasons in the trail

L is trail redundant iff $N \cup \neg C \setminus \{L\} \models \neg L$.

Theorem (Completeness)

All trail redundant literals of C can be removed

Completeness

Theorem

The minimization algorithm computes exactly the trail redundant literals.

Poison Criteria

Theorem (When can I stop?)

1. *Literals with a decision level not in the deduced clause are irredundant.* *classical argument*
2. *Literal appearing on a level before any other literal of the deduced clause are irredundant.* CaDiCaL *novelty*
3. *Literals that are alone on a level are irredundant.* *Don Knuth*

Poison Criteria

Theorem (When can I stop?)

1. *Literals with a decision level not in the deduced clause are irredundant.* *classical argument*
2. *Literal appearing on a level before any other literal of the deduced clause are irredundant.* CaDiCaL *novelty*
3. *Literals that are alone on a level are irredundant.* *Don Knuth*

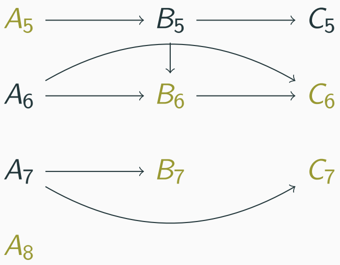
Poison Criteria

Theorem (When can I stop?)

1. *Literals with a decision level not in the deduced clause are irredundant.* *classical argument*
2. *Literal appearing on a level before any other literal of the deduced clause are irredundant.* CaDiCaL *novelty*
3. *Literals that are alone on a level are irredundant.* *Don Knuth*

Shrinking

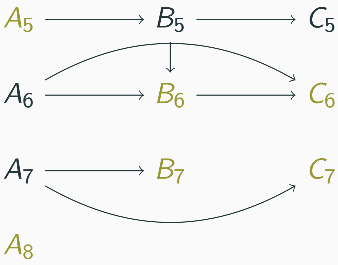
Implication graph



Implication graph

No minimization is possible

Shrinking Example



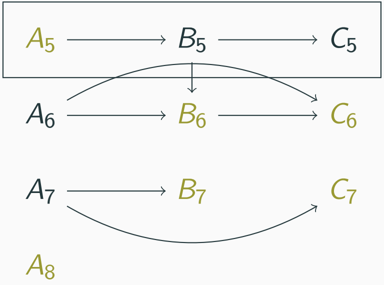
Implication graph

Our algorithm:

1. From smallest level to top:
   - Go up the arrows ...
   - ... unless a irredundant literal of lower level is added, then minimize
   - ... if successful, update minimization cache

Deduced clause: $\neg A_5 \vee \neg B_6 \vee \neg C_6 \vee \neg B_7 \vee \neg C_7 \vee \neg A_8$

Shrinking Example



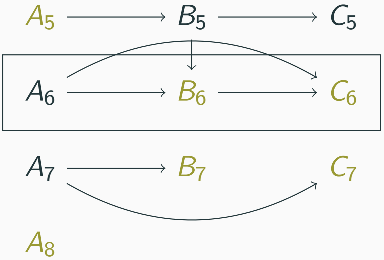
Implication graph

Our algorithm:

1. From smallest level to top:
   - Go up the arrows ...
   - ... unless a irredundant literal of lower level is added, then minimize
   - ... if successful, update minimization cache

Deduced clause: $\neg A_5 \lor \neg B_6 \lor \neg C_6 \lor \neg B_7 \lor \neg C_7 \lor \neg A_8$

Shrinking Example



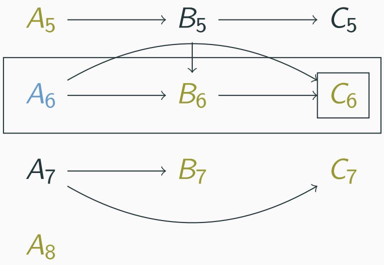
Implication graph

Our algorithm:

1. From smallest level to top:
   - Go up the arrows ...
   - ... unless a irredundant literal of lower level is added, then minimize
   - ... if successful, update minimization cache

Deduced clause: $\neg A_5 \vee \neg B_6 \vee \neg C_6 \vee \neg B_7 \vee \neg C_7 \vee \neg A_8$

Shrinking Example



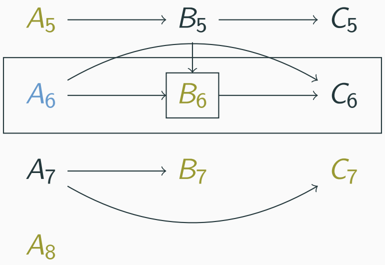
Implication graph

Our algorithm:

1. From smallest level to top:
   - Go up the arrows ...
   - ... unless a irredundant literal of lower level is added, then minimize
   - ... if successful, update minimization cache

Deduced clause: $\neg A_5 \vee \neg B_6 \vee \neg C_6 \vee \neg B_7 \vee \neg C_7 \vee \neg A_8$

Shrinking Example



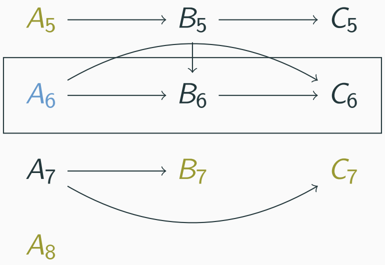
Implication graph

Our algorithm:

1. From smallest level to top:
   - Go up the arrows ...
   - ... unless a irredundant literal of lower level is added, then minimize
   - ... if successful, update minimization cache

Deduced clause: $\neg A_5 \vee \neg B_6 \vee \neg C_6 \vee \neg B_7 \vee \neg C_7 \vee \neg A_8$

Shrinking Example



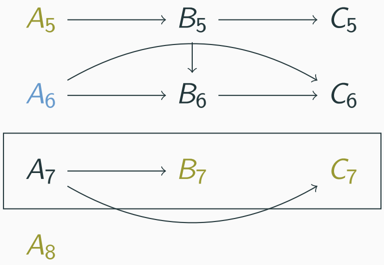
Implication graph

Our algorithm:

1. From smallest level to top:
   - Go up the arrows ...
   - ... unless a irredundant literal of lower level is added, then minimize
   - ... if successful, update minimization cache

Deduced clause: $\neg A_5 \vee$
$\neg \not{B_6} \vee \neg \not{C_6} \vee \neg B_7 \vee \neg C_7 \vee$
$\neg A_8 \vee \neg A_6$

Shrinking Example



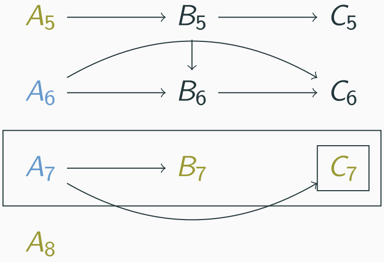
Implication graph

Our algorithm:

1. From smallest level to top:
    - Go up the arrows ...
    - ... unless a irredundant literal of lower level is added, then minimize
    - ... if successful, update minimization cache

Deduced clause: $\neg A_5 \vee$
$\neg \cancel{B_6} \vee \neg \cancel{C_6} \vee \neg B_7 \vee \neg C_7 \vee$
$\neg A_8 \vee \neg A_6$

Shrinking Example



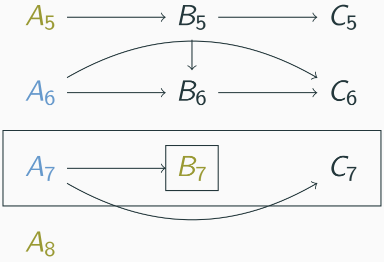
Implication graph

Our algorithm:

1. From smallest level to top:
   - Go up the arrows ...
   - ... unless a irredundant literal of lower level is added, then minimize
   - ... if successful, update minimization cache

Deduced clause: $\neg A_5 \lor$ $\neg B_6 \lor \neg C_6 \lor \neg B_7 \lor \neg C_7 \lor$ $\neg A_8 \lor \neg A_6$

Shrinking Example



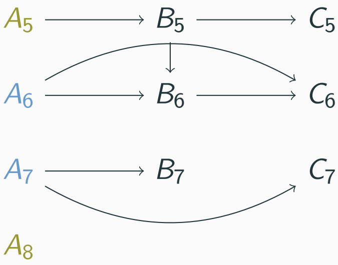
Implication graph

Our algorithm:

1. From smallest level to top:
   - Go up the arrows ...
   - ... unless a irredundant literal of lower level is added, then minimize
   - ... if successful, update minimization cache

Deduced clause: $\neg A_5 \lor \neg \cancel{B_6} \lor \neg \cancel{C_6} \lor \neg B_7 \lor \neg C_7 \lor \neg A_8 \lor \neg A_6$

Shrinking Example



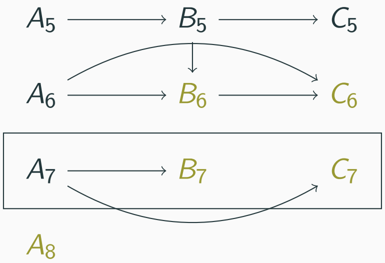
Implication graph

Our algorithm:

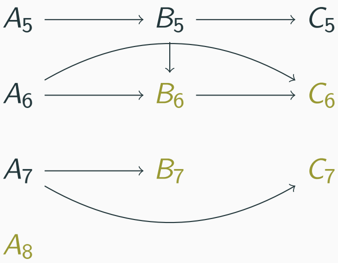
1. From smallest level to top:
   - Go up the arrows ...
   - ... unless a irredundant literal of lower level is added, then minimize
   - ... if successful, update minimization cache

Deduced clause: $\neg A_5 \vee \neg \cancel{B_6} \vee \neg \cancel{C_6} \vee \neg \cancel{B_7} \vee \neg \cancel{C_7} \vee \neg A_8 \vee \neg A_6 \vee \neg A_7$

Behavior Difference

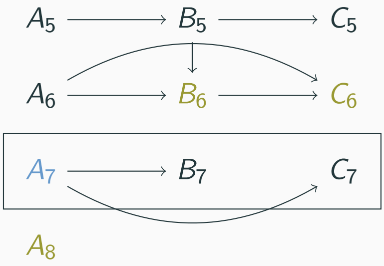


Example for algorithm from [F&B'20]

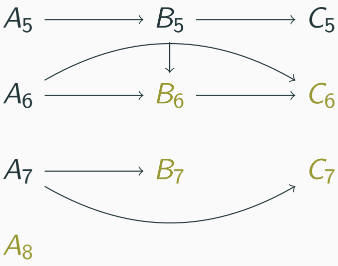


Example for our algorithm

Behavior Difference

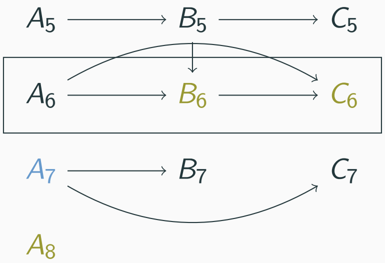


Example for algorithm from [F&B'20]

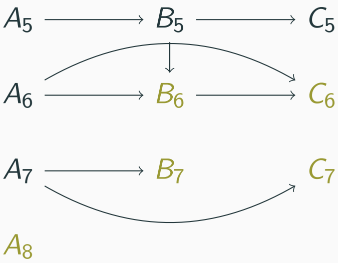


Example for our algorithm

Behavior Difference

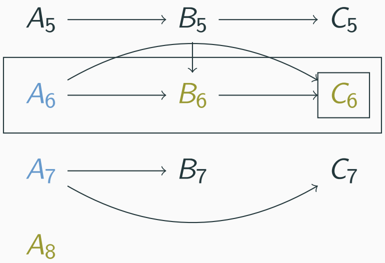


Example for algorithm from [F&B'20]

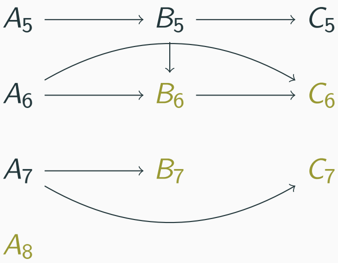


Example for our algorithm

Behavior Difference

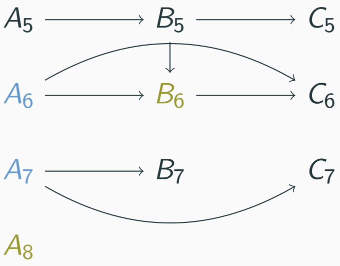


Example for algorithm from [F&B'20]

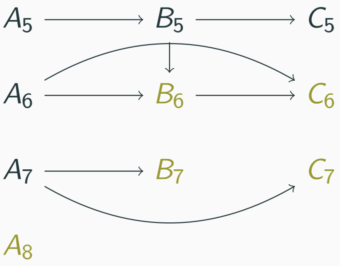


Example for our algorithm

Behavior Difference

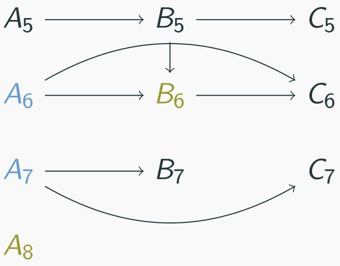


Example for algorithm from [F&B'20]

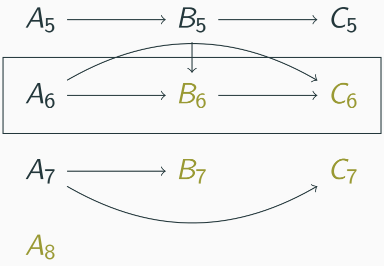


Example for our algorithm

Behavior Difference

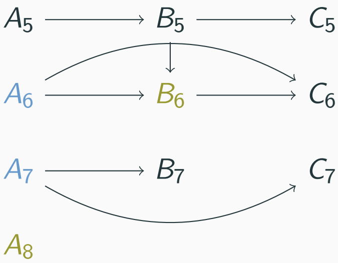


Example for algorithm from [F&B'20]

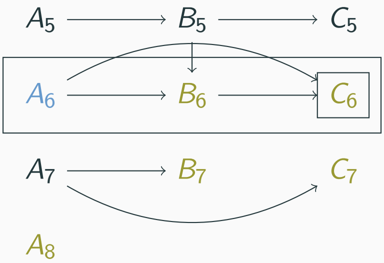


Example for our algorithm

Behavior Difference

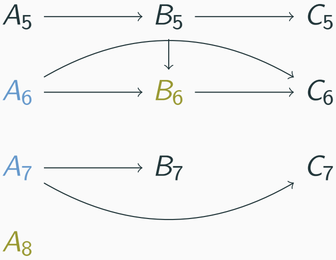


Example for algorithm from [F&B'20]

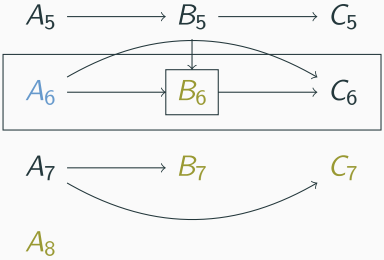


Example for our algorithm

Behavior Difference

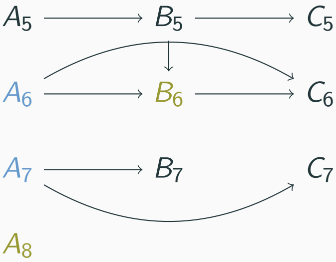


Example for algorithm from [F&B'20]

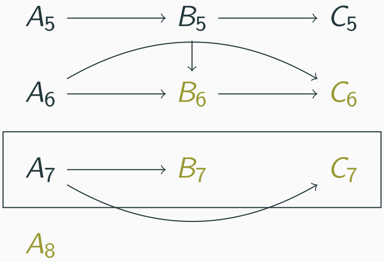


Example for our algorithm

Behavior Difference

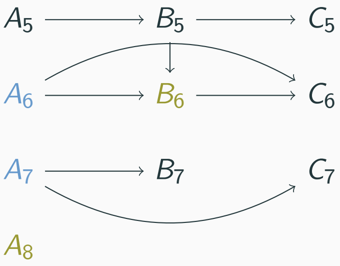


Example for algorithm from [F&B'20]

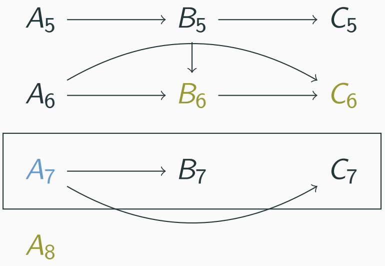


Example for our algorithm

Behavior Difference



Example for algorithm from [F&B'20]



Example for our algorithm

Trail Redundancy

Theorem

Redundant literal remain redundant after shrinking.

Theorem

Minimization cache for lower levels remains correct in our algorithm.

Algorithm Comparison

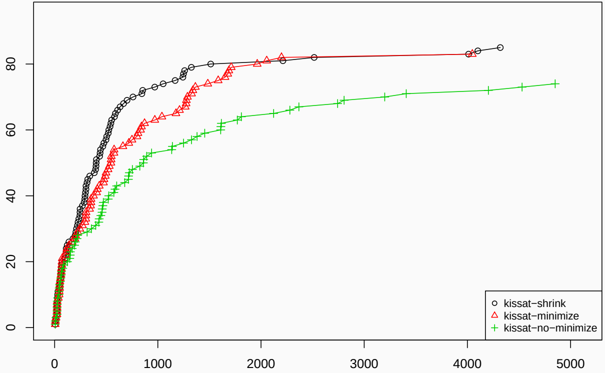
		F&B [F&B'20]		Shrinking (this)
Conditional	✗	too expensive	✓	cheap enough
Always smaller	✗	resulting clause discarded	✓	
Minimization	✗	separate	✓	combined
Implementations	✗	one strategy only (`min-alluip`, SAT Comp 2020)	✓	CaDiCaL, Kissat, Satch /arminbiere

Implementation

Kissat on the SAT Competition 2020 benchmarks

Track	Config.	Solved	PAR-2	Average clause size
Main (400 CNFs)	shrink	**270**	**1 561 735**	**46**
	mini	267	1 566 688	110
	no-mini	235	1 891 872	183
Planning (200 CNFs)	shrink	**85**	**1 197 799**	**5 398**
	mini	83	1 222 535	13 076
	no-mini	74	1 325 957	16 637

Kissat solving time on the planning track.



Shrinking vs minalluip

Solver	Track	Config.	Solved	PAR-2	Average clause size
shrinking (this paper)	Main	shrink	235	**1 897 387**	**92**
	Planning	shrink	73	1 351 542	5 373
min-alluip [F&B'20]	Main	shrink	**237**	1 904 745	104
	Planning	shrink	**81**	**1 271 930**	**3 261**

Conclusion

Summary

This work:

- Definition of trail redundancy and completeness of minimization
- More conditions to stop minimization
- Combination all-UIP and minimization, fast enough

Open questions:

- Generalization of trail redundancy to express optimality of shrinking?
- Derivation of the smallest clause without new level in NP?

Appendix Outline

Appendix

Appendix

Minimization

Appendix

Algorithm

Function $\underline{\texttt{IsLiteralRedundant}(L, d, C)}$

Input: Literal L assigned to *true*, recursion depth d, deduced clause C

Output: Whether L can be removed

```
if L is a decision then
    return false
D ∨ L ⟵ reason(L);
foreach literal K ∈ D do
    if ¬IsLiteralRedundant(¬K, d + 1, C) then
        return false
return true
```

Algorithm 0: Basic recursive minimization algorithm [Sörensson&Biere, SAT'09].

Appendix

Shrinking

Algorithm

Function ShrinkingSlice(B, C)

Input: Slice B of literals of the deduced clause C

Output: B unchanged or shrunken to UIP if successful

while $|B| > 1$ **do**

  Remove from B last assigned literal $\neg L$

  $D \vee L \longleftarrow$ reason(L)

  if $\exists\, K \in D \backslash C$ at lower level and $\neg$IsLiteralRedundant($\neg K$, 1, C) **then**

    return with failure (keep original B in C)

  else

    $B \longleftarrow B \cup \{K \in D \mid K \text{ on slice level}\}$

Replace in deduced clause C original B with the remaining UIP in B

Algorithm 0: Our new method for integrated shrinking with minimization.

Theorem

Algorithm

Function `Shrinking(`C`)`

Input: The deduced clause C (passed by reference)

Output : The shrunken and minimized clause using our new strategy

foreach Level i of literals in the deduced clause – lowest to highest **do**

$B \longleftarrow \{L \in C \mid L \text{ assigned at level } i\}$

`ShrinkingSlice(`B, C`)`

if shrinking the slice failed **then** `MinimizeSlice(`B, C`)`;

Algorithm 0: Our new method for integrated shrinking with minimization.

[F&B'20]:

1. Minimize
2. From top level to down:
   - Go up the arrows...
   - ... unless a literal from new level is added
   - ... unless heuristics trigger
3. (Minimize) strategy dependant

Our algorithm:

1. From smallest level to top:
   - Go up the arrows ...
   - ... unless a irredundant literal of lower level is added, then minimize
   - ... if successful, update minimization cache

Appendix

Implementation

Complexity

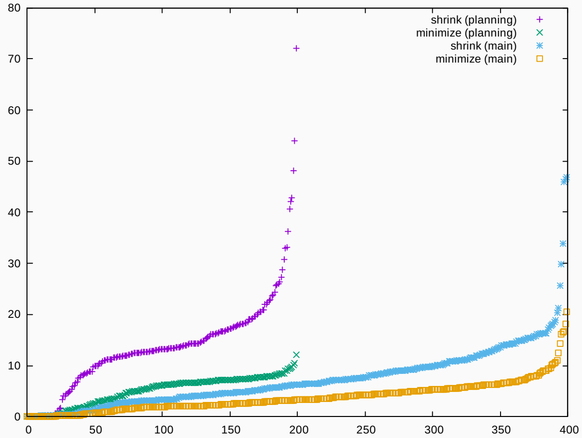
For a given level, go over all literals from the highest:

Radix Heap: $\mathcal{O}(n \log(n))$ in the size of the implication graph

Trail: size of one level (w/o chronological backtracking), size of the trail (w/ it)

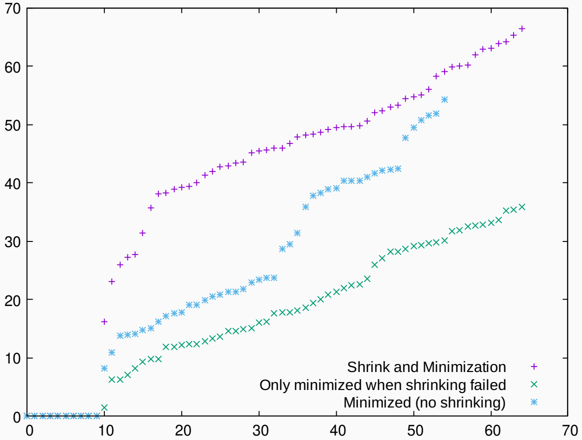
NB: sorting all literals is required anyway afterwards.

Kissat time spent in minimization and shrinking



Amount of time in percent spent during shrinking and minimization of KISSAT.

CaDiCaL number of removed literals



Percentage of removed literals in learned clauses for CaDiCaL in planning track.

Solver	Track	Config.	Solved	PAR-2	Average clause size
shrinking (this paper)	Main (400 CNFs)	shrink	235	**1 897 387**	**92**
		mini	230	1 972 949	135
	Planning (200 CNFs)	shrink	73	1 351 542	5 373
		mini	63	1 454 871	6 433
min-alluip [F&B'20]	Main	shrink	**237**	1 904 745	104
	Planning	shrink	**81**	**1 271 930**	**3 261**

Results for solvers based on CaDiCaL 1.2.1 on the SAT Competition 2020 benchmarks (128 GB RAM)