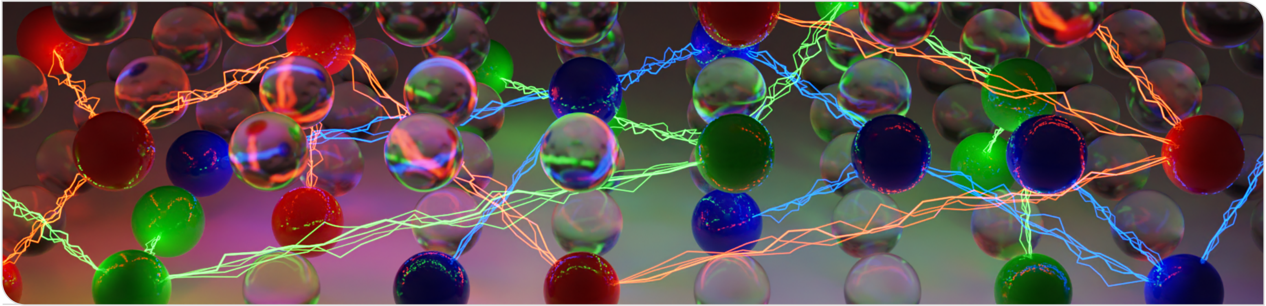
Practical SAT Solving

Lecture 6

Markus Iser, Dominik Schreiber, Tomáš Balyo | May 27, 2024



Overview

Karlsruher Institut für Technologie

Recap.

Lecture 4: Classic Heuristics and Modern SAT Solving 1:

- Decision Heuristics, Restart Strategies, Phase Saving
- Modern SAT Solving 1: Conflict Analysis / Clause Learning

Lecture 5: Parallel SAT Solving 1:

- To continued on June 10, 2024: Parallel SAT Solving 2

Today's Topic: Modern SAT Solving 2

- Efficient Unit Propagation
- Clause Forgetting
- Modern Decision Heuristics
- Preprocessing

Conflict-driven Clause Learning (CDCL) Algorithm

Last Time

- Classic Decision Heuristics
- Restart Strategies
- Clause Learning
- Non-Chronological Backtracking

Today

- Efficient Unit Propagation
- Clause Forgetting
- Modern Decision Heuristics
- Preprocessing

Algorithm 1: CDCL(CNF Formula F, &Assignment A $\leftarrow \emptyset$)

```
if not PREPROCESSING then return UNSAT
while A is not complete do
    UNIT PROPAGATION
    if A falsifies a clause in F then
        if decision level is 0 then return UNSAT
        else
            (clause, level) ← CONFLICT-ANALYSIS
            add clause to F and backtrack to level
            continue
    if RESTART then backtrack to level 0
    if CLEANUP then forget some learned clauses
    BRANCHING
return SAT
```

Unit Propagation

Hot Paths in CDCL Solvers

heat	$\varnothing$ *per sec.*[a]	
Clause Access		Unpredictable memory access: most expensive
Iterate Occurrences		Predictable memory access: array of pointers (hardware prefetching)
Propagation	$\sim \mathbf{10^6}$	**Access occurrence-list of yet unpropagated literal**
Decision	$\sim 10^3$	
Conflict	$\sim 10^3$	*Learn a clause → more to check for propagation*
Restart	$\sim 10^{-1}$	
Cleanup		*Forget some learned clauses → less to check for propagation*

[a] Order of magnitude of average event count per second (in runs of Cadical on a large combined benchmark set)

Unit Propagation

Example: Unit Propagation with Full Occurrence Lists

Trail

level	value	reason
1	a	$\perp$

Occurrence Lists

idx.	occurrences	
a	$*1$	
$\neg a$	$*2$	$*3$
b	$*1$	$*2$
$\neg b$	$*3$	
c	$*3$	$*1$
$\neg c$	$*2$	

Formula

addr.	clause		
$*1$	a	b	c
$*2$	$\neg a$	b	$\neg c$
$*3$	$\neg a$	$\neg b$	c

Unit Propagation

Example: Unit Propagation with Full Occurrence Lists

Trail

level	value	reason
1	a	$\bot$

Occurrence Lists

idx.	occurrences	
a	^{*1}	
$\neg a$	^{*2}	^{*3}
b	^{*1}	^{*2}
$\neg b$	^{*3}	
c	^{*3}	^{*1}
$\neg c$	^{*2}	

Formula

addr.	clause		
^{*1}	a	b	c
^{*2}	$\neg a$	b	$\neg c$
^{*3}	$\neg a$	$\neg b$	c

Unit Propagation

Example: Unit Propagation with Full Occurrence Lists

Trail

level	value	reason
1	a	$\perp$
2	c	$\perp$

Occurrence Lists

idx.	occurrences	
a	$*1$	
$\neg a$	$*2$	$*3$
b	$*1$	$*2$
$\neg b$	$*3$	
c	$*3$	$*1$
$\neg c$	$*2$	

Formula

addr.	clause		
$*1$	a	b	c
$*2$	$\neg a$	b	$\neg c$
$*3$	$\neg a$	$\neg b$	c

Unit Propagation

Example: Unit Propagation with Full Occurrence Lists

Trail

level	value	reason
1	**a**	$\bot$
2	**c**	$\bot$
2	**b**	$*2$

Occurrence Lists

idx.	occurrences	
a	$*1$	
$\neg a$	$*2$	$*3$
b	$*1$	$*2$
$\neg b$	$*3$	
c	$*3$	$*1$
$\neg c$	$*2$	

Formula

addr.	clause		
$*1$	a	b	c
$*2$	$\neg a$	b	$\neg c$
$*3$	$\neg a$	$\neg b$	c

Unit Propagation: Two Watched Literals

Motivation: Hot Path

heat	∅ *per sec.*[a]
Clause Access	
Iterate Occurrences	
Propagation	$\sim$ **10^6**

Idea: Reduced occurrence tracking by only keeping the following **invariant:**

Each yet unsatisfied clause is watched by, i.e., in the occurrence list of, two of its unassigned literals.

Reasoning: less literals watched $\rightarrow$ shorter occurrence lists $\rightarrow$ less clause accesses $\rightarrow$ fast unit propagation

[a] Order of magnitude of average event count per second (in runs of Cadical on a large combined benchmark set)

Unit Propagation: Two Watched Literals

Karlsruher Institut für Technologie

Motivation: Hot Path

heat	∅ *per sec.*[a]
Clause Access	
Iterate Occurrences	
Propagation	$\sim \mathbf{10^6}$

Idea: Reduced occurrence tracking by only keeping the following **invariant:**

Each yet unsatisfied clause is watched by, i.e., in the occurrence list of, two of its unassigned literals.

Reasoning: less literals watched $\rightarrow$ shorter occurrence lists $\rightarrow$ less clause accesses $\rightarrow$ fast unit propagation

- Why do two watched literals per clause suffice?

[a]Order of magnitude of average event count per second (in runs of Cadical on a large combined benchmark set)

Unit Propagation: Two Watched Literals

Motivation: Hot Path

heat	∅ *per sec.*[a]
Clause Access	
Iterate Occurrences	
Propagation	$\sim$ **10^6**

Idea: Reduced occurrence tracking by only keeping the following **invariant:**

Each yet unsatisfied clause is watched by, i.e., in the occurrence list of, two of its unassigned literals.

Reasoning: less literals watched $\rightarrow$ shorter occurrence lists $\rightarrow$ less clause accesses $\rightarrow$ fast unit propagation

- Why do two watched literals per clause suffice?
- Why does one watched literal per clause not suffice?

[a] Order of magnitude of average event count per second (in runs of Cadical on a large combined benchmark set)

Unit Propagation: Two Watched Literals

Motivation: Hot Path

heat	∅ *per sec.*[a]
Clause Access	
Iterate Occurrences	
Propagation	$\sim$ **10^6**

Idea: Reduced occurrence tracking by only keeping the following **invariant:**

Each yet unsatisfied clause is watched by, i.e., in the occurrence list of, two of its unassigned literals.

Reasoning: less literals watched $\rightarrow$ shorter occurrence lists $\rightarrow$ less clause accesses $\rightarrow$ fast unit propagation

- Why do two watched literals per clause suffice?
- Why does one watched literal per clause not suffice?
- How do we keep that invariant? (Branching?, Backtracking?)

[a] Order of magnitude of average event count per second (in runs of Cadical on a large combined benchmark set)

Unit Propagation

Karlsruher Institut für Technologie

Example: Unit Propagation with Two Watched Literals

Trail

level	value	reason

Two Watched Literals

idx.	occurrences	
a	*1	
$\neg a$	*2	*3
b	*1	*2
$\neg b$	*3	
c		
$\neg c$		

Formula

addr.	clause		
*1	a	b	c
*2	$\neg a$	b	$\neg c$
*3	$\neg a$	$\neg b$	c

Unit Propagation

Karlsruher Institut für Technologie

Example: Unit Propagation with Two Watched Literals

Trail

level	value	reason
1	a	$\bot$

Two Watched Literals

idx.	occurrences	
a	^{*1}	
$\neg a$	^{*2}	^{*3}
b	^{*1}	^{*2}
$\neg b$	^{*3}	
c		
$\neg c$		

Formula

addr.	clause		
^{*1}	a	b	c
^{*2}	$\neg a$	b	$\neg c$
^{*3}	$\neg a$	$\neg b$	c

Unit Propagation

Karlsruher Institut für Technologie

Example: Unit Propagation with Two Watched Literals

Trail

level	**value**	**reason**
1	**a**	$\perp$

Two Watched Literals

idx.	**occurrences**	
a	*1	
$\neg a$		*3
b	*1	*2
$\neg b$	*3	
c		
$\neg c$	*2	

Formula

addr.	**clause**		
*1	a	b	c
*2	$\neg c$	b	$\neg a$
*3	$\neg a$	$\neg b$	c

Unit Propagation

Example: Unit Propagation with Two Watched Literals

Trail

level	value	reason
1	**a**	$\perp$

Two Watched Literals

idx.	occurrences	
a	^{*1}	
$\neg a$		
b	^{*1}	^{*2}
$\neg b$	^{*3}	
c	^{*3}	
$\neg c$	^{*2}	

Formula

addr.	clause		
^{*1}	a	b	c
^{*2}	$\neg c$	b	$\neg a$
^{*3}	$\boldsymbol{c}$	$\neg b$	$\boldsymbol{\neg a}$

Unit Propagation

Example: Unit Propagation with Two Watched Literals

Trail

level	value	reason
1	a	⊥
2	c	⊥

Two Watched Literals

idx.	occurrences	
a	$*1$	
$\neg a$		
b	$*1$	$*2$
$\neg b$	$*3$	
c	$*3$	
$\neg c$	$*2$	

Formula

addr.	clause		
$*1$	a	b	c
$*2$	$\neg c$	b	$\neg a$
$*3$	c	$\neg b$	$\neg a$

Unit Propagation

Example: Unit Propagation with Two Watched Literals

Trail

level	value	reason
1	a	⊥
2	c	⊥
2	b	*2

Two Watched Literals

idx.	occurrences	
a	*1	
$\neg a$		
b	*1	*2
$\neg b$	*3	
c	*3	
$\neg c$	*2	

Formula

addr.	clause		
*1	a	b	c
*2	$\neg c$	b	$\neg a$
*3	c	$\neg b$	$\neg a$

Unit Propagation: Two Watched Literals

Karlsruher Institut für Technologie

Two Watched Literals: Optimizations

heat	∅ per sec.[a]
Clause Access	
Iterate Occurrences	
Propagation	$\sim$ **10^6**

Invariant: Each yet unsatisfied clause is watched by two of its unassigned literals.

$\rightarrow$ **Reduced Load in Occurrence Tracking**

Optimization 1: Keep watched literals the first two in clause
$\rightarrow$ Alternative: Store watched literals in other location

Note: What happens if clauses are kept in shared memory for parallel solving?

Optimization 2: Also keep a literal of each clause directly in occurrence list
$\rightarrow$ Skip clause access if that literal is satisfied

[a] Order of magnitude of average event count per second (in runs of Cadical on a large combined benchmark set)

Recap

Karlsruher Institut für Technologie

Unit Propagation

- Hottest path in CDCL solvers
- Two watched literals per clause suffice for unit propagation (and conflict detection)
- Other optimizations: keep watched literals first in clause, keep a literal of each clause directly in occurrence list

Next Up

Clause Forgetting

Clause Forgetting

Motivation

Clause learning is most important pruning strategy in CDCL solvers.[a]

Problem:

- Slows down unit propagation
- Risk of running out of memory

Solution:

- Periodically forget some learned clauses
- Keep only “the best” learned clauses

[a] “Empirical Study of the Anatomy of Modern Sat Solvers”, Katebi et al., 2013

Clause Forgetting

Motivation

Clause learning is most important pruning strategy in CDCL solvers.[a]

Problem:

- Slows down unit propagation
- Risk of running out of memory

Solution:

- Periodically forget some learned clauses
- Keep only “the best” learned clauses
- **How to figure out which learned clauses are “the best”?**

[a] “Empirical Study of the Anatomy of Modern Sat Solvers”, Katebi et al., 2013

Clause Forgetting

Karlsruher Institut für Technologie

Periodic Clause Forgetting: Heuristics

- **Clause Size**

  Keep short clauses

- **Least Recently Used (LRU)**

  Keep clauses which where reasons in recent conflicts: clause activity (moving average)

- **Literal Block Distance (LBD)**

  Keep clauses with a low number of decision levels[a]

[a] Predicting Learnt Clauses Quality in Modern SAT Solvers, Audemard & Simon (IJCAI 2009)

Forgetting Heuristic: Literal Block Distance (LBD)

"Impact of Community Structure on SAT Solver Performance", Newsham et al., SAT 2014

Take home: LBD correlates with number of touched communities

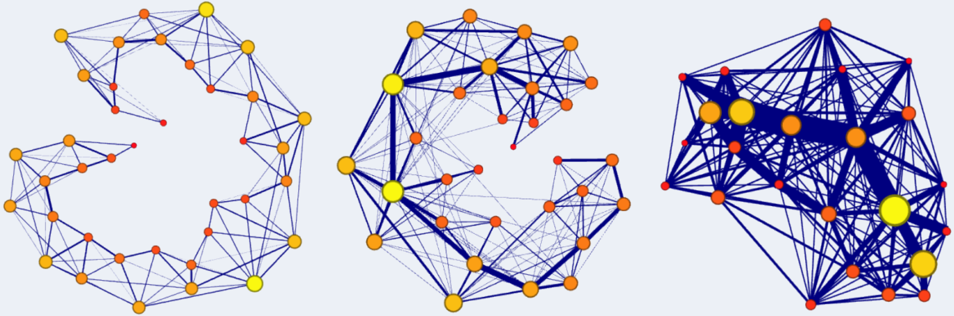


Image Source: "Community Structure in Industrial SAT Instances", Ansotegui et al., AIJ 2019

Clause Forgetting: Modern Hybrid Approach

Manage clauses differently in three tiers

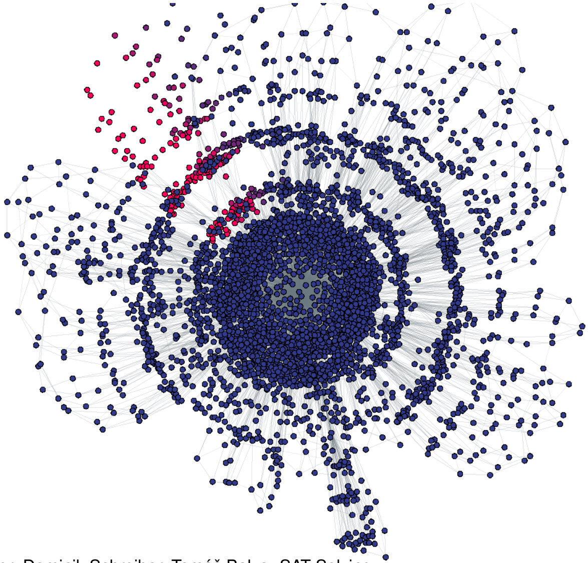
Tier	Strategy	Description
`core`	LBD	Permanently store clauses of LBD $\leq k$ (core-cut value, 3 in practice)
`mid-tier`	LRU	Clauses stay here if used in recent conflicts
`local`	LRU	Keep fixed number of clauses (say 5000) of highest activity

History

- `core` and `local` tier introduced in SWDiA5BY (Chanseok Oh, 2014)
- `mid-tier` introduced in CoMinisatPS (Chanseok Oh, 2015)
- "Between SAT and UNSAT: The Fundamental Difference in CDCL SAT" (Chanseok Oh, 2015)
- Note: MapleCOMSPS (2016) is a CoMinisatPS fork

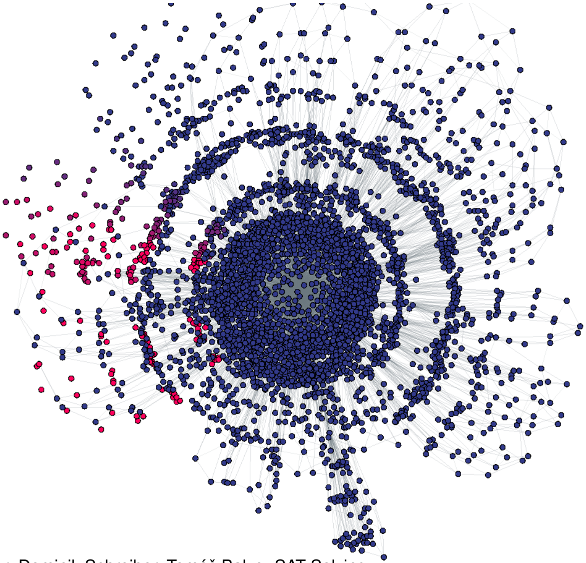
Visualized Instance: Aprove (Termination Analysis, SAT)

initial layout, recently active variables after 1000 conflicts



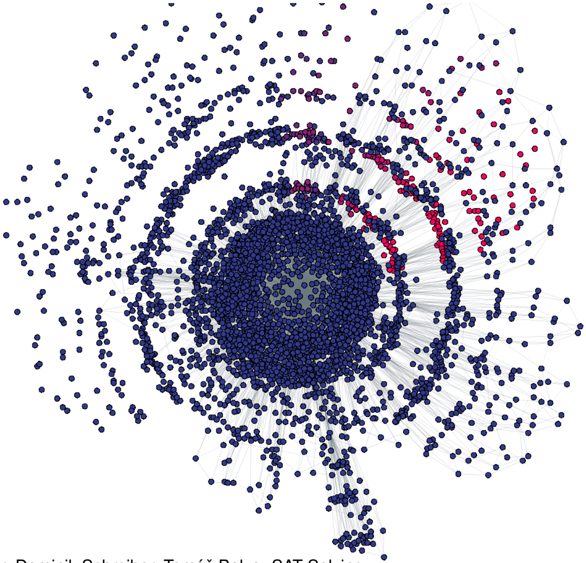
Visualized Instance: Aprove (Termination Analysis, SAT)

initial layout, recently active variables after 1690 conflicts



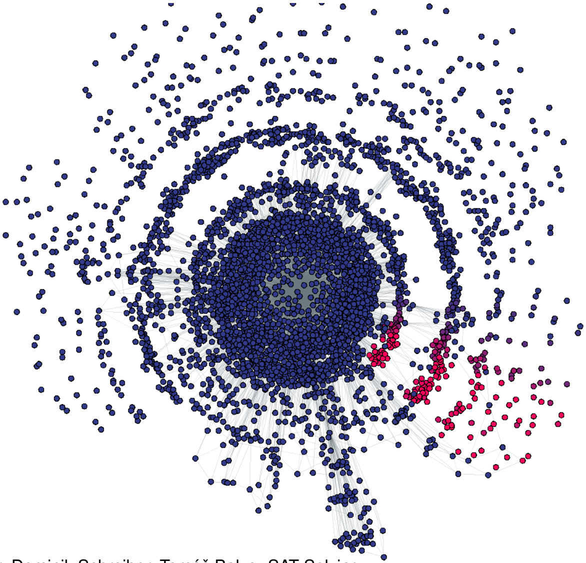
Visualized Instance: Aprove (Termination Analysis, SAT)

initial layout, recently active variables after 3090 conflicts



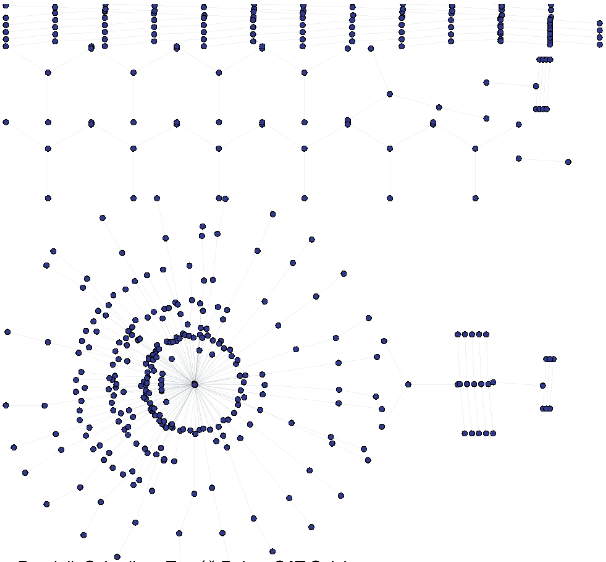
Visualized Instance: Aprove (Termination Analysis, SAT)

initial layout, recently active variables after 5000 conflicts



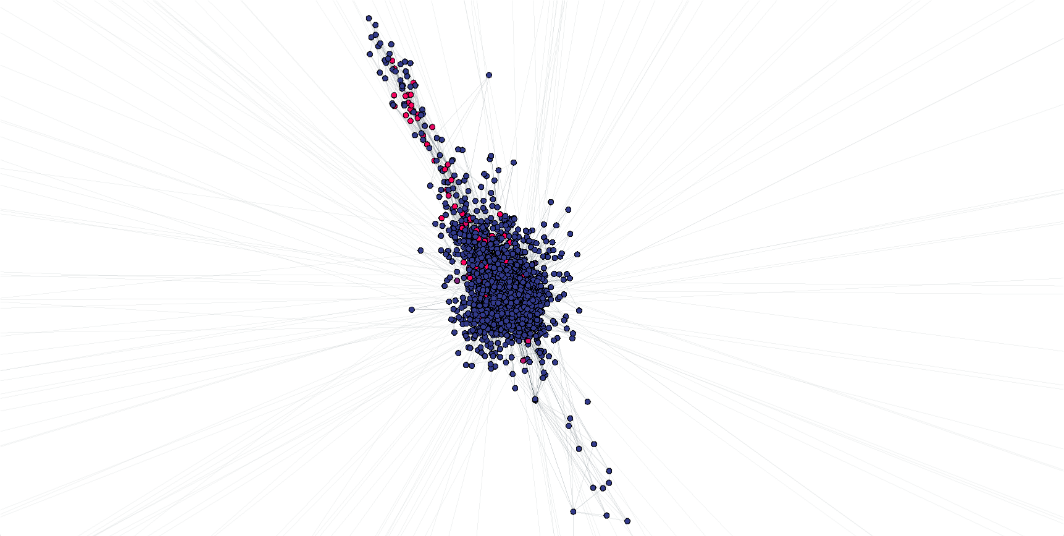
Visualized Instance: Aprove (Termination Analysis, SAT)

relayout after 6000 conflicts



Visualized Instance: Aprove (Termination Analysis, SAT)

core after 52500 conflicts

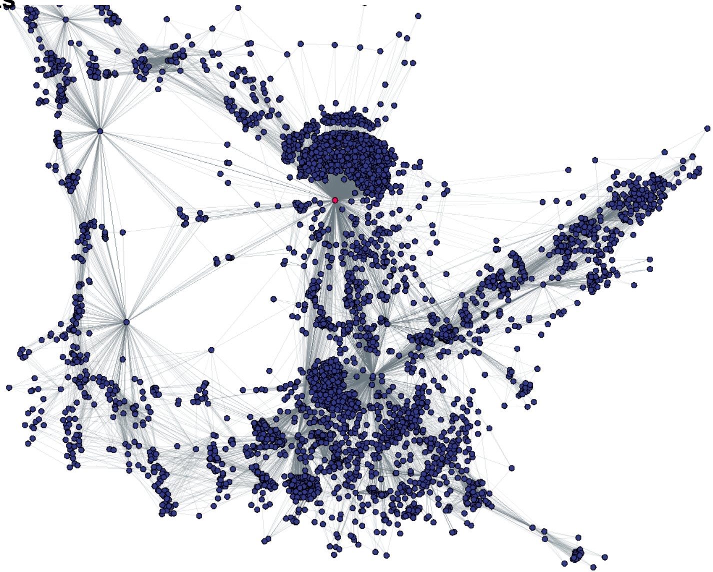


Visualized Instance: Newton SMT (SV Competition, SAT)

initial layout

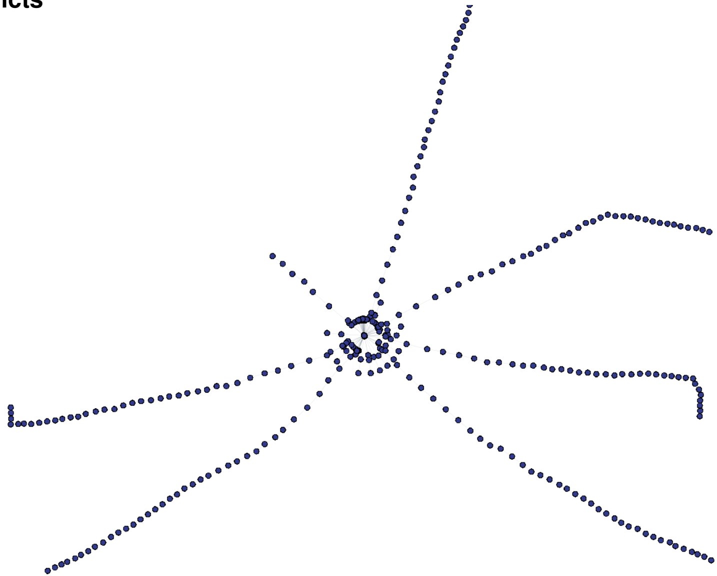
Visualized Instance: Newton SMT (SV Competition, SAT)

after 10000 conflicts



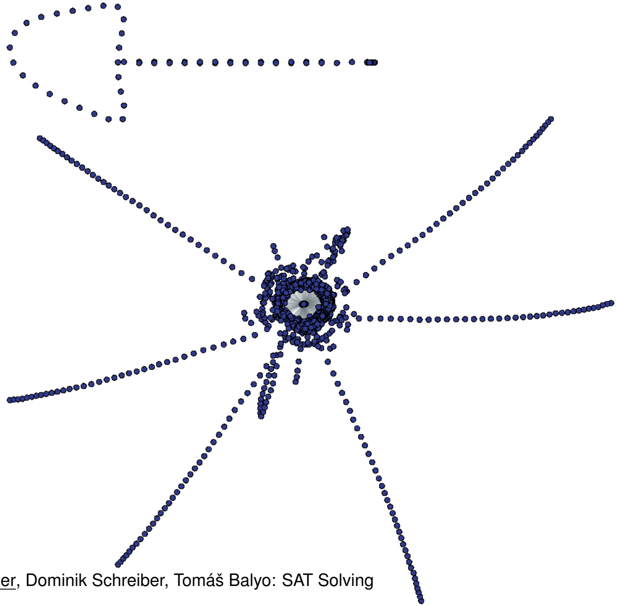
Visualized Instance: Newton SMT (SV Competition, SAT)

after 1000000 conflicts



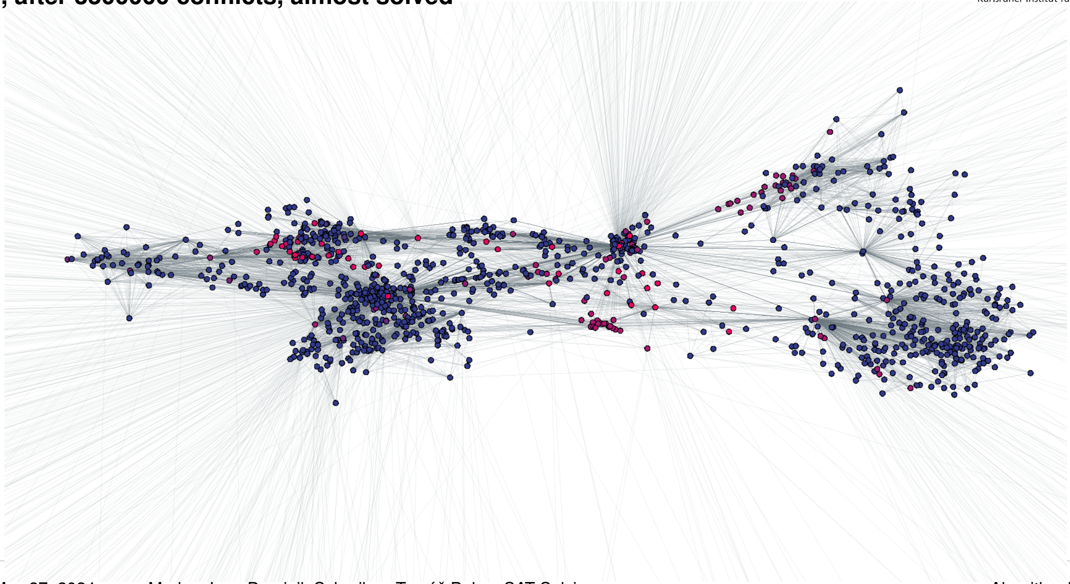
Visualized Instance: Newton SMT (SV Competition, SAT)

after 3000000 conflicts



Visualized Instance: Newton SMT (SV Competition, SAT)

core, after 3500000 conflicts, almost solved



Recap

So far

- Efficient Unit Propagation
- Clause Forgetting Heuristics:
  - Size, LRU, LBD
  - Three-Tier Clause Management

Next Up

Modern Decision Heuristics

Variable State Independent Decaying Sum (VSIDS)

VSIDS Heuristic

Implemented in most CDCL solvers. First presented in SAT solver Chaff.[a]

Always select variable with highest score for branching. Scores are updated after each conflict.

- Initialize variable score (with zero or use some static heuristic)
- New conflict clause c: score is incremented for all variables in c
- Periodically, divide all scores by a constant

[a] Chaff: Engineering an efficient SAT solver (Moskewicz et al., 2001)

Variable State Independent Decaying Sum (VSIDS)

Example: Score Update after Conflict

Formula:

$\{x_1, x_4\}, \{x_1, \overline{x_3}, \overline{x_8}\}, \{x_1, x_8, x_{12}\}, \{x_2, x_{11}\},$

$\{\overline{x_7}, \overline{x_3}, x_9\}, \{\overline{x_7}, x_8, \overline{x_9}\}, \{x_7, x_8, \overline{x_{10}}\}$

$\{x_7, x_{10}, \overline{x_{12}}\}$ (new learned clause)

Scores before:

4 : x_8

3 : x_1, x_7

2 : x_3

1 : $x_2, x_4, x_9, x_{10}, x_{11}, x_{12}$

Scores after:

4 : x_8, x_7

3 : x_1

2 : x_3, x_{10}, x_{12}

1 : x_2, x_4, x_9, x_{11}

Variable State Independent Decaying Sum (VSIDS)

Example: Score Update after Conflict

Formula:

$\{x_1, x_4\}, \{x_1, \overline{x_3}, \overline{x_8}\}, \{x_1, x_8, x_{12}\}, \{x_2, x_{11}\},$
$\{\overline{x_7}, \overline{x_3}, x_9\}, \{\overline{x_7}, x_8, \overline{x_9}\}, \{x_7, x_8, \overline{x_{10}}\}$
$\{x_7, x_{10}, \overline{x_{12}}\}$ (new learned clause)

Scores before:

$4 : x_8$
$3 : x_1, x_7$
$2 : x_3$
$1 : x_2, x_4, x_9, x_{10}, x_{11}, x_{12}$

Scores after:

$4 : x_8, x_7$
$3 : x_1$
$2 : x_3, x_{10}, x_{12}$
$1 : x_2, x_4, x_9, x_{11}$

- VSIDS leads to more “focused” search
- prefers variables that occurred in recent conflicts
- tends to find smaller unsatisfiable subsets

Variable State Independent Decaying Sum (VSIDS)

Keep list of variables sorted by scores

Common implementation: Binary Heap

Heap Operation	Complexity	Callee
`insert_with_priority`	$\mathcal{O}(\log n)$	Backtracking
`pull_highest_priority_element`	$\mathcal{O}(\log n)$	Branching
`increase_key / bump_variable`	$\mathcal{O}(\log n)$	Conflict Analysis
`decay`	$\mathcal{O}(n)$	[Periodic][a]

[a] Periodically divide scores to give priority to recently learned clauses

Variable State Independent Decaying Sum (VSIDS)

VSIDS Variants

Chaff (2001)

- decay: half scores every 256 conflicts
- sort priority queue after each decay only

Berkmin (2002)

- bump all literals in implication graph
- divide scores by 4

Minisat (2003)

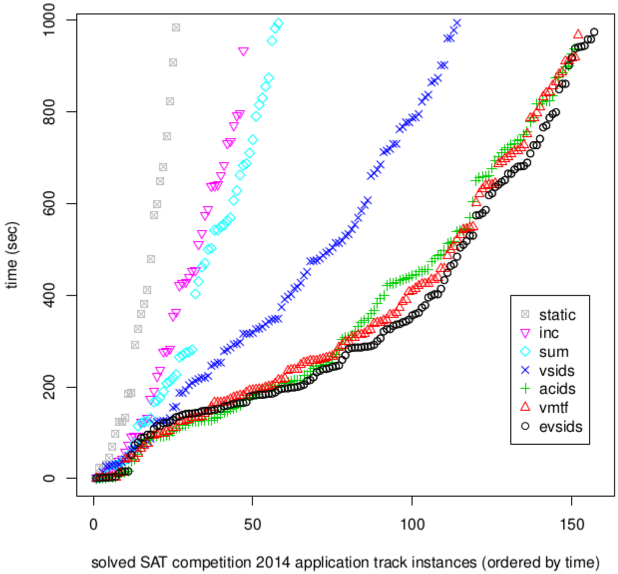
- Exponential VSIDS (EVSIDS)
- Reason-side Bumping

Alternatives

Siege (2004): Variable Move To Front (VMTF)   **HaifaSAT (2008)**: Clause Move To Front (CMTF)

"Evaluating CDCL Variable Scoring Schemes"

Biere & Fröhlich, 2015



Recent Hybrid Approaches

Hybrid Approaches

- **Warmup Phase**:
  - MapleCOMSPS (2016): use Learning Rate-based Branching (LRB) in *initial* period, then switch to VSIDS
  - Maple_LCM_Dist (2017): use Distance Heuristic (Dist.) in *initial* period, then switch to VSIDS
- **Reinforcement Learning**: Kissat_MAB (2021)
  - Two-armed Bandid switches between VSIDS and Conflict History-Based (CHB) Heuristic
  - Reward function favors variables that contribute to learning “good” clauses

Recap

Karlsruher Institut für Technologie

So far

- Unit Propagation
- Clause Forgetting
- Modern Branching Heuristics
  - Mostly VSIDS
  - Hybrid approaches: warmup VSIDS scores, reinforcement learning