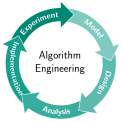
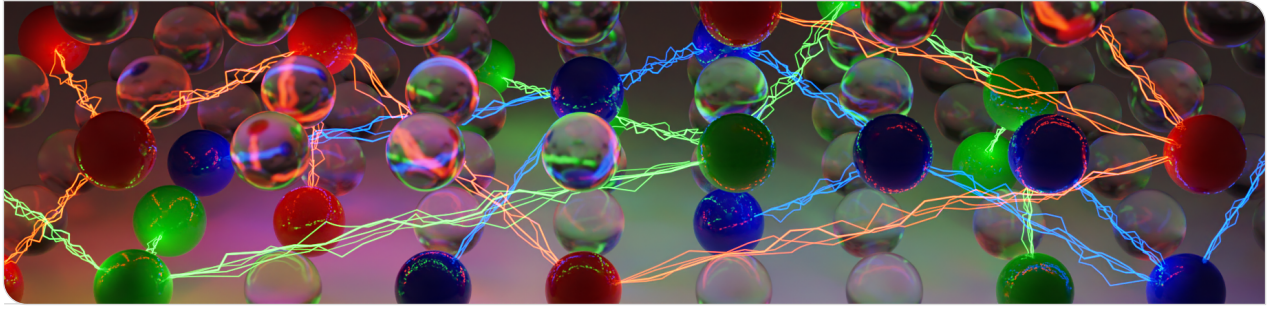
Karlsruher Institut für Technologie



Practical SAT Solving

Lecture 4

Markus Iser, Dominik Schreiber, Tomáš Balyo | May 06, 2024



Results of Feedback Round 1

Result 1: Grades

- Content/Relevance: $2 \times 5p, 4 \times 4p$
- Quality: $1 \times 5p, 4 \times 4p, 1 \times 3p$

Result 2: Aspects

- Positive: practical, discussions, examples, application oriented, exercises, small group size, C++
- Negative: C++, “winner takes all” in competitions, explanation of sequential counter encoding, technical terms not introduced

Lessons learned

- Better care of technical terms introduction
- Split distribution of bonus points in competitions
- Some Python examples in exercises

Overview

Recap. Lecture 4: Heuristics and Modern SAT Solving 1

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

Regarding Lecture 5: Parallel SAT Solving 1

This topic will be continued on June 10, 2024: Parallel SAT Solving 2.

Today's Topic: Modern SAT Solving 2

- Variable State Independent Decaying Sum (VSIDS) Heuristic
- Efficient Unit Propagation, Watched Literals
- Clause Forgetting
- Preprocessing

Conflict-driven Clause Learning (CDCL) Algorithm

Karlsruher Institut für Technologie

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 $\leftarrow \emptyset$)

```
1  if not PREPROCESSING then return UNSAT
2  while A is not complete do
3      UNIT PROPAGATION
4      if A falsifies a clause in F then
5          if decision level is 0 then return UNSAT
6          else
7              (clause, level) ← CONFLICT-ANALYSIS
8              add clause to F and backtrack to level
9              continue
10     if RESTART then backtrack to level 0
11     if CLEANUP then forget some learned clauses
12     BRANCHING
13 return SAT
```

Unit Propagation: Cost

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	$\sim$ **10^6**	**Access occurrence-list of yet unpropagated literal**
Decision	$\sim 10^3$	
Conflict	$\sim 10^3$	*Learn a clause* $\rightarrow$ *more to check for propagation*
Restart	$\sim 10^{-1}$	
Cleanup		*Forget some learned clauses* $\rightarrow$ *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

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**	$\bot$
2	**c**	$\bot$

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**	$\bot$
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

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?

[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

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

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

Example: Unit Propagation with Two Watched Literals

Trail

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

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
$*2$	$\neg \boldsymbol{c}$ b $\neg \boldsymbol{a}$
$*3$	$\neg a$ $\neg b$ c

Unit Propagation

Example: Unit Propagation with Two Watched Literals

Trail

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

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	$\neg a$
$*3$	$\mathbf{c}$	$\neg b$	$\neg \mathbf{a}$

Unit Propagation

Example: Unit Propagation with Two Watched Literals

Trail

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

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	$\bot$
2	c	$\bot$
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

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

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.

Problems:

- Without restrictions the number of clauses grows exponentially (Remember: naive Saturation Algorithm)
- Risk of running out of memory
- Slows down unit propagation

Idea:

- Periodically forget some learned clauses
- Keep only important learned clauses
- Use heuristics to estimate clause importance

Clause Forgetting

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

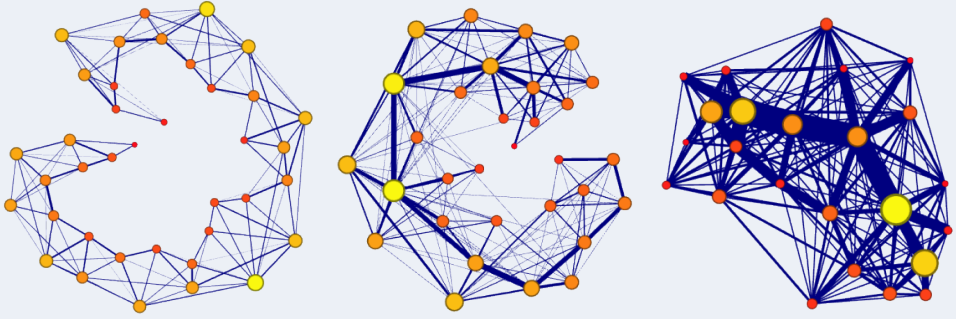


Image Source: "Community Structure in Industrial SAT Instances", Ansotegui et al., AIJ 2019

Clause Forgetting: Modern Hybrid Approach

Three-Tier Clause Management

Manage clauses differently in three tiers:

- `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

Recap

So far

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

Next Up

Modern Branching Heuristics

Variable State-Independent Decaying Sum (VSIDS)

Implemented in most CDCL solvers. First presented in SAT solver Chaff.[1]

VSIDS Heuristic

Compute score for each variable, select variable with highest score:

- 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

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

VSIDS Heuristic

VSIDS Example

Initial F:

$\{x_1, x_4\}$
$\{x_1, \overline{x_3}, \overline{x_8}\}$
$\{x_1, x_8, x_{12}\}$
$\{x_2, x_{11}\}$
$\{\overline{x_7}, \overline{x_3}, x_9\}$
$\{\overline{x_7}, x_8, \overline{x_9}\}$
$\{x_7, x_8, \overline{x_{10}}\}$

Scores:

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

VSIDS Heuristic

VSIDS Example

Initial F:

$\{x_1, x_4\}$
$\{x_1, \overline{x_3}, \overline{x_8}\}$
$\{x_1, x_8, x_{12}\}$
$\{x_2, x_{11}\}$
$\{\overline{x_7}, \overline{x_3}, x_9\}$
$\{\overline{x_7}, x_8, \overline{x_9}\}$
$\{x_7, x_8, \overline{x_{10}}\}$

Scores:

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

F with new learned clause added:

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

Scores:

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 Heuristic

Keep list of variables sorted by score

VSIDS Common Implementation: Binary Heap

- Backtrack: `insert_with_priority` $\in \mathcal{O}(\log n)$
- Branch: `pull_highest_priority_element` $\in \mathcal{O}(\log n)$
- Bump: `increase_key` $\in \mathcal{O}(\log n)$
- Decay: `decay` $\in \mathcal{O}(n)$

Reasoning behind VSIDS

Make heuristics more “focused”

- try to find small unsatisfiable subsets
- prefer variables that occurred in a recent conflict

VSIDS Heuristic

Periodically divide scores to give priority to recently learned clauses

VSIDS Variants

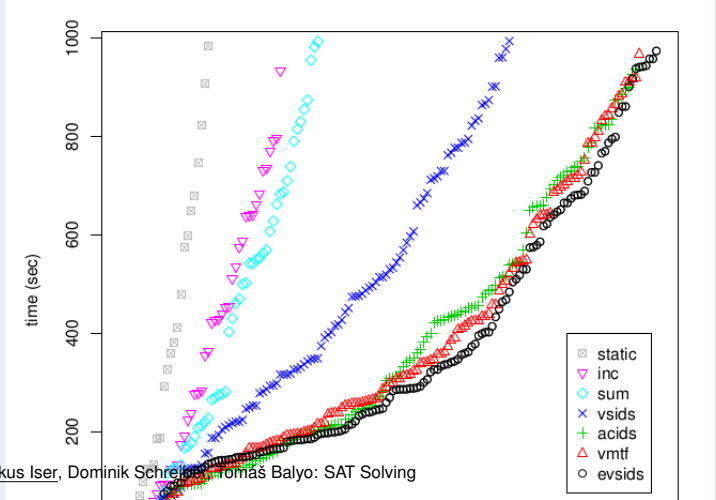
- Chaff (2001)
  - half scores every 256 conflicts (“decay”)
  - sort priority queue after each decay only
- Berkmin (2002)
  - bump all literals in implication graph
  - divide scores by 4
- Minisat (2003)
  - exponential VSIDS (eVSIDS)

Alternatives

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

Comparison of Heuristics

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

So far

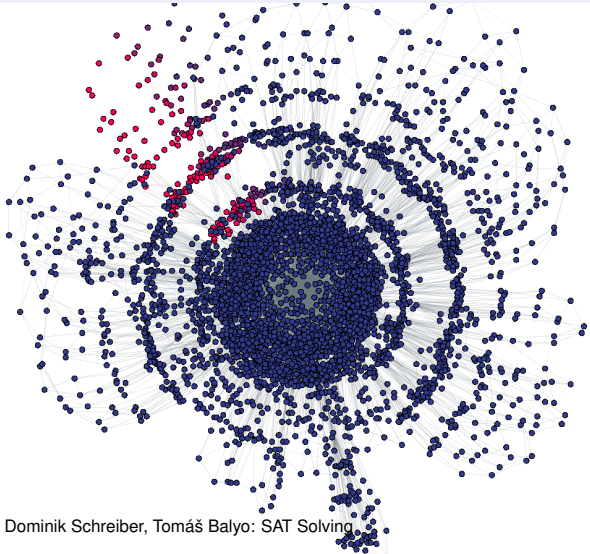
- Unit Propagation
- Clause Forgetting
- Modern Branching Heuristics
  - Mostly VSIDS
  - Hybrid approaches: warmup VSIDS scores, reinforcement learning

Next Up

Evolution of formula structure with clause learning

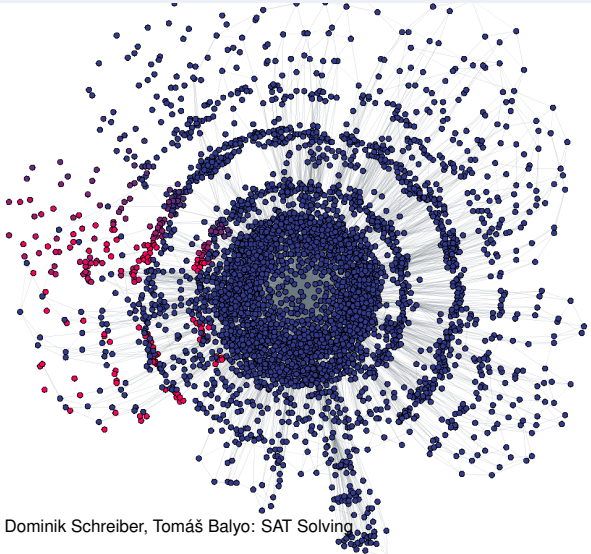
Instance: Aprove (Termination Analysis)

1000 Conflicts



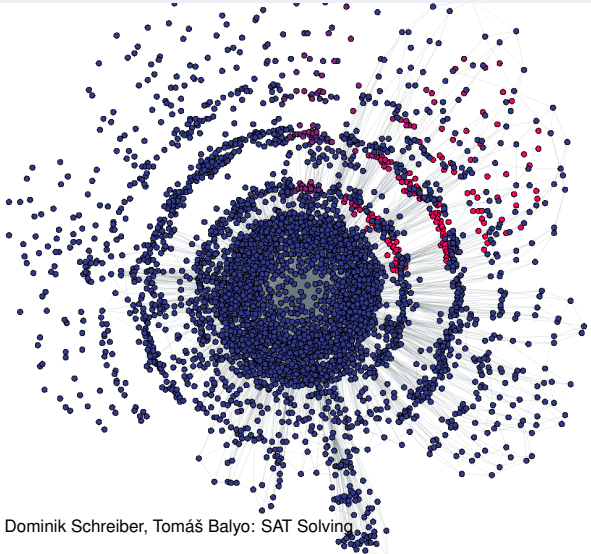
Instance: Aprove (Termination Analysis)

1690 Conflicts



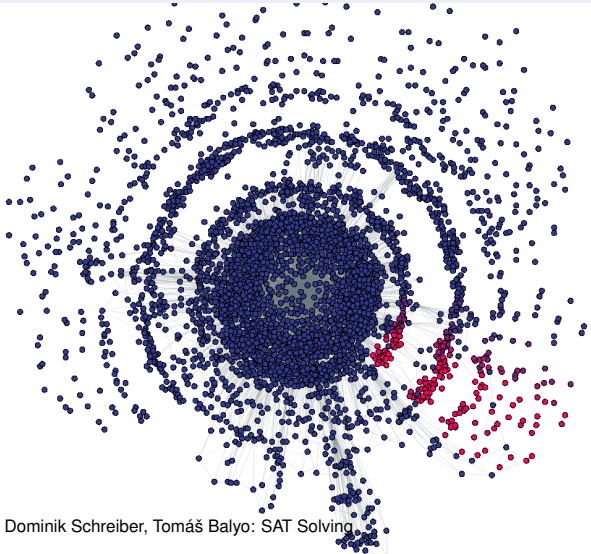
Instance: Aprove (Termination Analysis)

3090 Conflicts



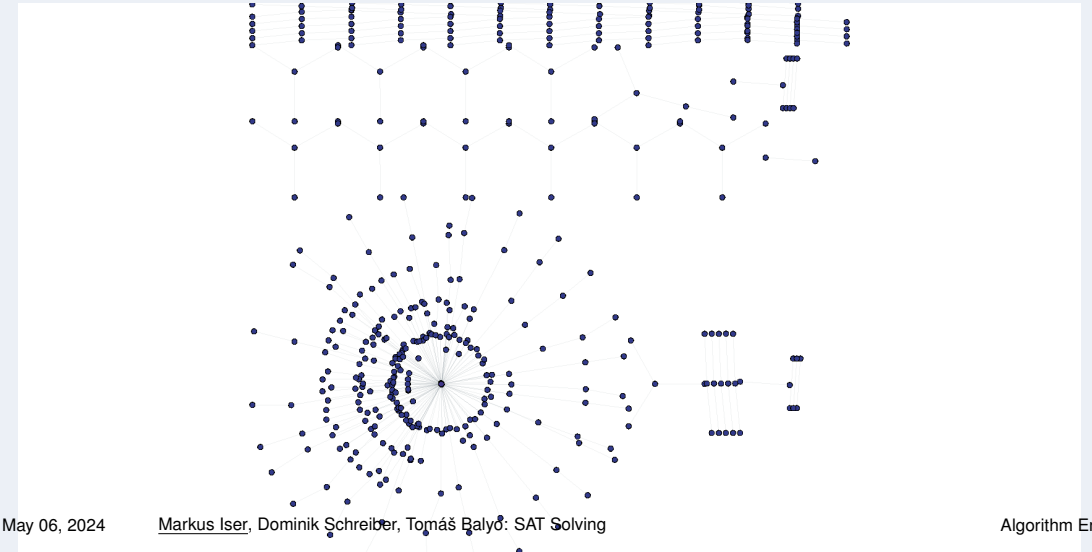
Instance: Aprove (Termination Analysis)

5000 Conflicts



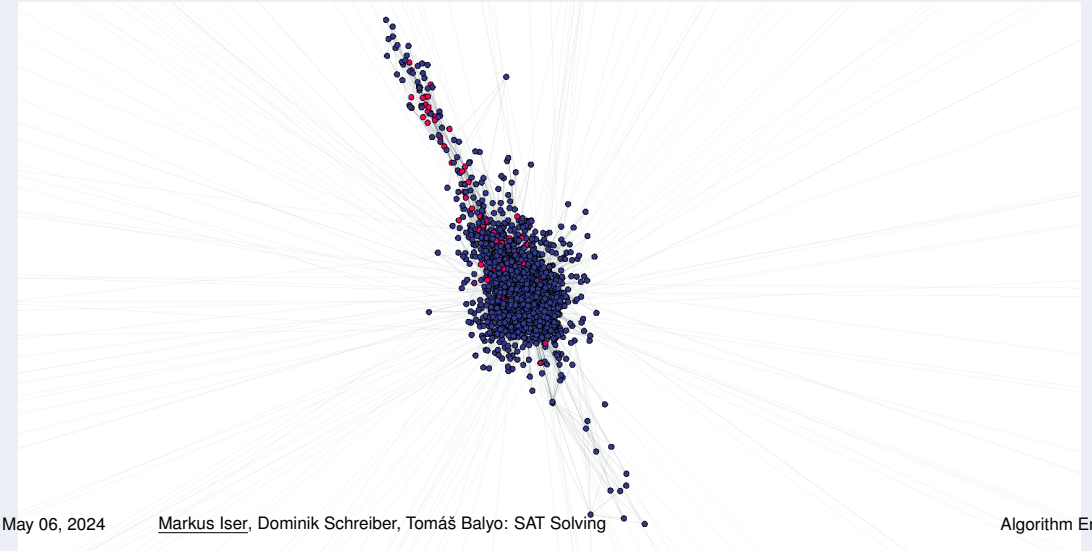
Instance: Aprove (Termination Analysis)

6000 Conflicts (Relayout)

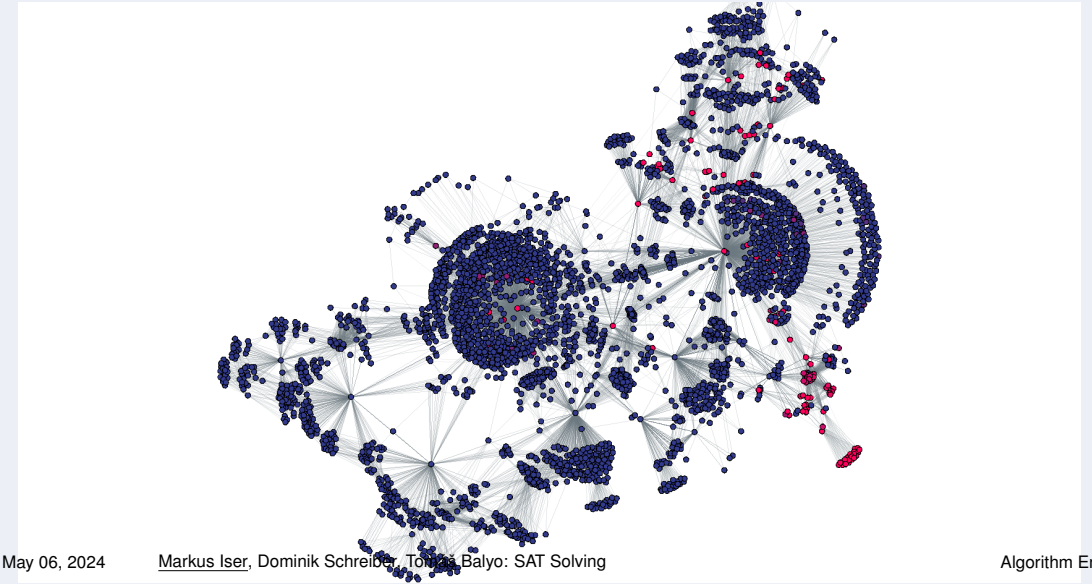


Instance: Aprove (Termination Analysis)

52500 Conflicts (Core)

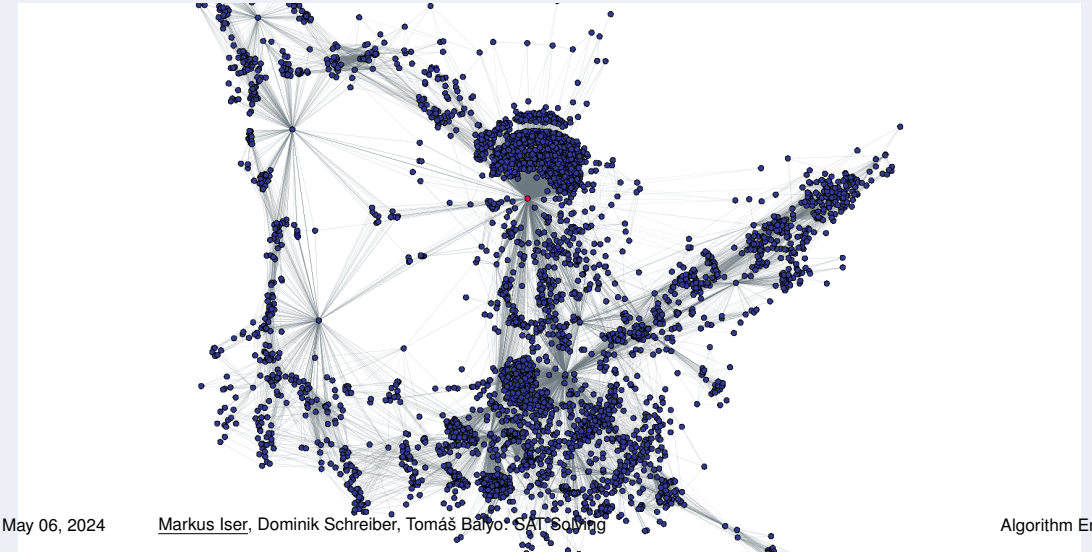


Instance: Newton SMT (SV Competition)



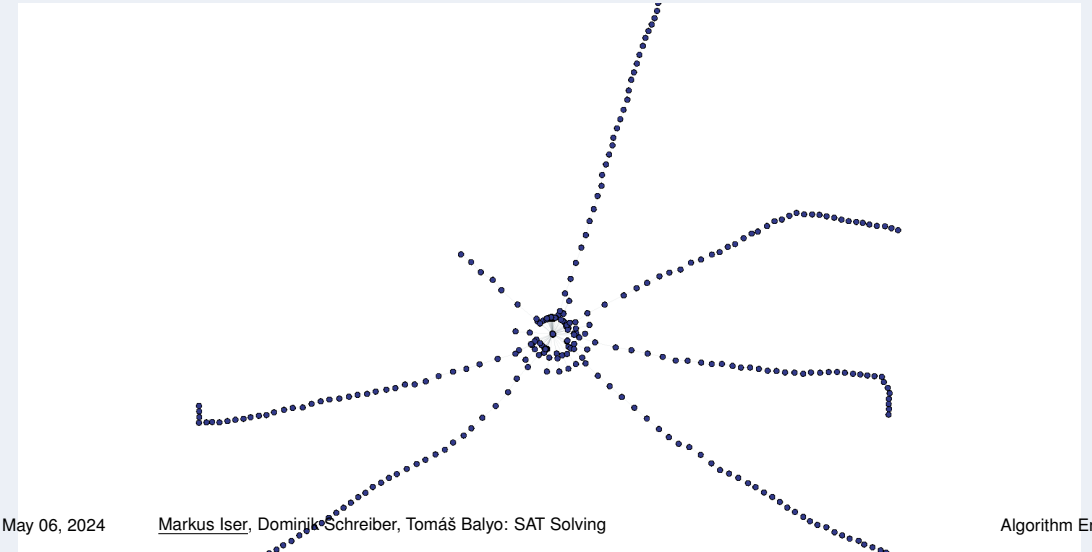
Instance: Newton SMT (SV Competition)

10000 Conflicts (Relayout)



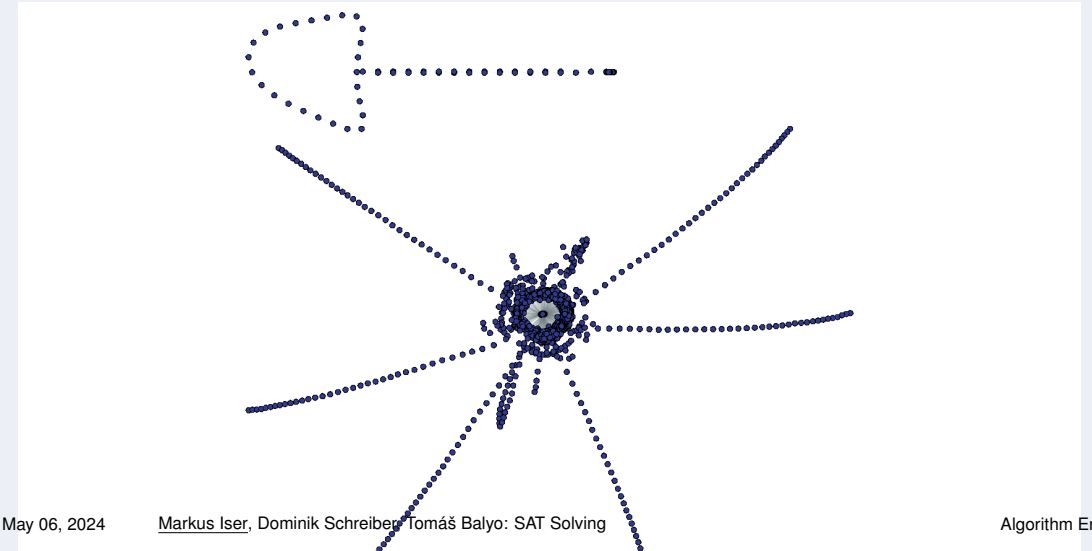
Instance: Newton SMT (SV Competition)

1000000 Conflicts (Relayout)



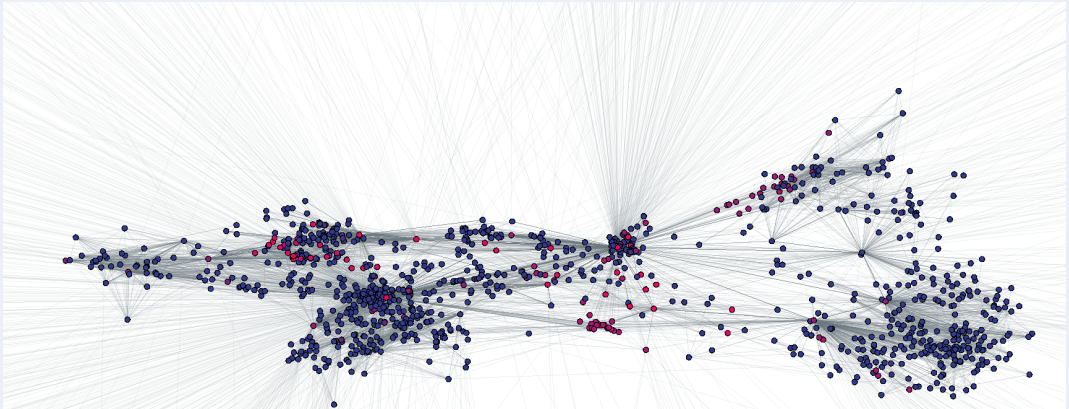
Instance: Newton SMT (SV Competition)

3000000 Conflicts (Relayout)



Instance: Newton SMT (SV Competition)

3500000 Conflicts (Core)



Preprocessing

General Idea

Conjecture: Smaller problems are easier to solve

$\Longrightarrow$ Try to reduce the size of the input formula by (polynomial time) simplification procedures.

- Subsumption
- Self-subsuming resolution
- Variable elimination
- Blocked clause elimination
- Failed literal elimination
- Autarkies

Subsumption

Subsumption

- Definition: A clause D subsumes a clause C iff $D \subseteq C$
- We also say that clause C is subsumed by D
- To check satisfiability, subsumed clauses are irrelevant: $\forall D \subseteq C, D \models C$

Example

$\{a, b\}$ subsumes $\{a, b, c\}$ and $\{a, b, d\}$

Self-Subsuming Resolution

Combination of Subsumption and Resolution

Self-Subsuming Resolution

- Definition: Let C, D be clauses and $\otimes_x$ the resolution operator on variable x. If $C \otimes D \subseteq C$ then C is said to be *self-subsumed by D with respect to x*.
- The resolvent of C and another clause D subsumes C

Example

- $C := \{\neg b, \neg e, f, \neg h\}$
- $D := \{\neg b, \neg e, \neg f\}$
- $E := C \otimes_f D = \{\neg b, \neg e, \neg h\}$
- The clause C is subsumed by the resolvent E

Variable Elimination

Variable Elimination

- Definition:
  - Let $S_x, S_{\overline{x}} \subset F$ be the sets of clauses containing x resp. $\overline{x}$
  - Let $R = \{C \otimes_x D \mid C \in S_x, D \in S_{\overline{x}}\}$ be the set of all resolvents on x
  - Modify F such that $F' := (F \setminus (S_x \cup S_{\overline{x}})) \cup R$
- The formulas F and F' are *satisfiability equivalent* (not equivalent)

Bounded Variable Elimination (BVE)

- Simulate variable elimination, but replace clauses only if number of clauses decreases (due to tautological resolvents)
- Most important simplification technique today

Blocked Clauses

Definition

A clause $(l \vee C)$ is blocked in F by l if either l is *pure* in F or if for every clause $(\neg l \vee D)$ in F the resolvent $(C \vee D)$ is a *tautology*.

Example

$F := (a \vee b) \wedge (a \vee \neg b \vee \neg c) \wedge (\neg a \vee c)$
First clause is not blocked, second is blocked by both a and $\neg c$, third is blocked by c.

Blocked Clause Elimination

Removal of an arbitrary blocked clause preserves satisfiability. Blocked clause elimination (BCE) has a unique fixpoint.

Blocked Clauses and Gates

Left-Totality of Gate Encodings

Given a gate G with output variable o and its clausal encoding E, it holds that for every clause $C \in E$ either $o \in C$ or $\overline{o} \in C$ (part 1) and all resolvents in $E[o] \otimes_o E[\overline{o}]$ are tautologic (part 2).

Proof of Part 1

Assume that there is a clause $C \in E$ such that $o \notin \text{vars}(C)$. It follows that there exists an assignment to input variables which falsifies C for any assignment to o. This contradicts left-totality.

Proof of Part 2

Let R be a non-tautological resolvent in $E[o] \otimes_o E[\overline{o}]$. By Definition of resolution, it holds that $o \notin \text{vars}(R)$ and $E \models R$. This contradicts left-totality.

Variable Elimination and Gates

Property of Gate Encodings

- Resolving the clauses of a gate results in tautological clauses
- Example: Tseitin encoding of gate $x = AND(y, z)$ results in the clauses $\{\neg x, y\}, \{\neg x, z\}, \{x, \neg y, \neg z\}$

Idea: Variable Elimination for Gate Encoding G

- Split formula F into $F = G \cup R$, where G are the gate clauses
- Apply variable elimination:

$$\begin{aligned} S' &= (G_x \cup R_x) \otimes (G_{\overline{x}} \cup R_{\overline{x}}) \\ &= (G_x \otimes R_{\overline{x}}) \cup (R_x \otimes G_{\overline{x}}) \cup (R_x \otimes R_{\overline{x}}) \cup \cancel{(G_x \otimes G_{\overline{x}})} \\ &= (G_x \otimes R_{\overline{x}}) \cup (R_x \otimes G_{\overline{x}}) \cup \cancel{(R_x \otimes R_{\overline{x}})} \\ &= (R_x \otimes G_{\overline{x}}) \cup (R_{\overline{x}} \otimes G_x) \end{aligned}$$

- $(G_x \otimes R_{\overline{x}}) \cup (R_x \otimes G_{\overline{x}}) \models (R_x \otimes R_{\overline{x}})$ (Why? $\rightarrow$ *Exercise*)

Failed Literal Elimination

Definition

If $F \wedge \{l\} \vdash_{UP} \bot$, i.e., applying unit propagation on $F \wedge \{l\}$ derives UNSAT, replace F by $F \wedge \{\neg l\}$.

Generalization

If $(F \setminus \{C\}) \wedge \neg C \vdash_{UP} \bot$, remove C from F.

Autarkies

Definition

Given a partial assignment σ and a formula F, a clause $C \in F$ is *touched by* σ if it contains the negation of a literal assigned in σ. A clause is *satisfied by* σ if it contains a literal assigned to *True* by σ. If all touched clauses are satisfied then σ is an *autarky*.

All clauses touched by an autarky can be removed.

Autarky-based Clause Removal

The partial assignment $\sigma = \{\neg a, \neg c\}$ is an autarky for $F := \{\neg a, b\}, \{\neg a, c\}, \{a, \neg b, \neg c\}, \{b, d\}, \ldots$ (more clauses without a and c)

Preprocessing Techniques that do not Reduce the Problem Size

- Bounded Variable Addition (BVA) a.k.a. Extension rule
- Some formulas have refutations of exponential size in the resolution calculus, but of polynomial size in extended resolution, e.g., pigeonhole formulas, mutilated chessboard, ...

Extended Resolution

Extended resolution adds a second rule to the resolution calculus, the Extension Rule. The idea is to introduce new variables as conjunction of existing literals, $x_{\text{new}} \leftrightarrow l_1 \wedge l_2$. As a rule for formulas in CNF:

$$\overline{(\neg x_{\text{new}} \vee l_1) \wedge (\neg x_{\text{new}} \vee l_2) \wedge (x_{\text{new}} \vee \neg l_1 \vee \neg l_2)}$$

Inprocessing

Idea: Interleave search and preprocessing

- Preprocessing can be extremely beneficial
- Most solvers in SAT competitions use bounded variable elimination, subsumption and self-subsuming resolution
- Problem: Many preprocessing techniques, though polynomial, require considerable time
- Possible Solutions:
  - Interrupt preprocessing techniques after some time
  - Resume preprocessing between restarts
  - Limit preprocessing time in relation to search

The End.

Recap

-