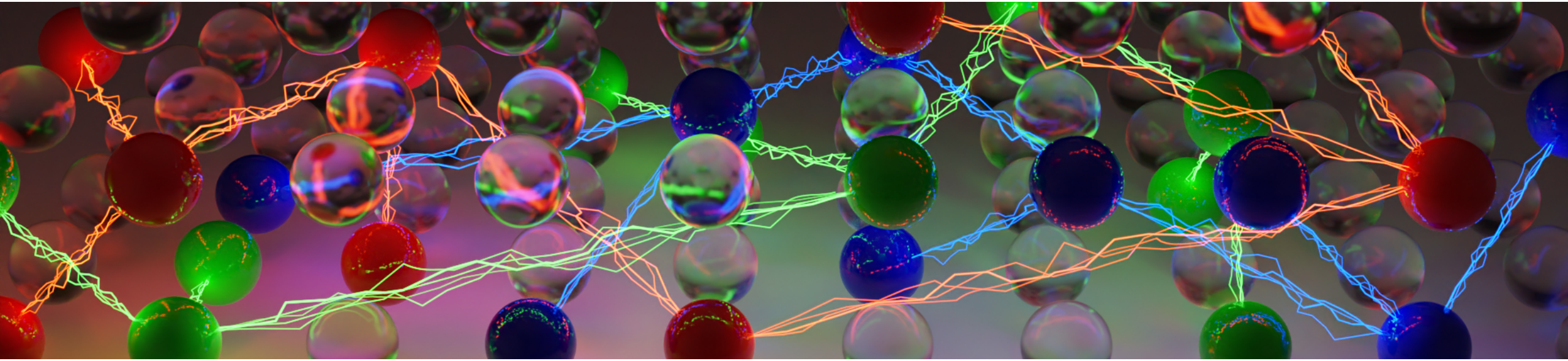
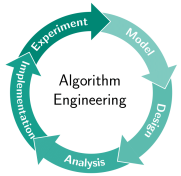


Practical SAT Solving



Lecture 7 – Conflict-Driven Clause Learning

Ashlin Iser, Dominik Schreiber | June 08, 2026

SAtRes
Scalable Automated Reasoning

Overview

Recap.

Lecture 6: Classic Heuristics and Modern SAT Solving 1:

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

Today's Topic: Modern SAT Solving 2

- Efficient Unit Propagation
- Clause Forgetting
- Modern Decision Heuristics

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

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

Hot Paths in CDCL Solvers

heat	$\varnothing$ *per sec.*[1]	
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*

[1] 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	$\bot$

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	$\bot$

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	$\bot$

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	$\varnothing$ *per sec.*[2]
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

[2] 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	$\varnothing$ *per sec.*[2]
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?

[2] 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	$\varnothing$ *per sec.*[2]
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?

[2] 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	$\varnothing$ *per sec.*[2]
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?)

[2] 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	$\perp$
2	c	$\perp$

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	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

Two Watched Literals: Optimizations

heat	∅ *per sec.*[3]
Clause Access	
Iterate Occurrences	
Propagation	$\sim$ **10^6**

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

→ **Reduced Load in Occurrence Tracking**

Optimization 1: Keep watched literals the first two in clause
→ 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
→ Skip clause access if that literal is satisfied

[3] 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)
- Further optimizations:
  - **Invariant:** the two first literals in each clause are the watched ones
  - **Blocking literals:** 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.[4]

Problem:

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

Solution:

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

[4] “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.[4]

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”?**

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

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 (EMA)

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 (EMA)

- **Literal Block Distance (LBD)**

  Keep clauses with a low number of decision levels[5]

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

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 (EMA)

- **Literal Block Distance (LBD)**

  Keep clauses with a low number of decision levels[5]

  Why is low LBD good?

[5] 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

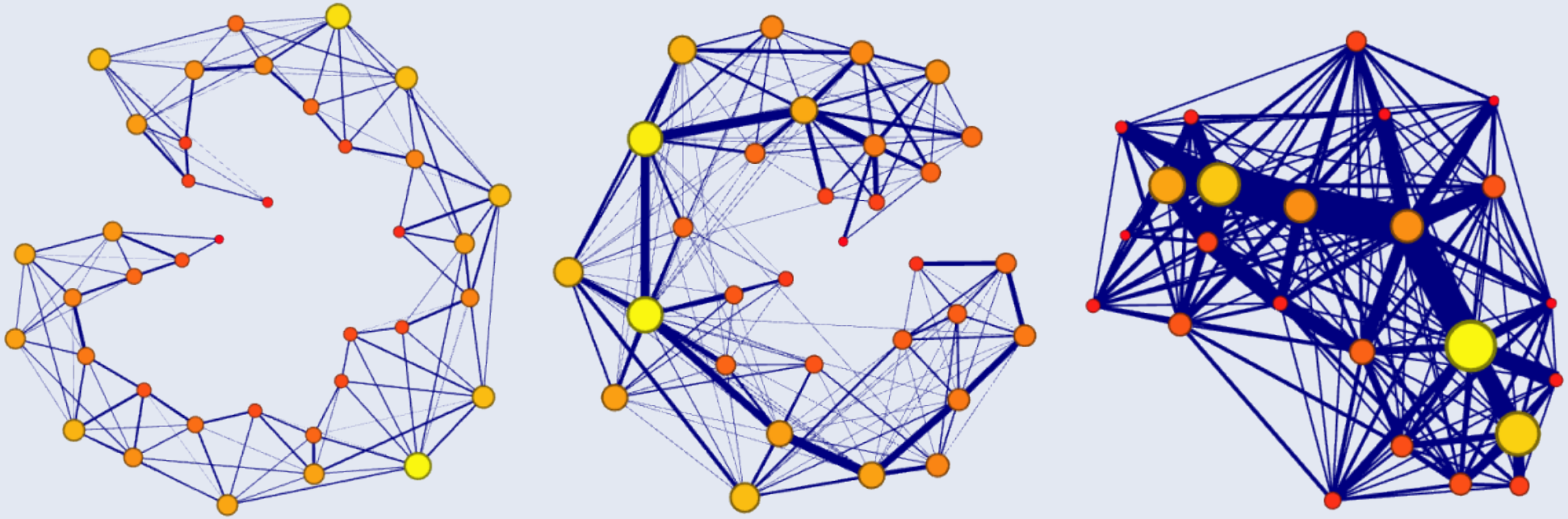


Image Source: "Community Structure in Industrial SAT Instances", Ansotegui et al., AIJ 2019

Clause Forgetting: Modern Hybrid Approach

Manage clauses differently in three tiers

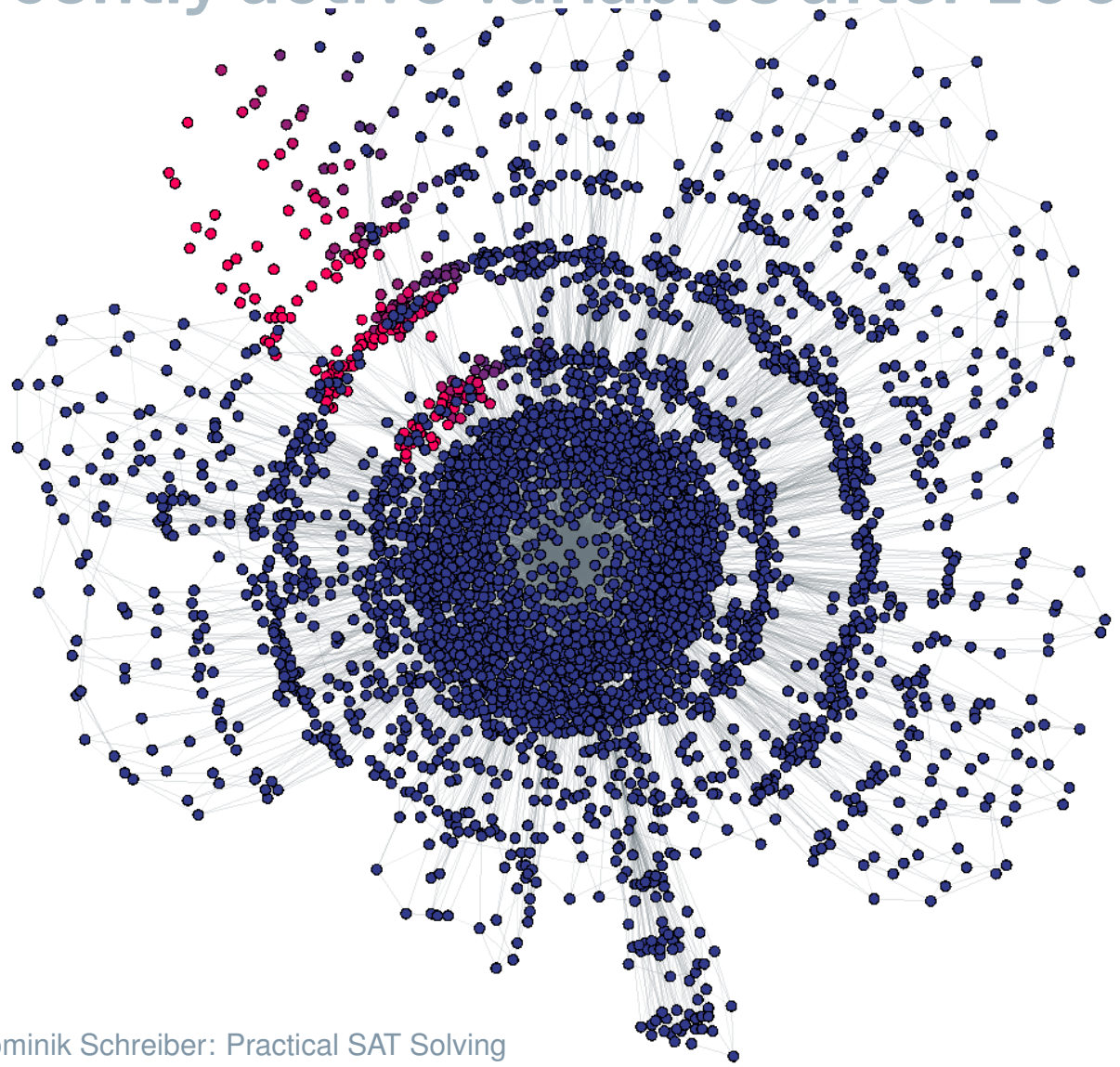
Tier	Strategy	Description
`core`	LBD	Permanently store clauses of LBD $\leq k$ (core-cut value, 3 in practice)
`mid-tier`	LRU	Clauses stay here if used in recent conflicts
`local`	LRU	Keep fixed number of clauses (say 5000) of highest activity

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: The award-winning SAT solver `MapleCOMSPS (2016)` is a CoMinisatPS fork

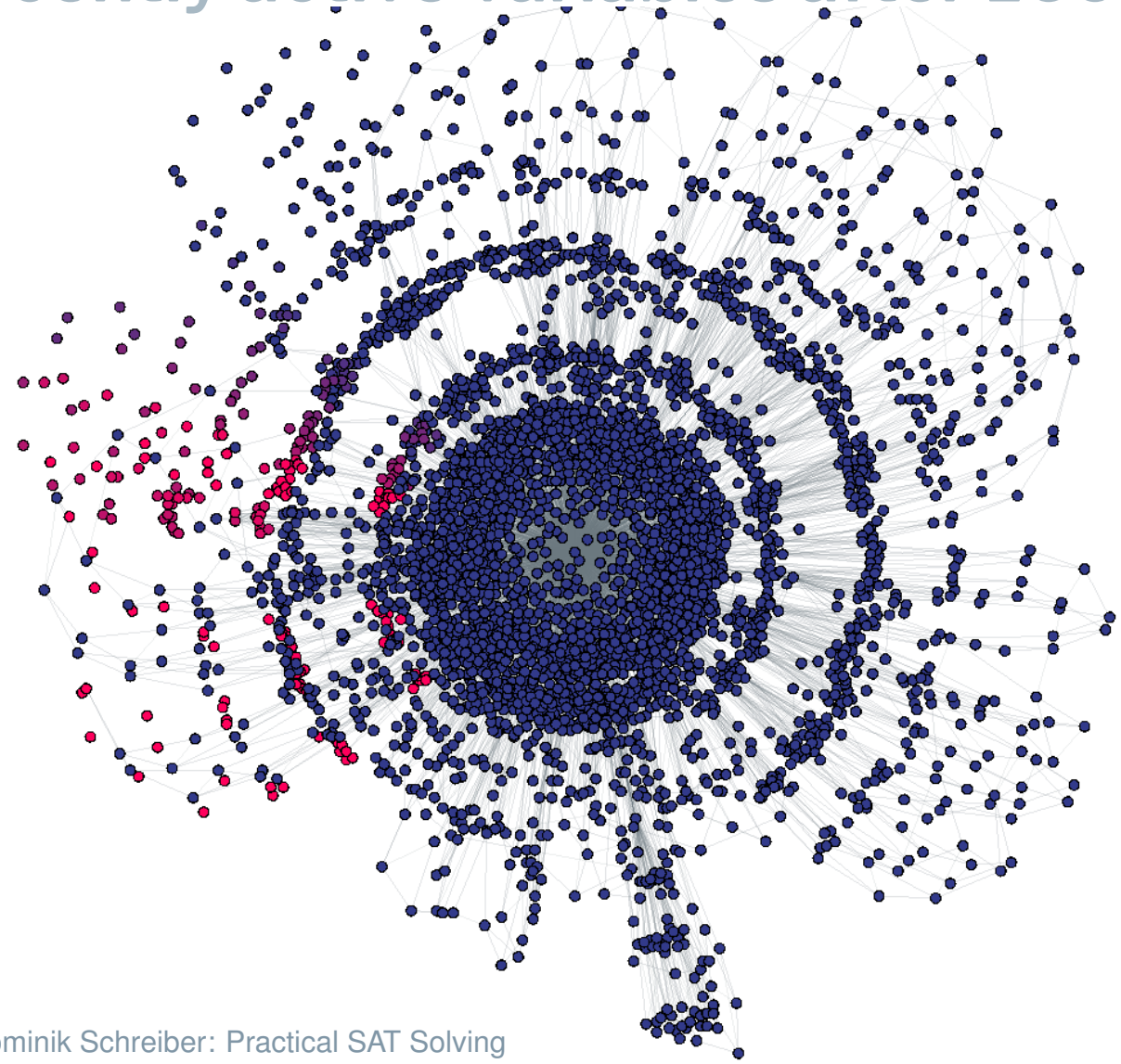
Instance from Termination Analysis, SAT

initial layout, recently active variables after 1000 conflicts



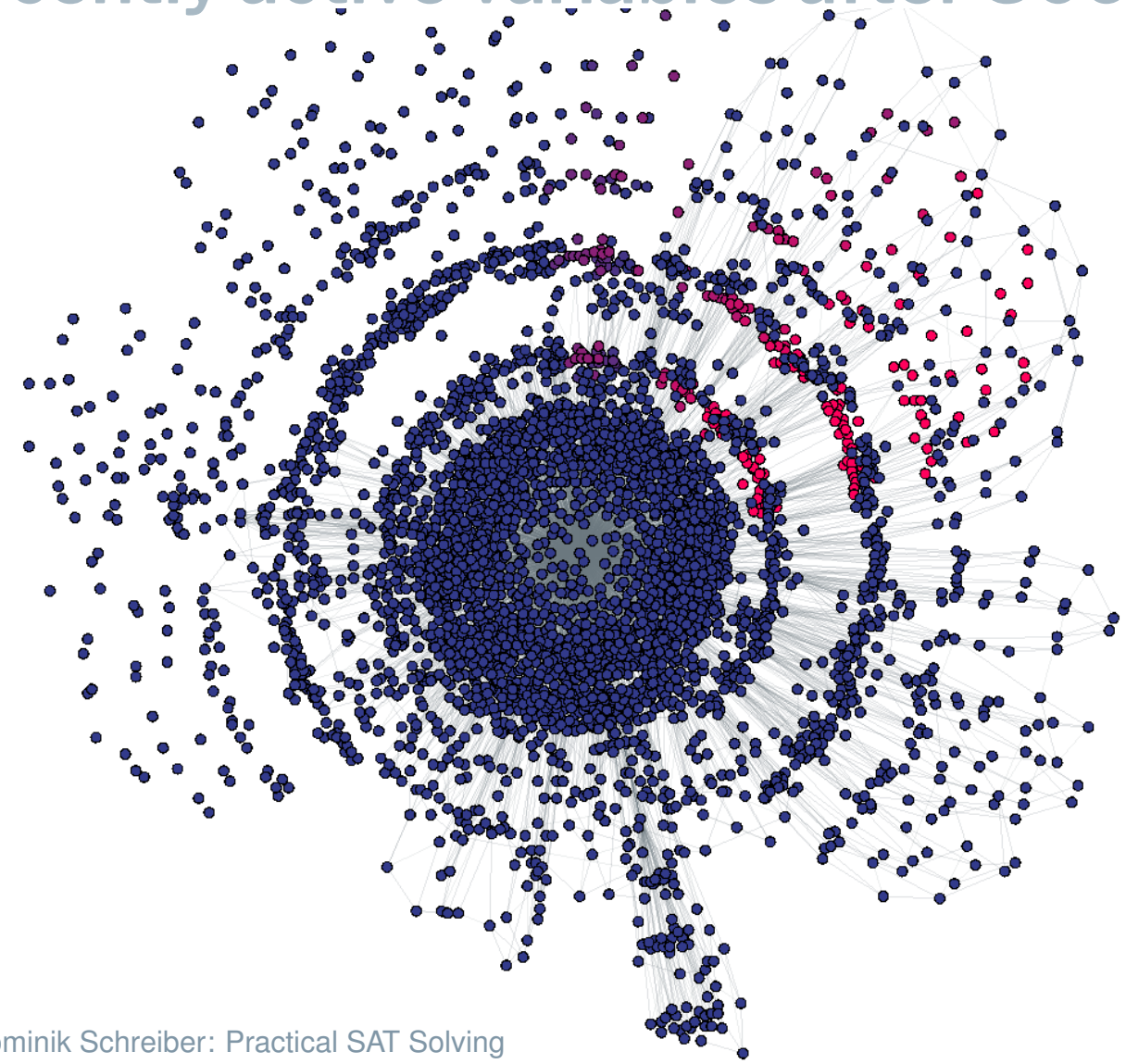
Instance from Termination Analysis, SAT

initial layout, recently active variables after 1690 conflicts



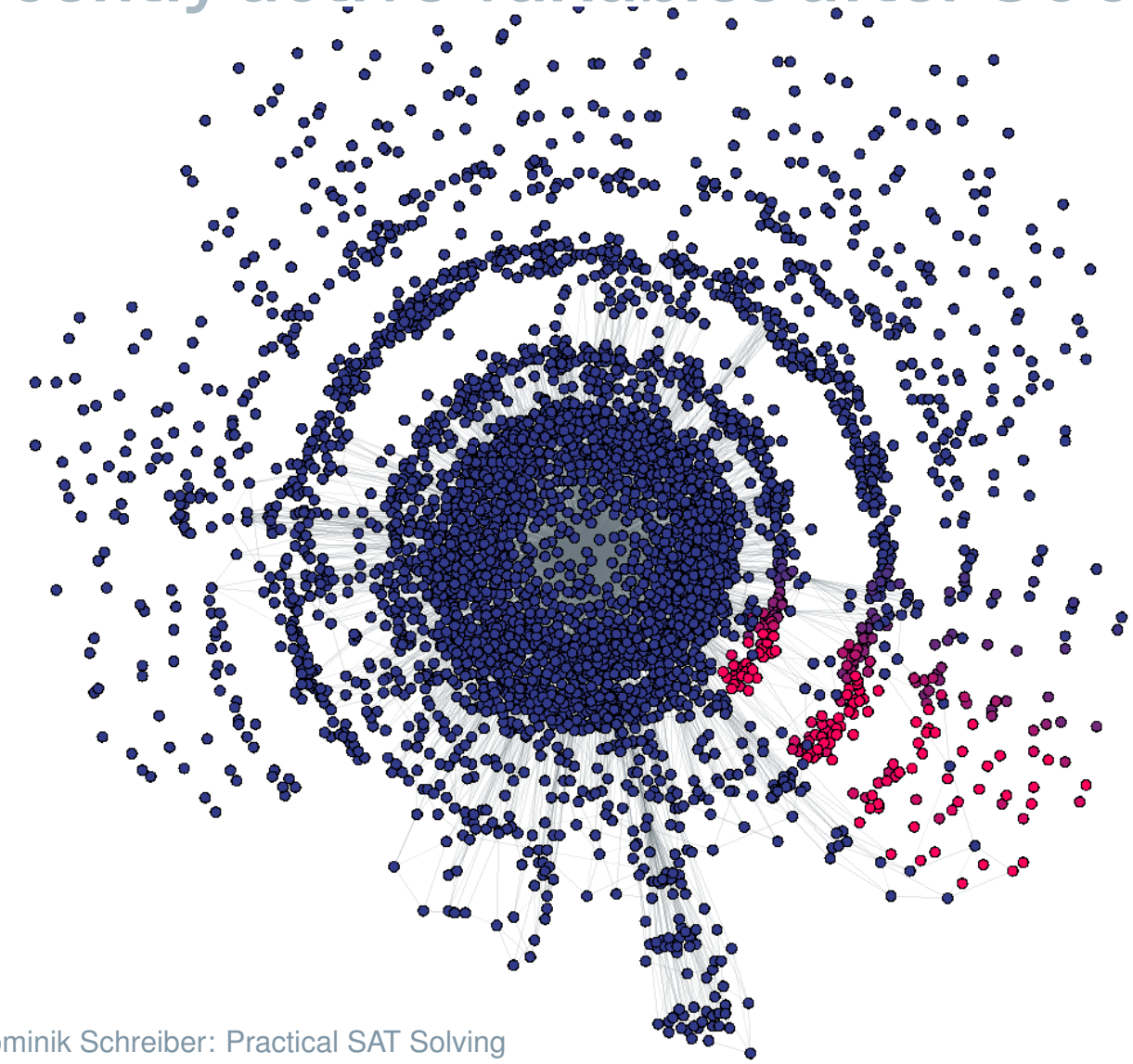
Instance from Termination Analysis, SAT

initial layout, recently active variables after 3090 conflicts



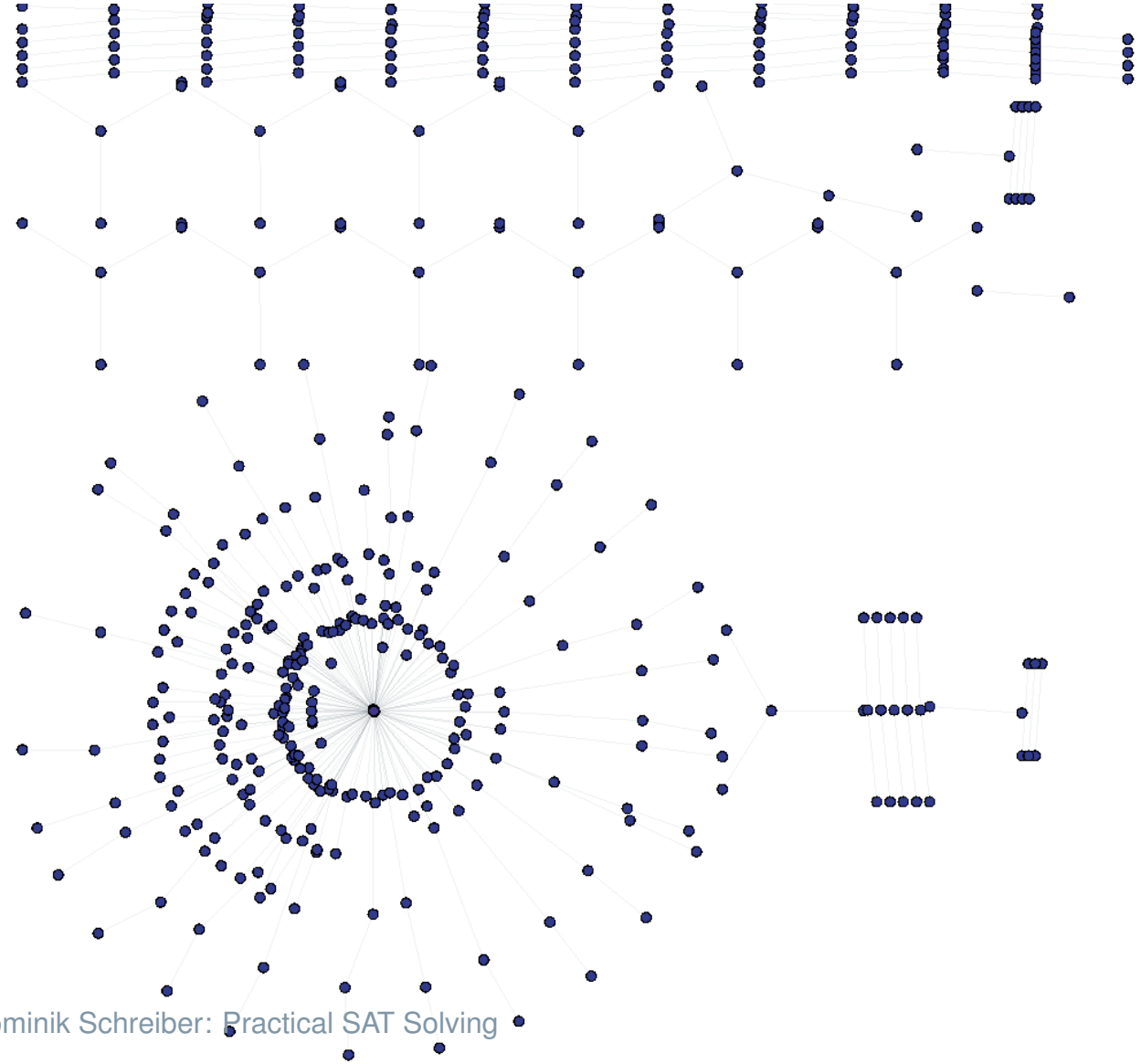
Instance from Termination Analysis, SAT

initial layout, recently active variables after 5000 conflicts



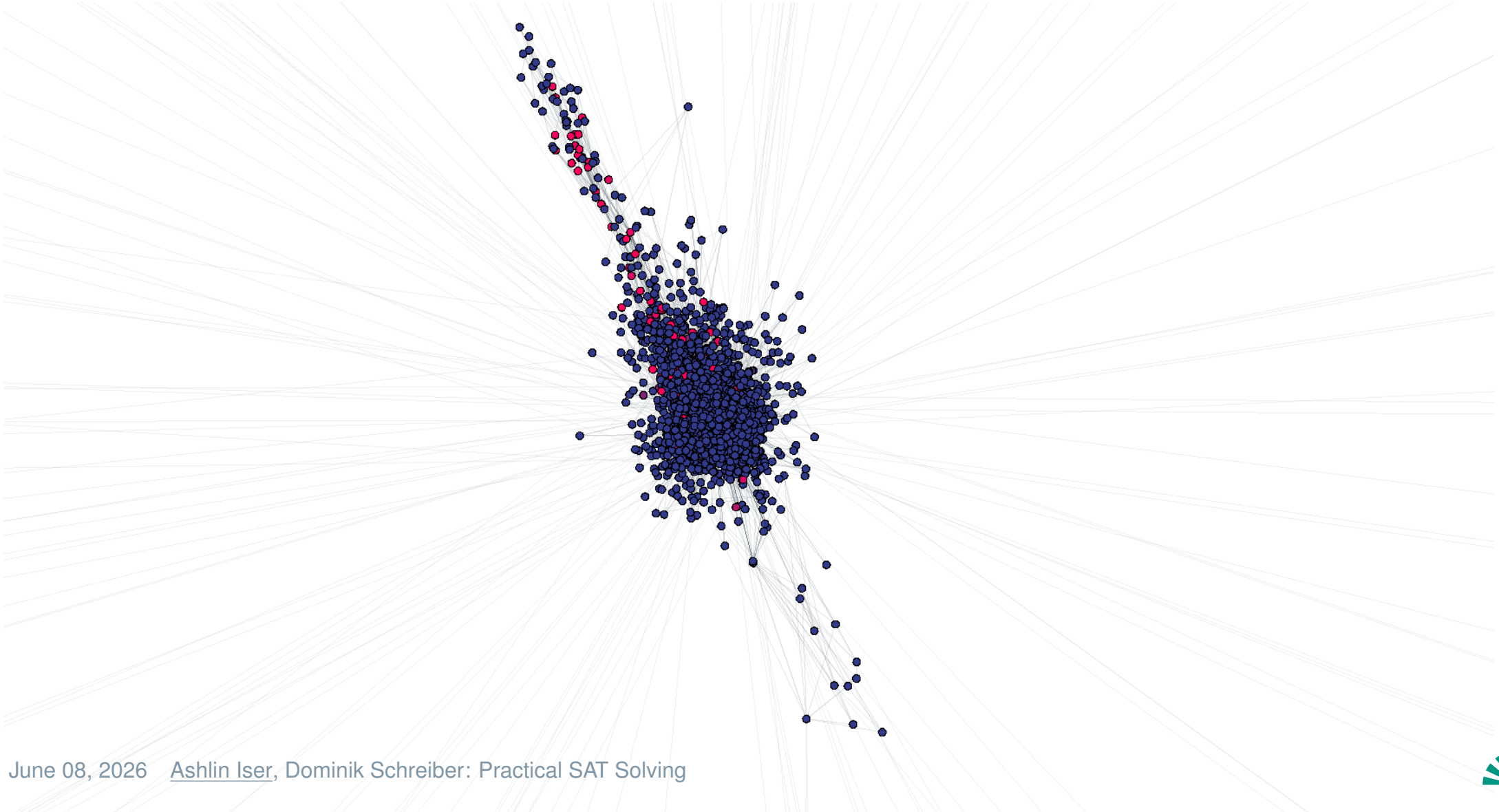
Instance from Termination Analysis, SAT

relayout after 6000 conflicts



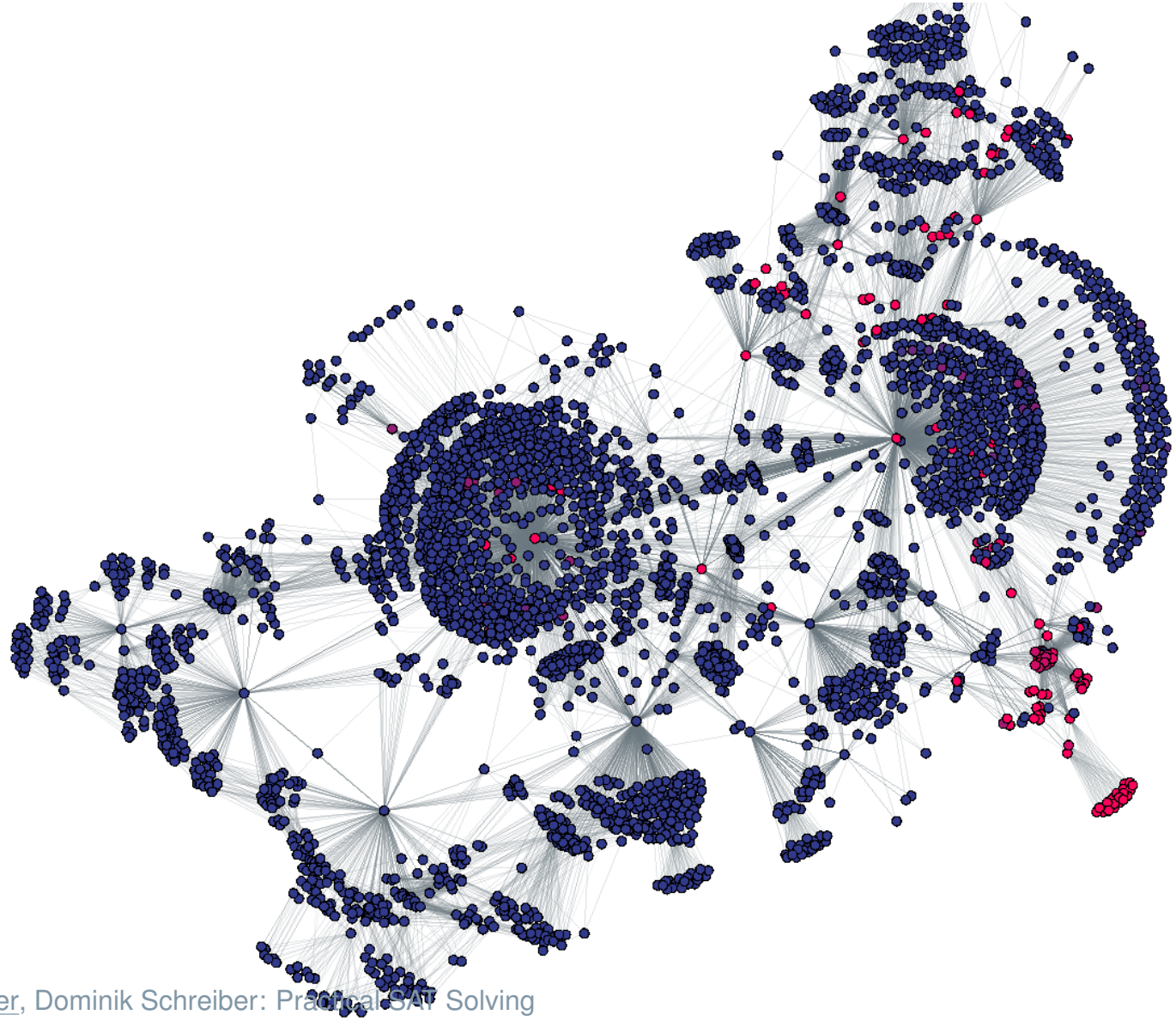
Instance from Termination Analysis, SAT

core after 52500 conflicts



Instance from SV Competition, SAT

initial layout

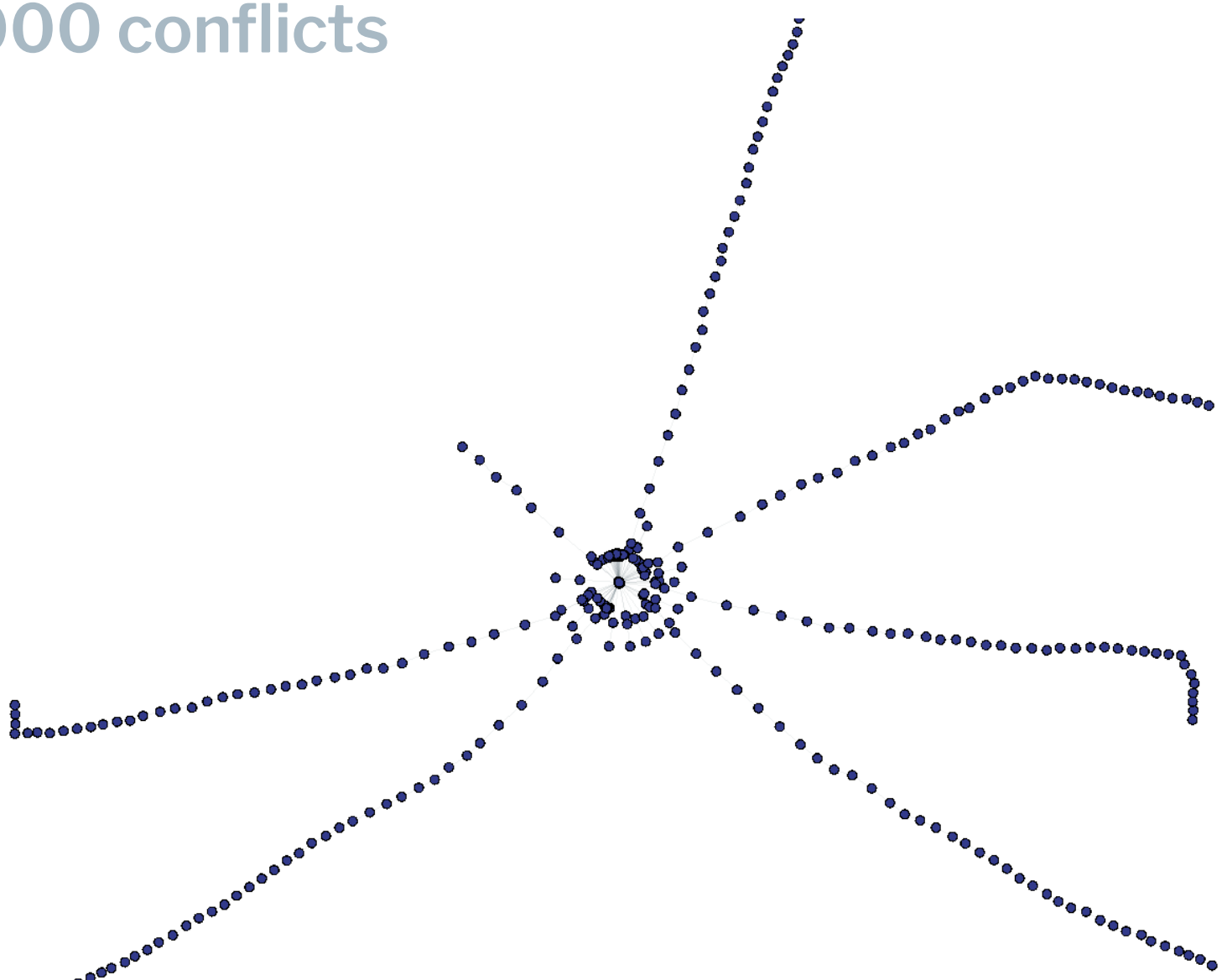


Instance from SV Competition, SAT

after 10000 conflicts

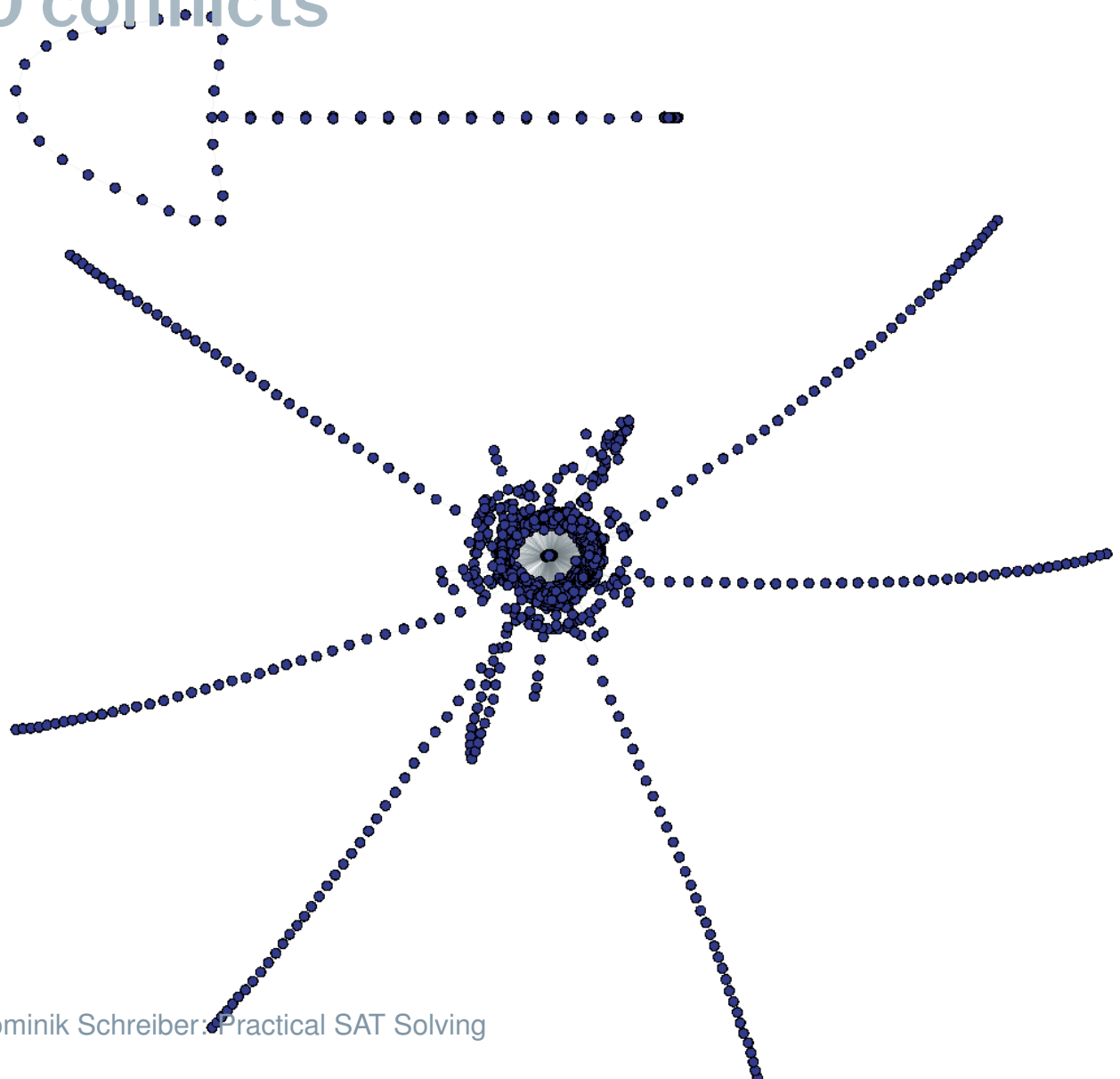
Instance from SV Competition, SAT

after 1000000 conflicts



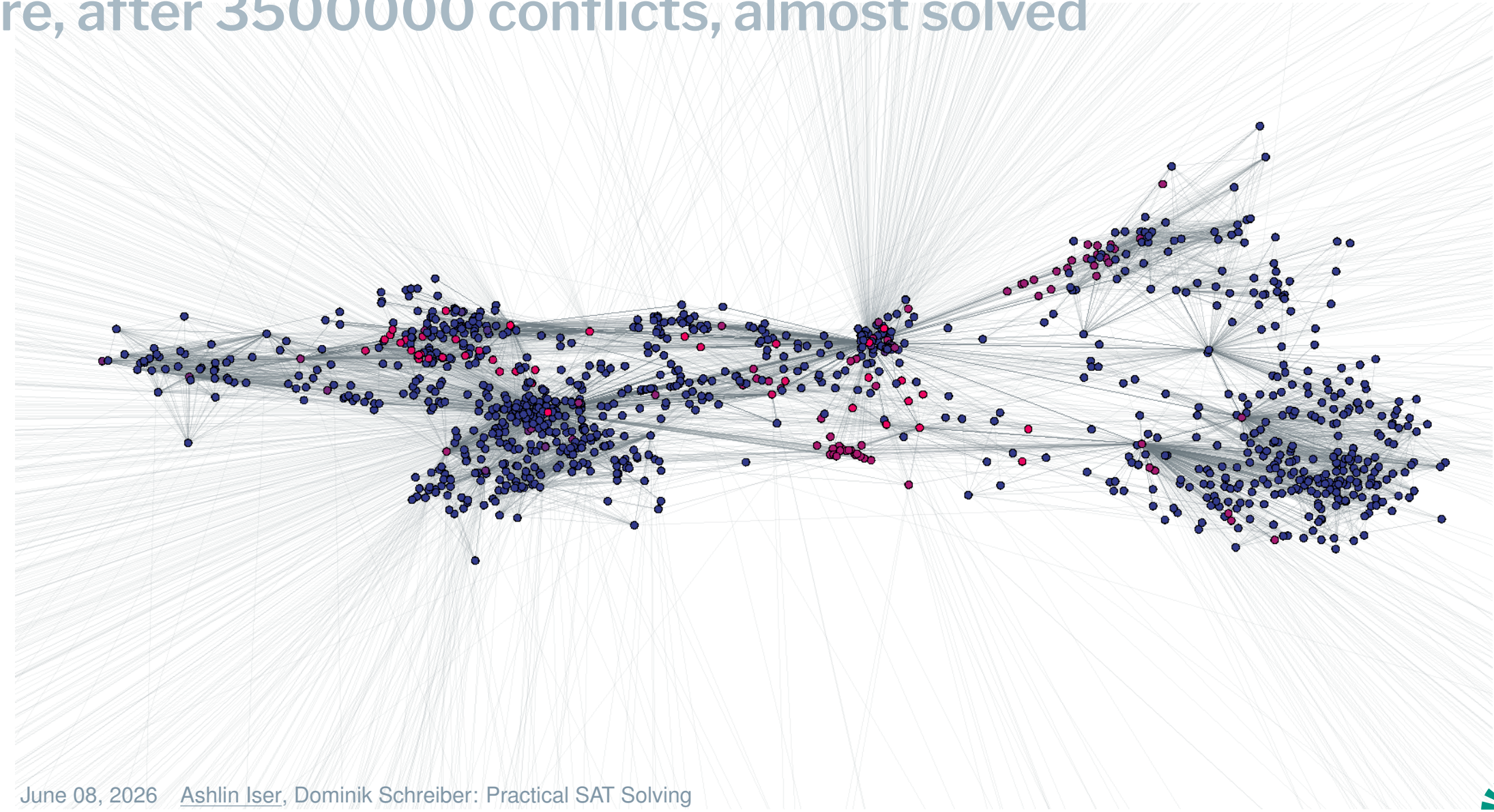
Instance from SV Competition, SAT

after 3000000 conflicts



Instance from SV Competition, SAT

core, after 3500000 conflicts, almost solved



Recap

So far

- Efficient Unit Propagation
- Clause Forgetting Heuristics

Next Up

Modern Decision Heuristics

Variable State Independent Decaying Sum (VSIDS)

VSIDS Heuristic

Implemented in most CDCL solvers. First presented in SAT solver Chaff.[6]

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

[6] 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)

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][7]

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

Variable State Independent Decaying Sum (VSIDS)

Historic Implementations

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)

Idea: Exponential decay of scores $s(v)$ with damping factor $0 < f < 1$

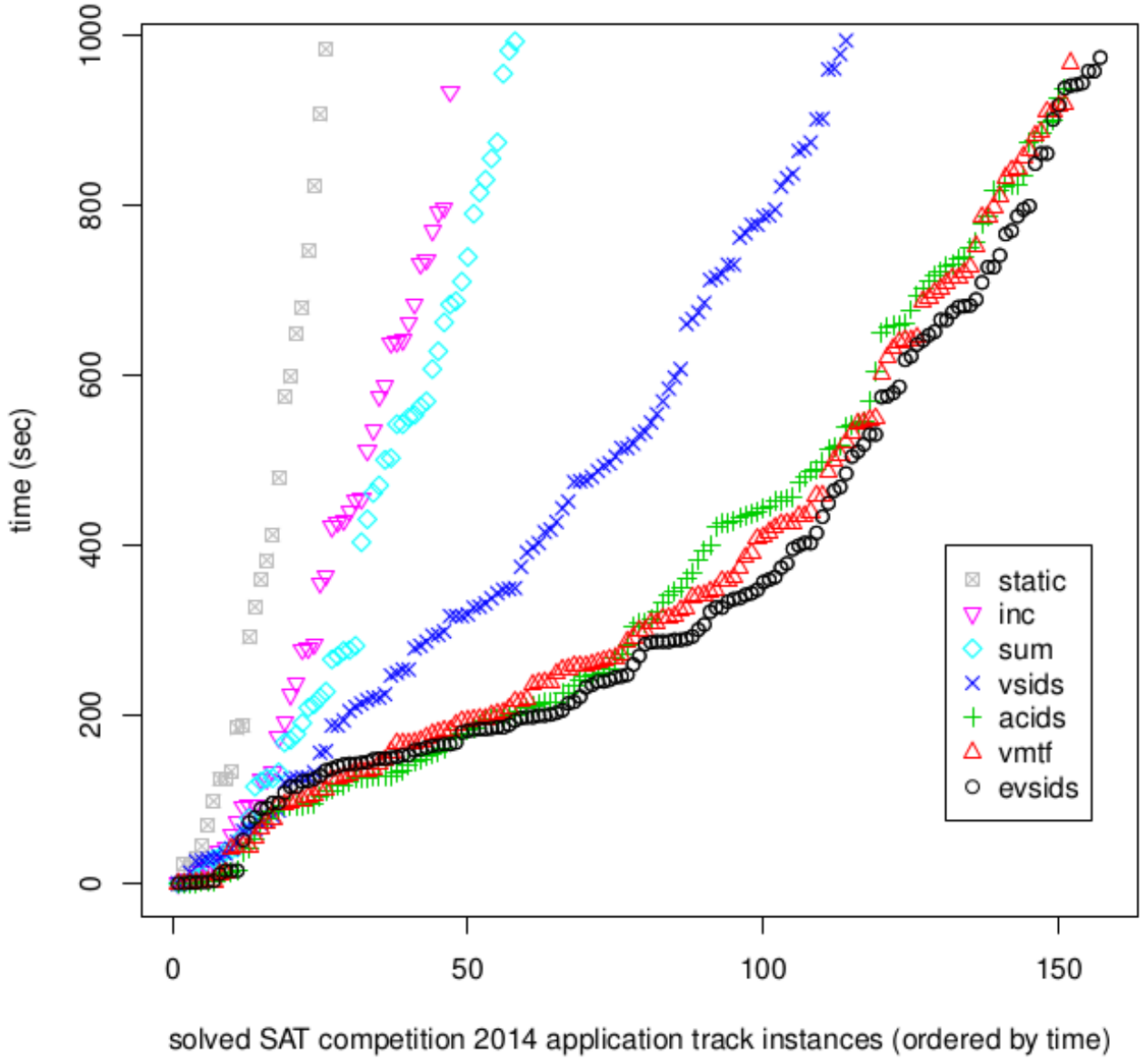
$$s'(v) := \begin{cases} f \cdot s(v) + (1 - f) & \text{if } v \text{ is to be bumped} \\ f \cdot s(v) & \text{otherwise} \end{cases}$$

Theory: Exponential Moving Average (EMA)

Implementation: Score increment by g^i, with i denoting the conflict-index and $g = \frac{1}{f}$ (no decay)

Reason-side Bumping: Also bump variables in the reason clauses of the conflict

Evaluating CDCL Variable Scoring Schemes (Biere & Fröhlich, 2015)



Alternative and Hybrid Approaches

Alternatives

Siege (2004): Variable Move To Front (VMTF)

HaifaSAT (2008): Clause Move To Front (CMTF)

Recent 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

Modern Branching: Local Search Intergration

Better Decision Heuristics in CDCL through Local Search and Target Phases (Cai et al., 2022)

Branching:

- Target Phases: cache and use the phases which led to the previously largest assignment
- Integrate Sprints of Local Search: use unit-propagation to complete the assignment (ignoring all conflicts)
- Rephasing: save the best assignment found during local search for phase selection (diversification)

Ordering:

- Import Statistics: use frequency of appearing in unsatisfied clauses to modify the variables VSIDS score

Recap

Recap

- Efficient Unit Propagation
- Clause Forgetting
- Modern Decision Heuristics

Next Time

- Preprocessing

Algorithm 2: 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
```