CIS 189

Lecture 5: Modern Techniques in SAT Solving

Jediah Katz jediahk@seas.upenn.edu

Recap: Iterative DPLL

```
dpll(φ):
    if unit_propagate() = CONFLICT: return UNSAT
    while not all variables have been set:
        let x = pick_variable()
        create new decision level
        set x = T
        while unit_propagate() = CONFLICT:
            if decision_level = 0: return UNSAT
            backtrack()
            set x = F
    return SAT
```

Preprocessing

- **Idea:** make poly-time modifications to boolean formula before performing main DPLL search procedure
  - "Clean" redundant information
- Generally reduces size of formula, but not always
- Simple preprocessing: **deduplication**
  - Remove duplicate literals in clause: $(1 \lor 2 \lor 1 \lor 3)$
  - Remove duplicate clauses: $(1 \lor 2) \land (2 \lor 3) \land (1 \lor 2)$

Preprocessing: Pure Literals

- Literal x is **pure** in formula φ if $\overline{x}$ doesn't appear in φ
- Pure literals can always be set to True
- **Pure literal elimination:** remove all clauses containing a pure literal from φ

$$\left(1 \vee \boxed{\overline{2}}\right) \wedge \left(\boxed{3} \vee \overline{1} \vee \overline{4}\right) \vee \left(\boxed{\overline{2}} \vee \boxed{3} \vee 4\right)$$

- Until early 2000s, pure literal elimination was used in DPLL as a 2nd inference rule along with UP. Why do you think this stopped?

Preprocessing: Subsumption

- Clause D is **subsumed by** clause C if $C \subseteq D$
  - If C is satisfied, then D is also satisfied
  - Can remove all subsumed clauses from formula

$$C: \ (1 \vee \overline{2} \vee 3)$$

$$D: \ (1 \vee \overline{2} \vee 3 \vee \overline{4} \vee \overline{5}) \ ✗$$

Chronological Backtracking

- DPLL uses **chronological backtracking:** when we find a conflict, backtrack to the previous decision level
- **Issue:** might reach conflicts (contradictions) caused by the same underlying reason over and over again

Chronological Backtracking

$$(1 \vee \overline{2})$$

$$(\overline{1} \vee 3 \vee 4)$$

$$(\overline{1} \vee \overline{3} \vee 4)$$

$$(\overline{1} \vee 3 \vee \overline{4})$$

$$(\overline{1} \vee \overline{3} \vee \overline{4})$$

Chronological Backtracking

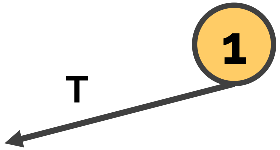
$$(1 \vee \overline{2})$$

$$(\overline{1} \vee 3 \vee 4)$$

$$(\overline{1} \vee \overline{3} \vee 4)$$

$$(\overline{1} \vee 3 \vee \overline{4})$$

$$(\overline{1} \vee \overline{3} \vee \overline{4})$$



Chronological Backtracking

$$(1 \vee \overline{2})$$

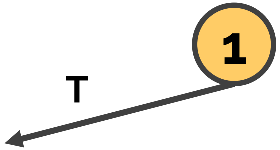
$$(\overline{1} \vee 3 \vee 4)$$

$$(\overline{1} \vee \overline{3} \vee 4)$$

$$(\overline{1} \vee 3 \vee \overline{4})$$

$$(\overline{1} \vee \overline{3} \vee \overline{4})$$

UNSAT subformula



Chronological Backtracking

$(1 \lor \overline{2})$

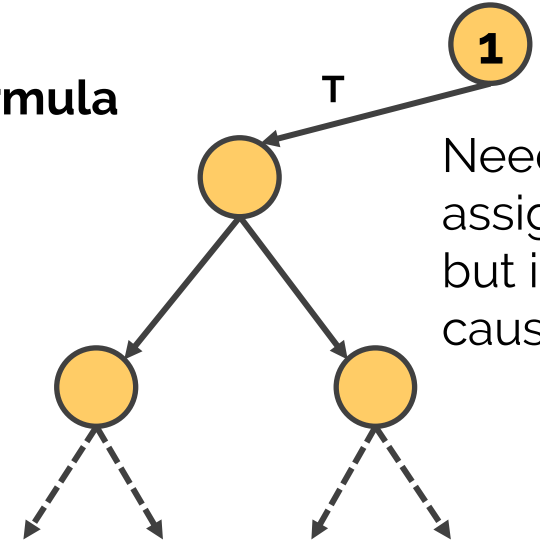
$(\overline{1} \lor 3 \lor 4)$

$(\overline{1} \lor \overline{3} \lor 4)$

$(\overline{1} \lor 3 \lor \overline{4})$

$(\overline{1} \lor \overline{3} \lor \overline{4})$

UNSAT subformula



Need to rule out all assignments of 2, 3, 4, but issue was really caused by $1 = T$!

Backjumping

- Not every decision actually contributes to a conflict
- **Idea:** upon conflict, instead of backtracking one level to the last decision, **backjump** to an *important* decision
  - i.e., a decision that contributed to the conflict
- But how do we know what is an important decision?

Implication Graphs

- An **implication graph** G is a DAG whose vertices are literal assignments at a particular decision level
  - Ex: $\overline{x}@3$ represents setting x to False at level 3
  - Assignments can be decisions or due to unit propagation
- Can also contain special vertex $\perp$ representing a conflict
- There is an edge $x@i \rightarrow y@j$ if the assignment $x@i$ implied the assignment $y@j$
  - i.e., $y@j$ was set by unit propagation from a clause containing $\overline{x}$

Implication Graphs

$c_1: \quad \overline{2} \vee \overline{3} \vee \overline{4}$

$c_2: \quad \overline{3} \vee \overline{5} \vee \overline{6}$

$c_3: \quad 4 \vee 6 \vee 7$

$c_4: \quad \overline{7} \vee \overline{8}$

$c_5: \quad \overline{1} \vee \overline{7} \vee \overline{9}$

$c_6: \quad \overline{1} \vee 8 \vee 9$

Implication Graphs

c_1: $\overline{2} \vee \overline{3} \vee \overline{4}$

c_2: $\overline{3} \vee \overline{5} \vee \overline{6}$

c_3: $4 \vee 6 \vee 7$

c_4: $\overline{7} \vee \overline{8}$

c_5: $\overline{1} \vee \overline{7} \vee \overline{9}$

c_6: $\overline{1} \vee 8 \vee 9$

Implication Graphs

c_1: $\overline{2} \vee \overline{3} \vee \overline{4}$

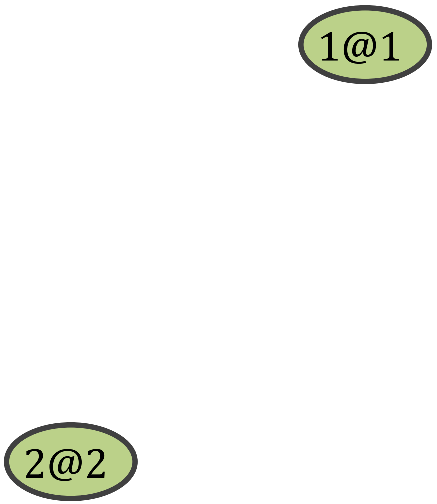
c_2: $\overline{3} \vee \overline{5} \vee \overline{6}$

c_3: $4 \vee 6 \vee 7$

c_4: $\overline{7} \vee \overline{8}$

c_5: $\overline{1} \vee \overline{7} \vee \overline{9}$

c_6: $\overline{1} \vee 8 \vee 9$



Implication Graphs

$c_1: \quad \overline{2} \vee \overline{3} \vee \overline{4}$

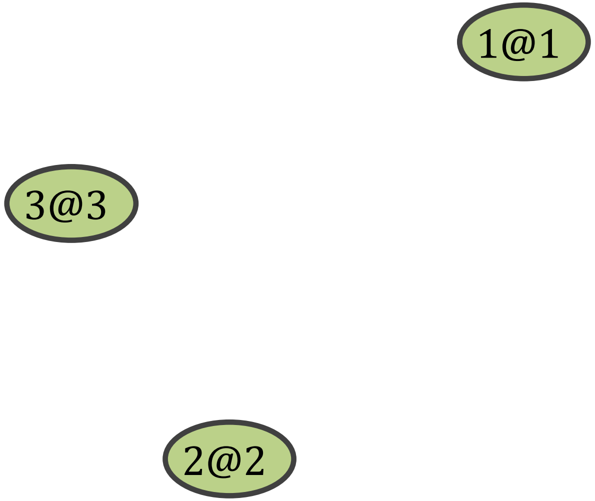
$c_2: \quad \overline{3} \vee \overline{5} \vee \overline{6}$

$c_3: \quad 4 \vee 6 \vee 7$

$c_4: \quad \overline{7} \vee \overline{8}$

$c_5: \quad \overline{1} \vee \overline{7} \vee \overline{9}$

$c_6: \quad \overline{1} \vee 8 \vee 9$



Implication Graphs

c_1: $\overline{2} \vee \overline{3} \vee \overline{4}$

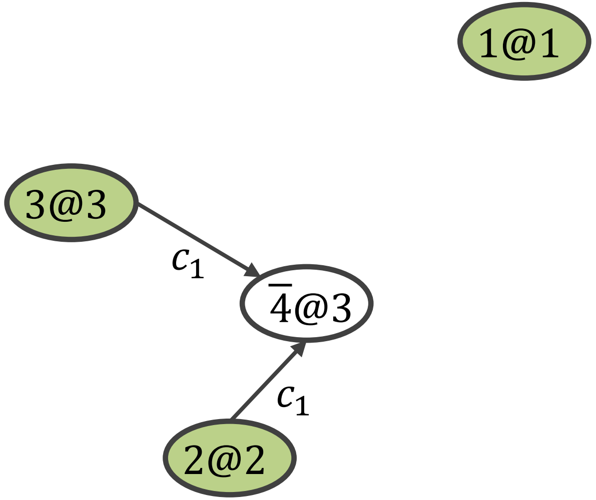
c_2: $\overline{3} \vee \overline{5} \vee \overline{6}$

c_3: $4 \vee 6 \vee 7$

c_4: $\overline{7} \vee \overline{8}$

c_5: $\overline{1} \vee \overline{7} \vee \overline{9}$

c_6: $\overline{1} \vee 8 \vee 9$



Implication Graphs

c_1: $\overline{2} \vee \overline{3} \vee \overline{4}$

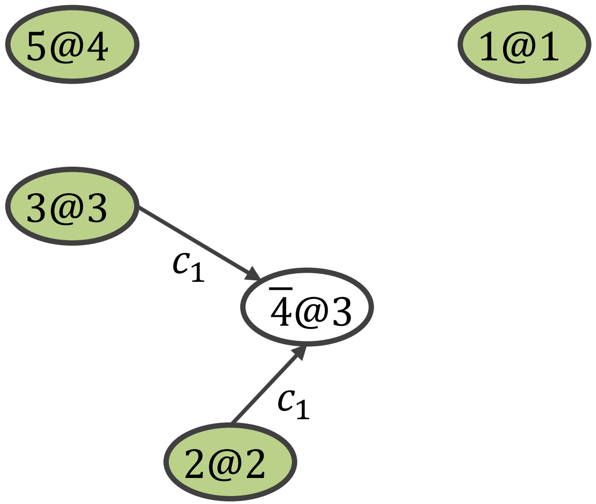
c_2: $\overline{3} \vee \overline{5} \vee \overline{6}$

c_3: $4 \vee 6 \vee 7$

c_4: $\overline{7} \vee \overline{8}$

c_5: $\overline{1} \vee \overline{7} \vee \overline{9}$

c_6: $\overline{1} \vee 8 \vee 9$



Implication Graphs

c_1: $\overline{2} \vee \overline{3} \vee \overline{4}$

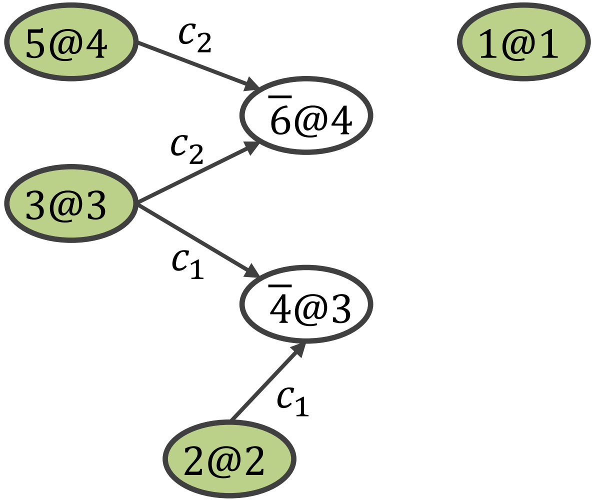
c_2: $\overline{3} \vee \overline{5} \vee \overline{6}$

c_3: $4 \vee 6 \vee 7$

c_4: $\overline{7} \vee \overline{8}$

c_5: $\overline{1} \vee \overline{7} \vee \overline{9}$

c_6: $\overline{1} \vee 8 \vee 9$



Implication Graphs

c_1: $\overline{2} \vee \overline{3} \vee \overline{4}$

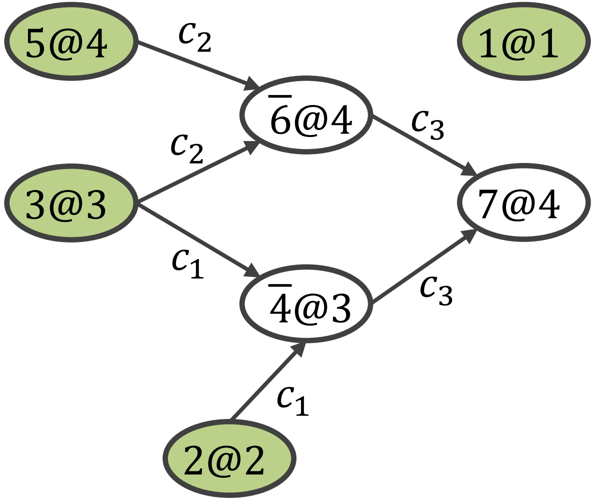
c_2: $\overline{3} \vee \overline{5} \vee \overline{6}$

c_3: $4 \vee 6 \vee 7$

c_4: $\overline{7} \vee \overline{8}$

c_5: $\overline{1} \vee \overline{7} \vee \overline{9}$

c_6: $\overline{1} \vee 8 \vee 9$



Implication Graphs

c_1: $\overline{2} \vee \overline{3} \vee \overline{4}$

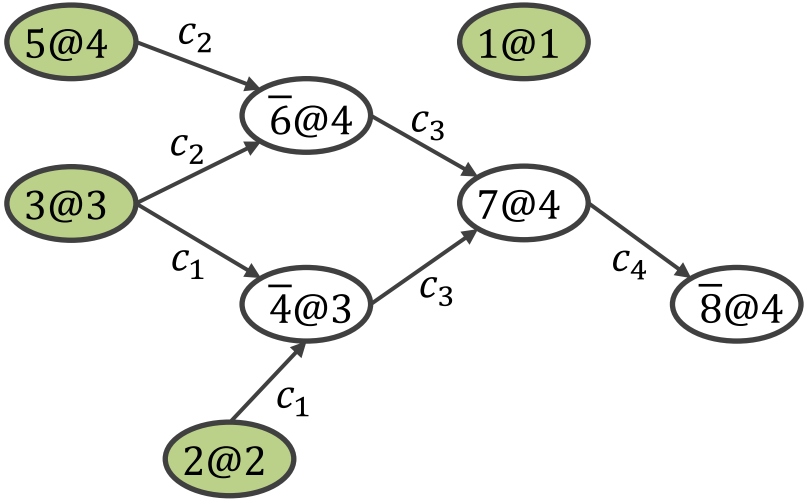
c_2: $\overline{3} \vee \overline{5} \vee \overline{6}$

c_3: $4 \vee 6 \vee 7$

c_4: $\overline{7} \vee \overline{8}$

c_5: $\overline{1} \vee \overline{7} \vee \overline{9}$

c_6: $\overline{1} \vee 8 \vee 9$



Implication Graphs

c_1: $\overline{2} \vee \overline{3} \vee \overline{4}$

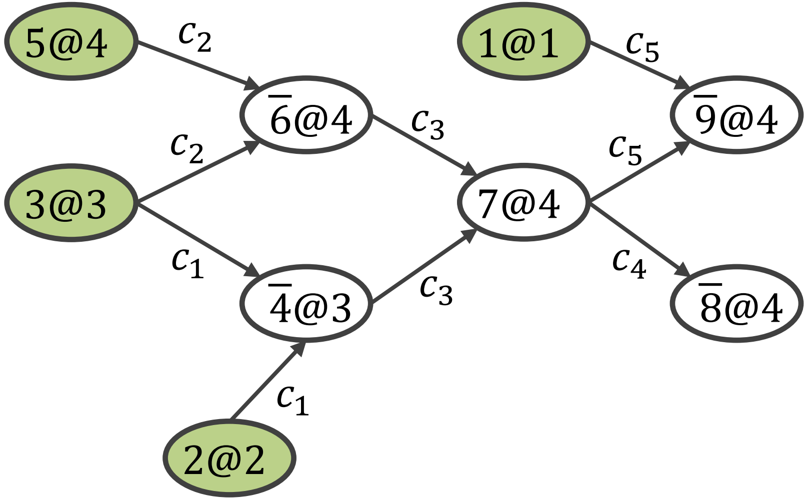
c_2: $\overline{3} \vee \overline{5} \vee \overline{6}$

c_3: $4 \vee 6 \vee 7$

c_4: $\overline{7} \vee \overline{8}$

c_5: $\overline{1} \vee \overline{7} \vee \overline{9}$

c_6: $\overline{1} \vee 8 \vee 9$



Implication Graphs

c_1: $\overline{2} \lor \overline{3} \lor \overline{4}$

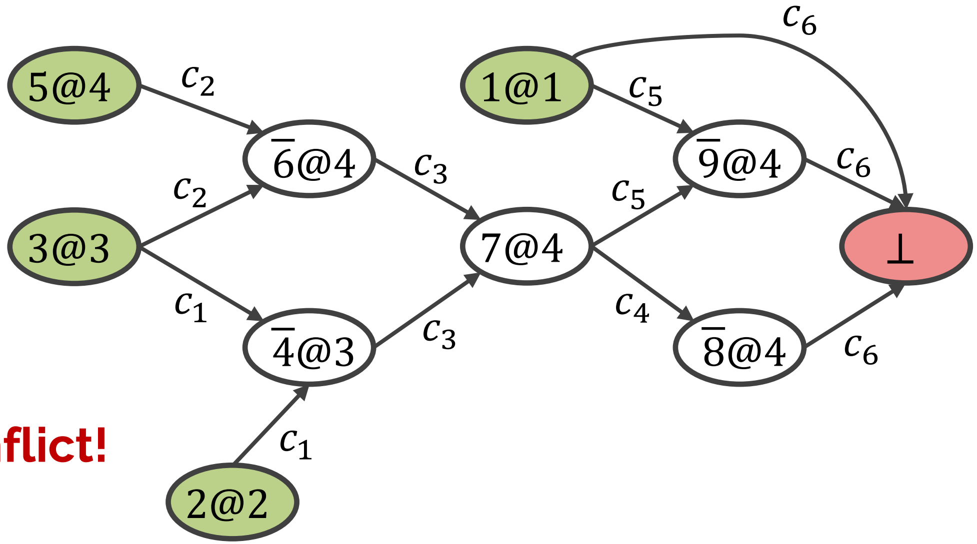
c_2: $\overline{3} \lor \overline{5} \lor \overline{6}$

c_3: $4 \lor 6 \lor 7$

c_4: $\overline{7} \lor \overline{8}$

c_5: $\overline{1} \lor \overline{7} \lor \overline{9}$

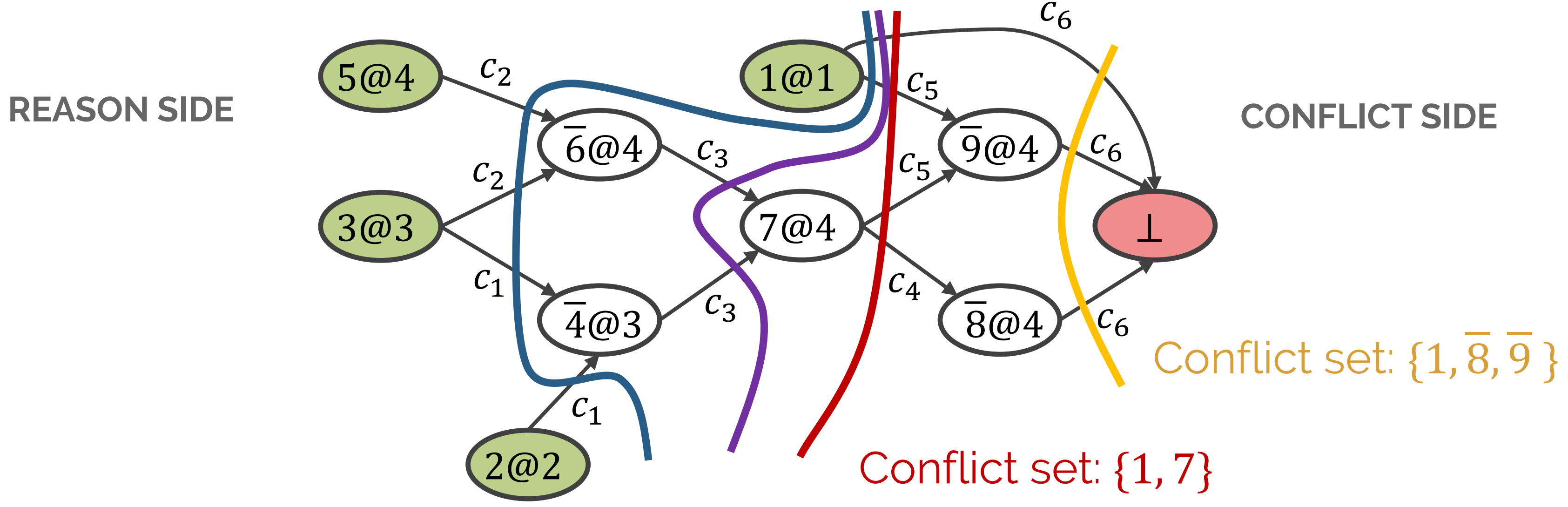
c_6: $\overline{1} \lor 8 \lor 9$ **Conflict!**



Conflicts

- A **conflict set** of assignments (collectively) imply a conflict
- A **conflict cut** in an implication graph is a bipartition of the vertices $V = R \cup C$ such that:
  - *Reason side* R contains all decisions (source nodes)
  - *Conflict side* C contains the conflict node (a sink)
  - No edges cross $C \to R$, only $R \to C$
- The set of vertices with an outgoing edge crossing a given conflict cut forms a conflict set

Ex: Conflict Cuts



Conflict set: $\{1, 2, 3, 5\}$ Conflict set: $\{1, \overline{4}, \overline{6}\}$

And more…

Clause Learning

- **Observation:** Given a conflict set $\{x_1, x_2, \ldots, x_k\}$, we know at least one literal in the set must be False
- Can derive the **conflict clause** $(\overline{x_1} \vee \overline{x_2} \vee \cdots \vee \overline{x_k})$
- **Conflict-driven clause learning (CDCL):** add conflict clauses to the original CNF we're solving
    - Introduced by GRASP (1996); revolutionized SAT solving
    - Many solvers have **aggressive deletion policy** for long, "inactive," "unhelpful" learned clauses – avoid explosion in CNF size

Asserting Clauses

- Many conflict cuts – how do we decide which to choose to build a conflict clause?
- **Goal:** after backjumping, be able to apply new knowledge from learned clause right away
  - Want learned clause to become a unit clause right after backjumping

Asserting Clauses

- A learned clause is **asserting** if it contains only one variable set on the same decision level as conflict
  - Is it possible for any conflict clause to contain zero?
- **Observation:** iff a clause is asserting, it will become a unit clause after backtracking
- How far can we backjump and still have asserting clauses become unit clauses?
  - Backjump to *second-largest* (i.e., deepest) decision level in asserting clause (or zeroth level if asserting clause has size 1)
    - i.e., return to that decision level (don't undo the decision)
  - Called the **asserting level**

CDCL (Pseudocode)

```
cdcl(φ):
    if unit_propagate() = CONFLICT: return UNSAT
    while not all variables have been set:
        let x = pick_variable()
        create new decision level; set x = T
        while unit_propagate() = CONFLICT:
            if level = 0: return UNSAT
            let (conflict_cls, assrt_lvl) = analyze_conflict()
            let φ = φ ∪ { conflict_cls }
            # discard all assignments after asserting level
            backjump(assrt_lvl)
    return SAT
```

Asserting Level Backjump

c_1: $\overline{2} \vee \overline{3} \vee \overline{4}$

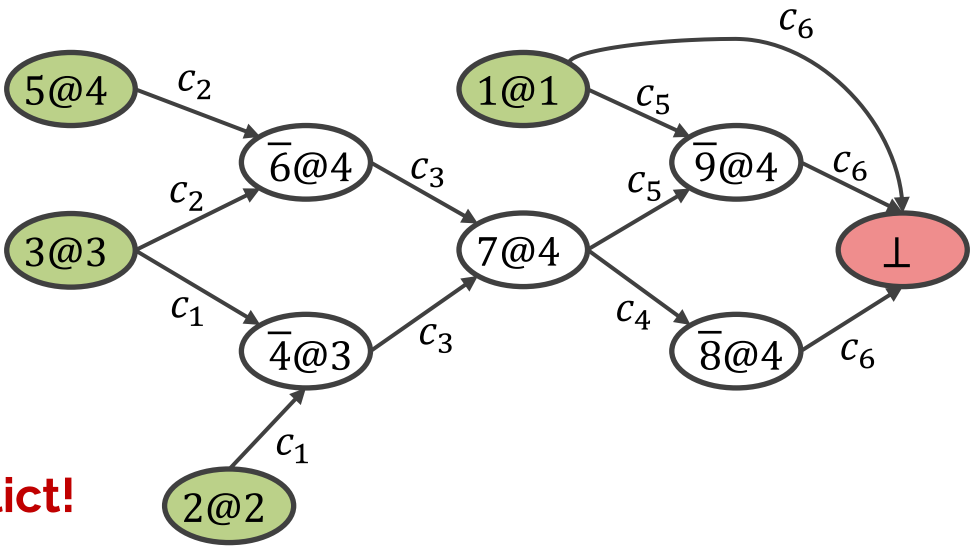
c_2: $\overline{3} \vee \overline{5} \vee \overline{6}$

c_3: $4 \vee 6 \vee 7$

c_4: $\overline{7} \vee \overline{8}$

c_5: $\overline{1} \vee \overline{7} \vee \overline{9}$

c_6: $\overline{1} \vee 8 \vee 9$ **Conflict!**



Asserting Level Backjump

c_1: $\overline{2} \vee \overline{3} \vee \overline{4}$

c_2: $\overline{3} \vee \overline{5} \vee \overline{6}$

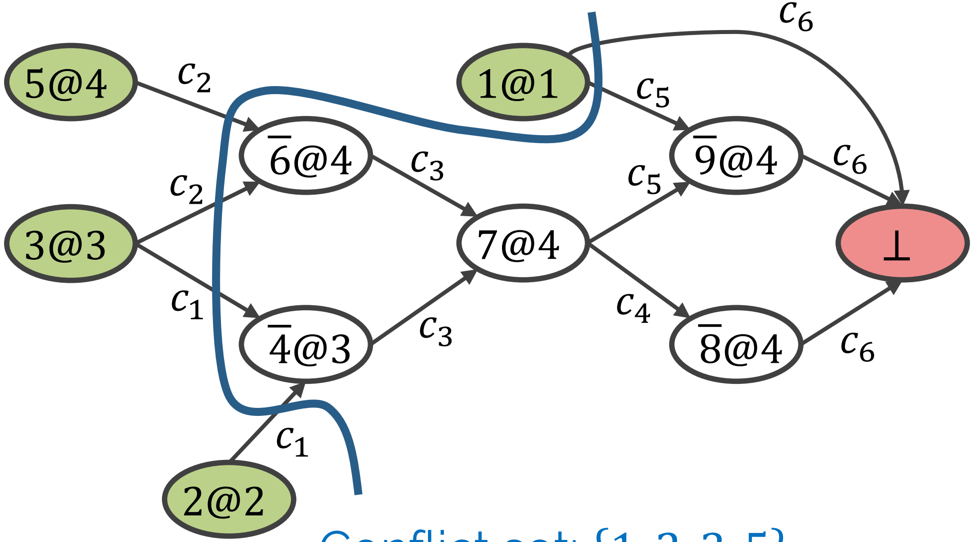
c_3: $4 \vee 6 \vee 7$

c_4: $\overline{7} \vee \overline{8}$

c_5: $\overline{1} \vee \overline{7} \vee \overline{9}$

c_6: $\overline{1} \vee 8 \vee 9$

c: $\overline{1} \vee \overline{2} \vee \overline{3} \vee \overline{5}$



Conflict set: $\{1, 2, 3, 5\}$
Asserting level = 3 (from 3@3)

Asserting Level Backjump

c_1: $\overline{2} \vee \overline{3} \vee \overline{4}$

c_2: $\overline{3} \vee \overline{5} \vee \overline{6}$

c_3: $4 \vee 6 \vee 7$

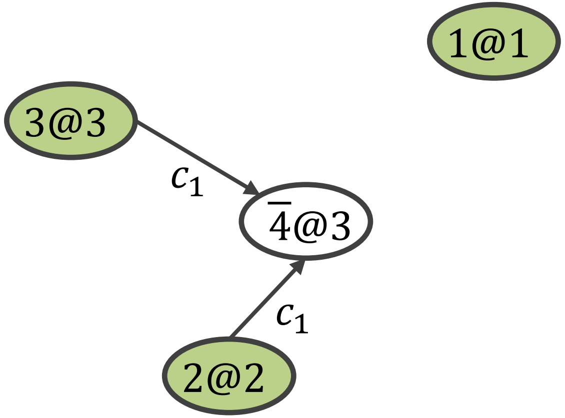
c_4: $\overline{7} \vee \overline{8}$

c_5: $\overline{1} \vee \overline{7} \vee \overline{9}$

c_6: $\overline{1} \vee 8 \vee 9$

c: $\overline{1} \vee \overline{2} \vee \overline{3} \vee \overline{5}$

Backjump!



Asserting Level Backjump

c_1: $\overline{2} \vee \overline{3} \vee \overline{4}$

c_2: $\overline{3} \vee \overline{5} \vee \overline{6}$

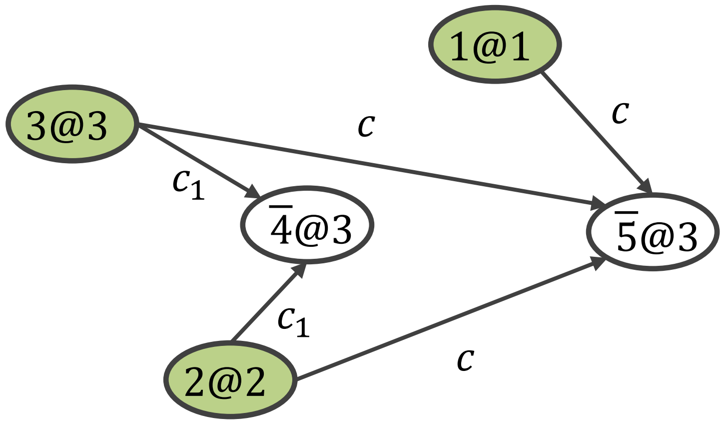
c_3: $4 \vee 6 \vee 7$

c_4: $\overline{7} \vee \overline{8}$

c_5: $\overline{1} \vee \overline{7} \vee \overline{9}$

c_6: $\overline{1} \vee 8 \vee 9$

c: $\overline{1} \vee \overline{2} \vee \overline{3} \vee \overline{5}$ **Unit!**



Asserting Level Backjump

c_1: $\overline{2} \lor \overline{3} \lor \overline{4}$

c_2: $\overline{3} \lor \overline{5} \lor \overline{6}$

c_3: $4 \lor 6 \lor 7$

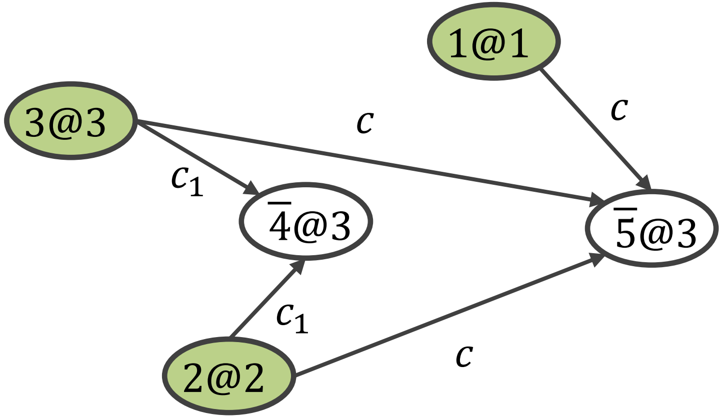
c_4: $\overline{7} \lor 8$

c_5: $\overline{1} \lor \overline{7} \lor \overline{9}$

c_6: $\overline{1} \lor 8 \lor 9$

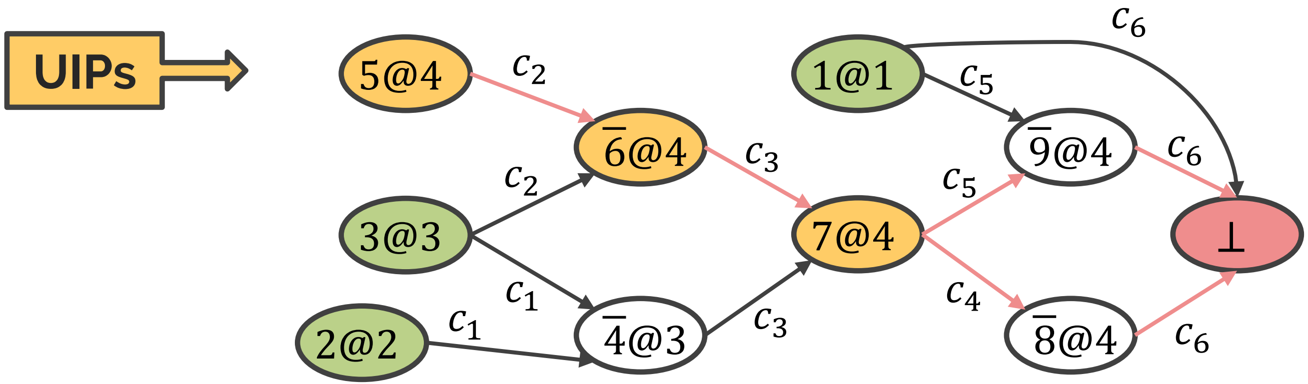
c: $\overline{1} \lor \overline{2} \lor \overline{3} \lor \overline{5}$ **Unit!**

Wait a second... same outcome as backtracking with DPLL.



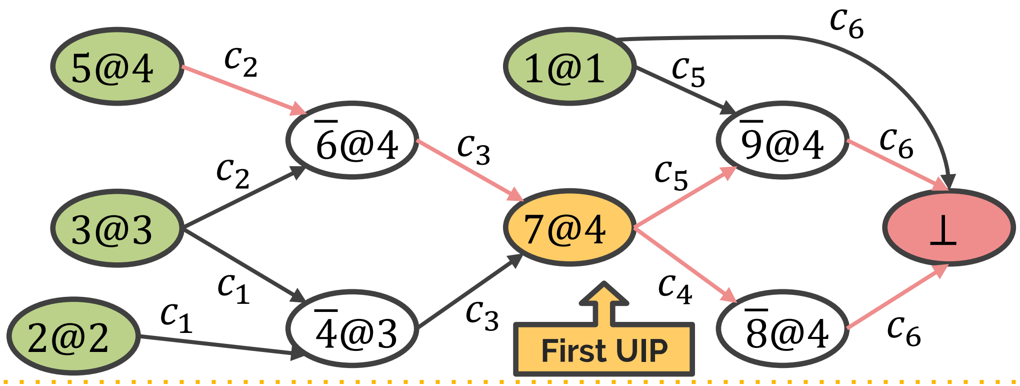
Unique Implication Points

- **Unique implication point (UIP):** a node in the implication graph that all paths from the most recent decision variable to the conflict must pass through
- **Intuition:** at the decision level of the conflict, the UIP is a literal that, by itself, implies a contradiction



The 1-UIP Scheme

- The **"first" UIP** is the closest UIP to the conflict node
  - i.e., the "rightmost"
- When we reach a conflict, cut after the first UIP
  - Generally produces shortest learned clauses



1-UIP Backjump

c_1: $\overline{2} \vee \overline{3} \vee \overline{4}$

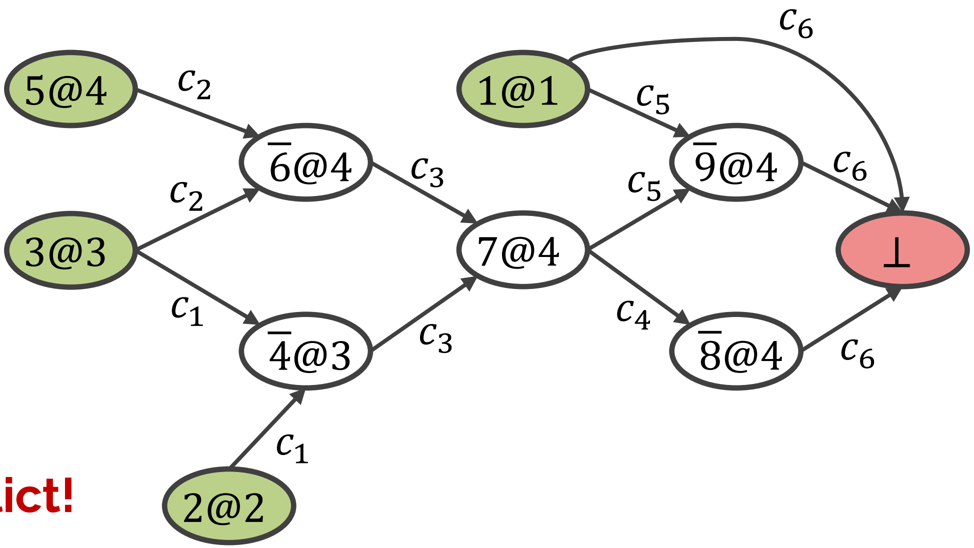
c_2: $\overline{3} \vee \overline{5} \vee \overline{6}$

c_3: $4 \vee 6 \vee 7$

c_4: $\overline{7} \vee \overline{8}$

c_5: $\overline{1} \vee \overline{7} \vee \overline{9}$

c_6: $\overline{1} \vee 8 \vee 9$ **Conflict!**



1-UIP Backjump

c_1: $\overline{2} \vee \overline{3} \vee \overline{4}$

c_2: $\overline{3} \vee \overline{5} \vee \overline{6}$

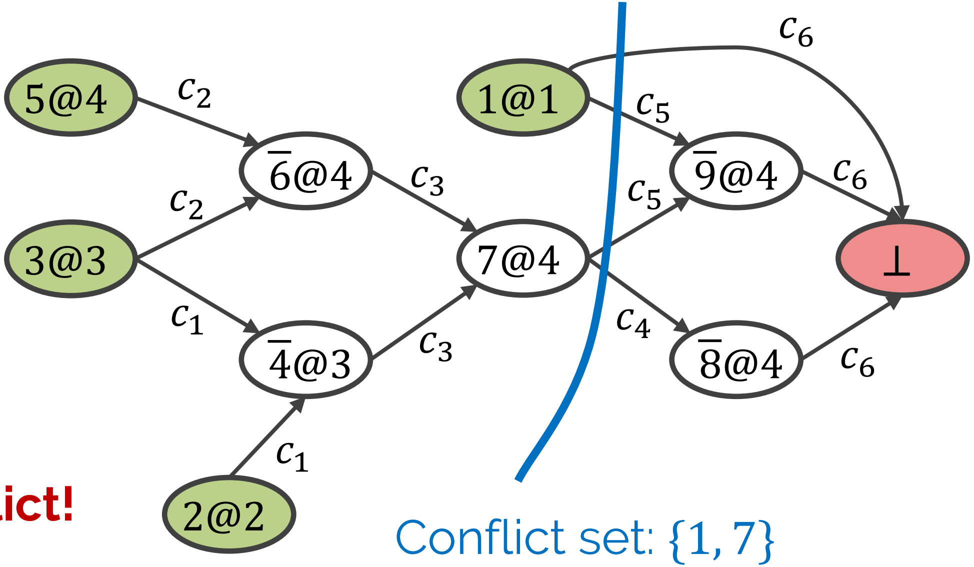
c_3: $4 \vee 6 \vee 7$

c_4: $\overline{7} \vee \overline{8}$

c_5: $\overline{1} \vee \overline{7} \vee \overline{9}$

c_6: $\overline{1} \vee 8 \vee 9$ **Conflict!**

c: $\overline{1} \vee \overline{7}$



Conflict set: $\{1, 7\}$

1-UIP Backjump

c_1: $\overline{2} \vee \overline{3} \vee \overline{4}$

c_2: $\overline{3} \vee \overline{5} \vee \overline{6}$

c_3: $4 \vee 6 \vee 7$

c_4: $\overline{7} \vee \overline{8}$

c_5: $\overline{1} \vee \overline{7} \vee \overline{9}$

c_6: $\overline{1} \vee 8 \vee 9$

c: $\overline{1} \vee \overline{7}$

Backjump!

1@1

1-UIP Backjump

c_1: $\overline{2} \vee \overline{3} \vee \overline{4}$

c_2: $\overline{3} \vee \overline{5} \vee \overline{6}$

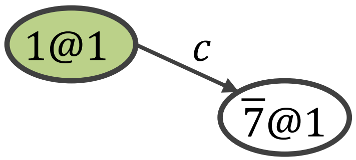
c_3: $4 \vee 6 \vee 7$

c_4: $\overline{7} \vee \overline{8}$

c_5: $\overline{1} \vee \overline{7} \vee \overline{9}$

c_6: $\overline{1} \vee 8 \vee 9$

c: $\overline{1} \vee \overline{7}$ **Unit!**



Restarts

- Problem: if we make bad early guesses, can get stuck in fruitless areas of search tree
- Solution: periodically **restart** the search – throw away the current partial assignment
    - Modern solvers favor aggressive restart policy
        - MiniSAT, PicoSAT: every ~100 conflicts
- **Key idea:** CDCL is deterministic, so why won't we end up back where we were?
    - Learned clauses remain in formula after restart

Incremental SAT Solving

- CDCL solvers give us a new method in our toolkit!

add_clause(C): add clause C to the formula

- New clauses can only rule out previously satisfying assignments
- Can re-solve CNF with new clauses added
- **Key:** keep learned clauses generated during last call to solve()
- Simple use case: generating all satisfying assignments

VSIDS Decision Heuristic

- **Variable State Independent Decaying Sum (VSIDS)**
  - For each literal x, maintain a score $s(x)$
  - Initially, let $s(x)$ be total number of occurrences of x
  - **Bump:** when x appears in a learned clause, increment $s(x)$
  - **Decay:** periodically, divide all scores by a constant
    - Typically 2
    - "Periodically": every k conflicts
  - Select literal with highest score
- Persists across restarts as well

VSIDS Decision Heuristic

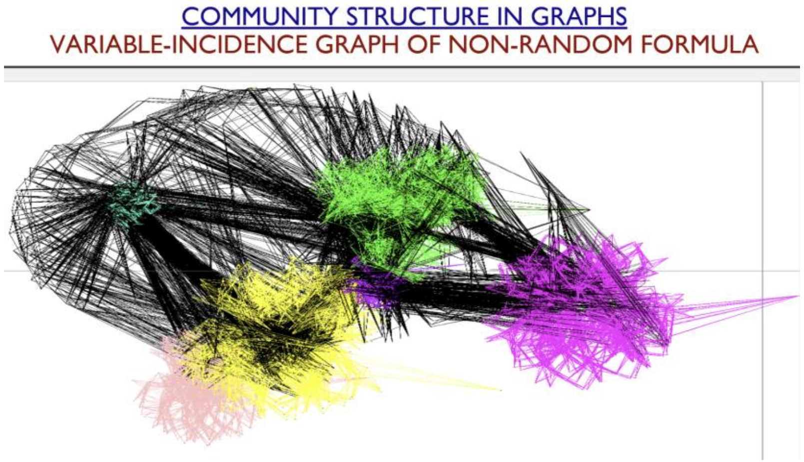
- **Intuition:** select literals that caused recent conflicts, and thus should be reassigned
- VSIDS is cheap: when we assign a variable, only need to update literals in learned clause
- Not only is it dynamic, it is **non-memoryless**: learns from the *path* to current state
    - All the dynamic heuristics we've seen only take into account current state

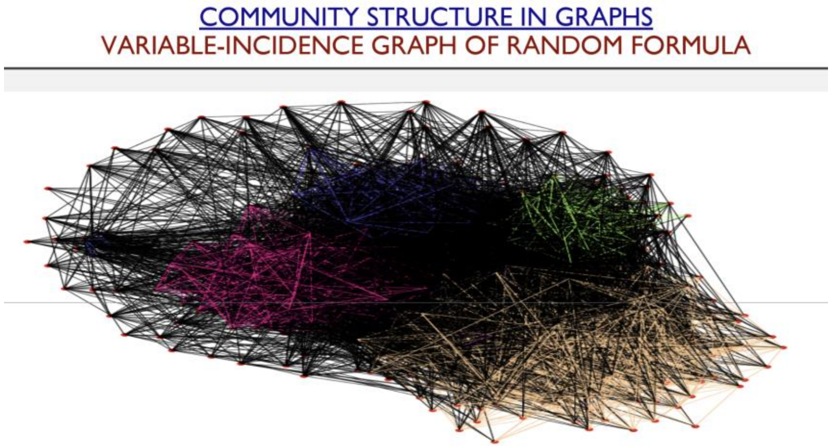
Let’s conclude with a brief tour of the latest research!

Success stories from the past 10 years

Why is CDCL so successful?

- CDCL gives massive improvement on large, industrial instances
- Attempts to explain success of CDCL by graph community structure
  - Nodes (variables) share an edge if they appear together in a clause



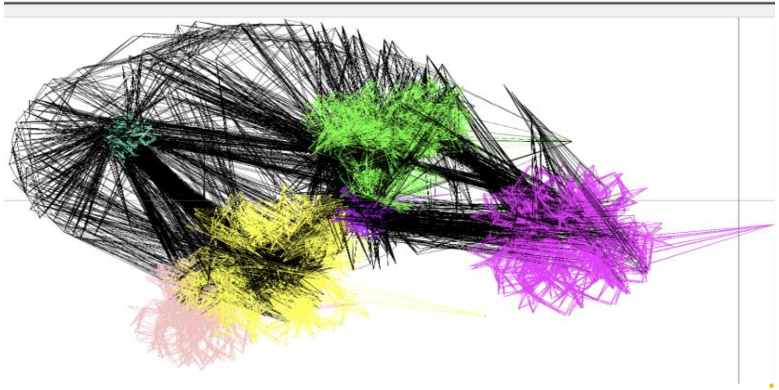


CDCL + Community Structure

- “Easier” to solve instances with highly isolated communities
- Empirically: learned clauses link few communities
- Empirically: VSIDS bumps “bridge variables” that link communities
  - When these variables are picked by VSIDS, breaks links between communities
- **Caution:** community structure doesn’t tell the whole story!

COMMUNITY STRUCTURE IN GRAPHS

VARIABLE-INCIDENCE GRAPH OF NON-RANDOM FORMULA



Recall: Lookahead Heuristics

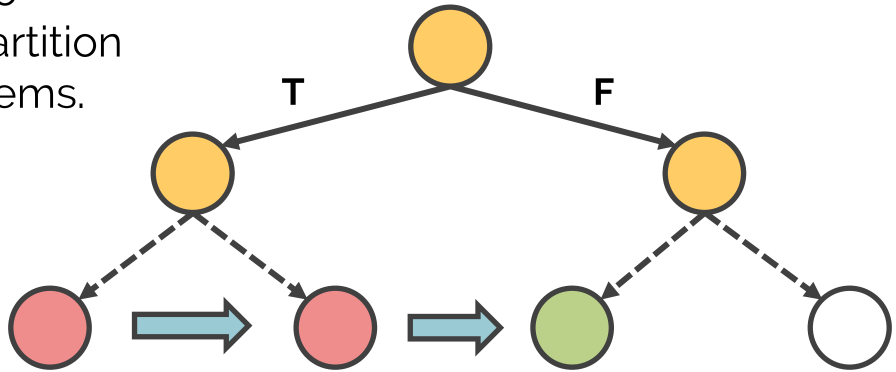
- **Lookahead heuristics:** for each unassigned variable x, try unit propagation from $x = T$ and $x = F$
  - Pick variable that reduces the formula most on both sides
- Highly global heuristic: good even for hard formulas
  - Good at making decisions that will partition the hard formulas into easier subproblems
- **Issue:** too expensive to be practical for large formulas

Cube and Conquer

- **Cube and conquer** (2012): combine lookaheads and CDCL
- Use lookaheads to guide initial search
- Partition formula into many easier subproblems
  - Many thousands (or millions) of subproblems
- Then solve those subproblems with CDCL…
  - …possibly in parallel!

Cube and Conquer

1. Use lookaheads to guide search and partition into many subproblems.



2. Once we are deep enough in the search tree, pass sub-problems off to CDCL solver. (Easily parallelizable!)

CDCL

4. If a subproblem is SAT, we are done.

3. If a subproblem is UNSAT, carry over relevant learned clauses and solve the next subproblem.

LRB Decision Heuristic

- **Learning rate branching** (2016): use ML/reinforcement learning techniques to pick next decision
- Give each variable a score that captures its likelihood of generating learned clauses over time
- **Multi-Armed Bandit** problem: maximize expected winnings from playing different slot machines, where the reward distribution of the machines is unknown and can change
- Solve MAB using RL – assigning a variable corresponds to playing on a slot machine

References

A. Biere, *Handbook of satisfiability*. Amsterdam: IOS Press, 2009.

E. Torlak, “A Modern SAT Solver,” *CSE507: Computer-Aided Reasoning for Software Engineering*. [Online]. Available: https://courses.cs.washington.edu/courses/cse507/.

V. Ganesh, *On the Unreasonable Effectiveness of Boolean SAT Solvers*. Saarbrücken, Germany: Max Planck Institute, 2017. Available: https://docs.google.com/a/gsd.uwaterloo.ca/viewer?a=v&pid=sites&srcid=Z3NkLnV3YXRlcmxvby5jYXxtYXBsZXNhdHxneDozYzQ3NDJjYjk4YWE4YTA0

Z. Newsham, V. Ganesh, S. Fischmeister, G. Audemard, and L. Simon. “Impact of Community Structure on SAT Solver Performance.” *Lecture Notes in Computer Science Theory and Applications of Satisfiability Testing – SAT 2014*, 2014, 252–68. https://doi.org/10.1007/978-3-319-09284-3_20.

J. Liang, V. Ganesh, E. Zulkoski, A. Zaman, and K. Czarnecki. “Understanding VSIDS Branching Heuristics in Conflict-Driven Clause-Learning SAT Solvers.” *Hardware and Software: Verification and Testing Lecture Notes in Computer Science*, 2015, 225–41. https://doi.org/10.1007/978-3-319-26287-1_14.

M. Heule, O. Kullmann, S. Wieringa, and A. Biere. “Cube and Conquer: Guiding CDCL SAT Solvers by Lookaheads.” *Hardware and Software: Verification and Testing Lecture Notes in Computer Science*, 2012, 50–65. https://doi.org/10.1007/978-3-642-34188-5_8.

J. Liang, V. Ganesh, P. Poupart, and K. Czarnecki. “Learning Rate Based Branching Heuristic for SAT Solvers.” *Theory and Applications of Satisfiability Testing – SAT 2016 Lecture Notes in Computer Science*, 2016, 123–40. https://doi.org/10.1007/978-3-319-40970-2_9.