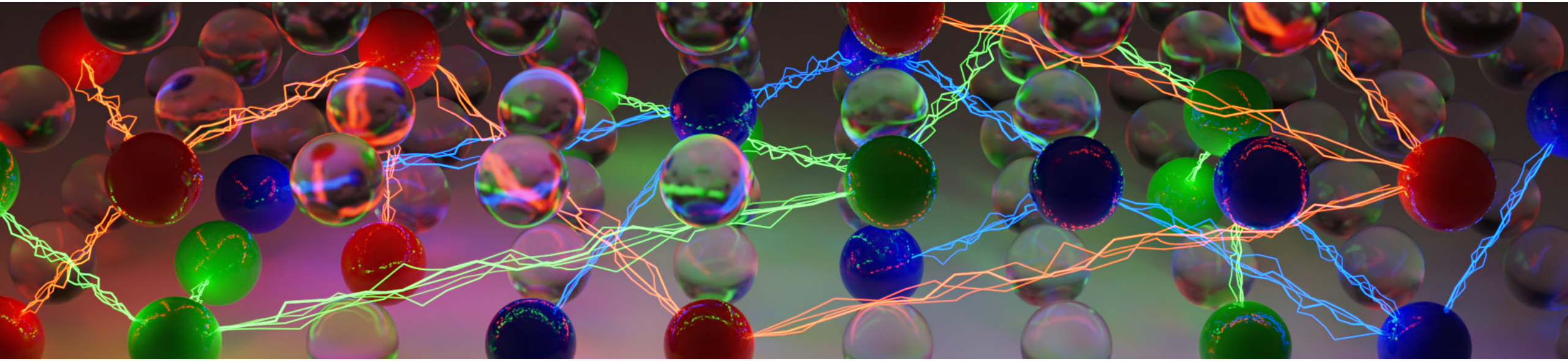
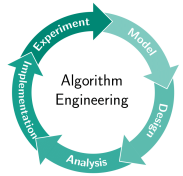


Practical SAT Solving



Lecture 6 – Elementary SAT Solving Heuristics, Conflict-Driven Clause Learning

Ashlin Iser, Dominik Schreiber | June 1, 2026

SAtRes
Scalable Automated Reasoning

Overview

Recap.

- Lectures 4 & 5: Applications of SAT Solving
- Lecture 3: Elementary SAT Solving Algorithms
  - Local Search
  - Resolution, Saturation, and DP Algorithm
  - DPLL Algorithm

Overview

Recap.

- Lectures 4 & 5: Applications of SAT Solving
- Lecture 3: Elementary SAT Solving Algorithms
  - Local Search
  - Resolution, Saturation, and DP Algorithm
  - DPLL Algorithm

Today's Topics: More on Algorithmic Methods

- DPLL: Elementary SAT Solving Heuristics
  - Branching Order
  - Branching Polarity
  - Restart Strategies
- Conflict-Driven Clause Learning (CDCL)
  - Implication Graphs
  - Conflict Analysis
  - Non-Chronological Backtracking

DPLL Algorithm: Iterative Variant

Decision Heuristics:

- Branching Order:

  Which variable to choose?

- Branching Polarity:

  Which value to assign?

Algorithm: iterativeDPLL(CNF Formula F)
Data: Trail (Stack of Literals)

```
1  while not all variables assigned by Trail do
2      if unitPropagation(F, Trail) has CONFLICT then
3          L ← last literal not tried both True and False
4          if no such L then return UNSAT
5          pop all literals after and including L from Trail
6          push {L = 0} on Trail
7      else
8          L ← pick an unassigned literal
9          push {L = 1} on Trail
10 return SAT
```

Properties of Decision Heuristics

Desired properties

- Fast to compute
- Gives easy sub-problems

Properties of Decision Heuristics

Desired properties

- Fast to compute
- Gives easy sub-problems
  - Satisfy many clauses
  - Maximize unit propagation

Properties of Decision Heuristics

Desired properties

- Fast to compute
- Gives easy sub-problems
  - Satisfy many clauses
  - Maximize unit propagation

Types of heuristics

- Static vs. Dynamic
  - Static: Based on formula statistics
  - Dynamic: Based on formula and current state
- Separate vs. Joint
  - Separate: Choose variable and value independently
  - Joint: Choose variable and value together

Decision Heuristics: Böhm's Heuristic

- $h_i(x)$: number of clauses of size i containing literal x which are not yet satisfied
- $H_i(x) := \alpha \max(h_i(x), h_i(\overline{x})) + \beta \min(h_i(x), h_i(\overline{x}))$ (let $\alpha := 1$ and $\beta := 2$, for example)
- Select literal x with the maximal vector $(H_1(x), H_2(x), \dots)$ under lexicographic order

Properties of Böhm's Heuristic

Goal: satisfy or reduce size of many and preferably short clauses

- Separate polarity heuristic (note that $H_i(x) = H_i(\overline{x})$)

  $\rightarrow$ select x if $\sum_i h_i(x) \geq \sum_i h_i(\overline{x})$
- depends on literal occurrence counts over the not yet satisfied clauses
- SAT Competition 1992: best heuristic for random instances

Decision Heuristics: Mom's Heuristic

Maximum Occurrences in clauses of Minimum Size

- $f^*(x)$: how often x occurs in the smallest not yet satisfied clauses
- Select variable x with a maximum $S(x) = \left(f^*(x) + f^*(\overline{x})\right) \cdot 2^k + f^*(x) \cdot f^*(\overline{x})$ (let $k := 10$, for example)

Properties of Mom's Heuristic

Goal: assign variables with high occurrence in short clauses

- Separate polarity heuristic

  $\rightarrow$ for example, select x if $f^*(\overline{x}) \geq f^*(x)$

- depends on literal occurrence counts over the not yet satisfied clauses
- Popular in the mid 90s (Find some variants in Freeman 1995, pages 39f.)

Decision Heuristics: Jeroslow-Wang Heuristic

- Choose the literal x with a maximum $J(x) = \sum_{x \in c,\ c \in F} 2^{-|c|}$

Properties of Jeroslow-Wang Heuristic

Goal: assign variables with high occurrence in short clauses

- Considers all clauses, but shorter clauses are more important
- Separate polarity heuristic
  $\rightarrow$ for example, use conflict-seeking polarity heuristic
- Two-sided variant: choose variable x with maximum $J(x) + J(\overline{x})$
  $\rightarrow$ one-sided version works better
- Much better experimental results than Böhm and MOMS

(R)DLCS and (R)DLIS Heuristics

(Randomized) Dynamic Largest (Combined | Individual) Sum

- based on positive $C_P(x)$ and negative occurences $C_N(x)$ of variable x
- used in the famous SAT solver GRASP in 2000

Properties of (R)DLCS and (R)DLIS Heuristics

- Dynamic: Take the current partial assignment into account
- Combined: select x with maximal $C_P(x) + C_N(x)$
- Individual: select x with maximal $\max(C_P(x), C_N(x))$
- Randomized: randomly select variable among the best

Recap

Decision Heuristics

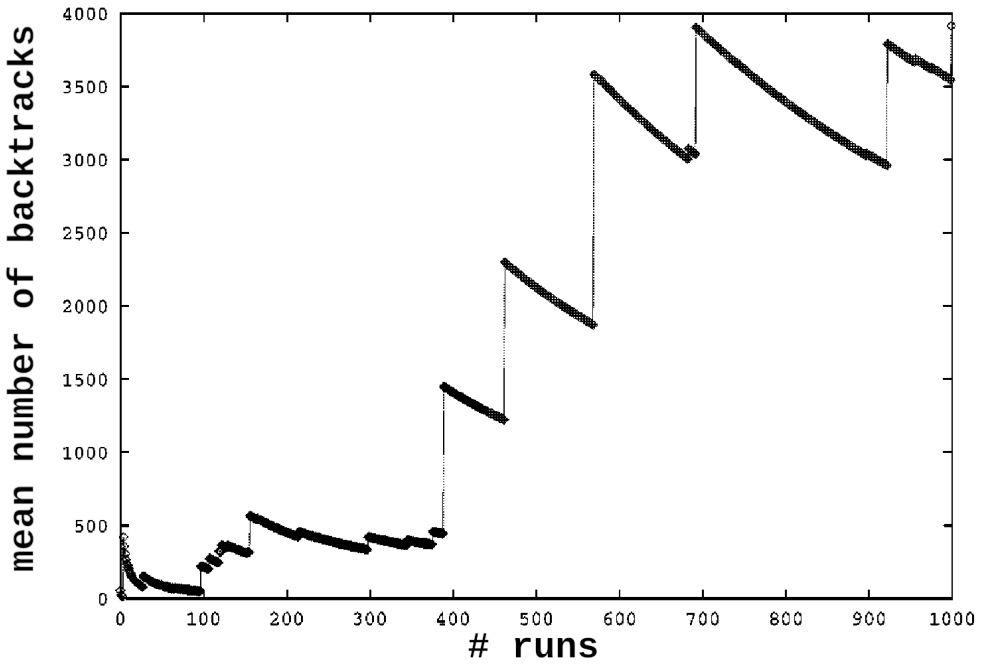
- Böhm's Heuristic
- Mom's Heuristic
- Jeroslow-Wang Heuristic
- (R)DLCS and (R)DLIS Heuristics

Next up

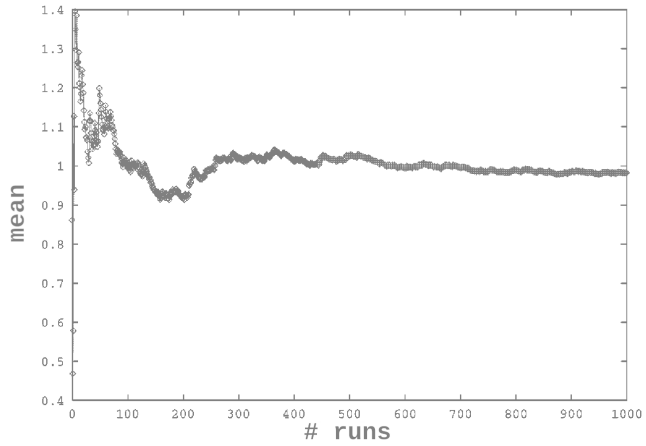
Restart Strategies

Restarts Strategies: Motivation

Given *n* runs of randomized DPLL search, what is the average number of backtracks per run?



Heavy-tailed distribution



For Reference:
Standard distribution

[Gomes et al. 2000]

Restart Strategies

Clear the partial assignment and backtrack to the root of the search tree.

- Recover from bad branching decisions
- Solve more instances on average
- Might decrease performance on easy instances

When to Restart?

- After some number of conflicts / backtracks
- The intervals between restarts should increase to guarantee completeness
  - Linear increase: too slow
  - Exponential increase: ok, with small exponent
  - MiniSat: k-th restart happens after 100×1.1^k conflicts

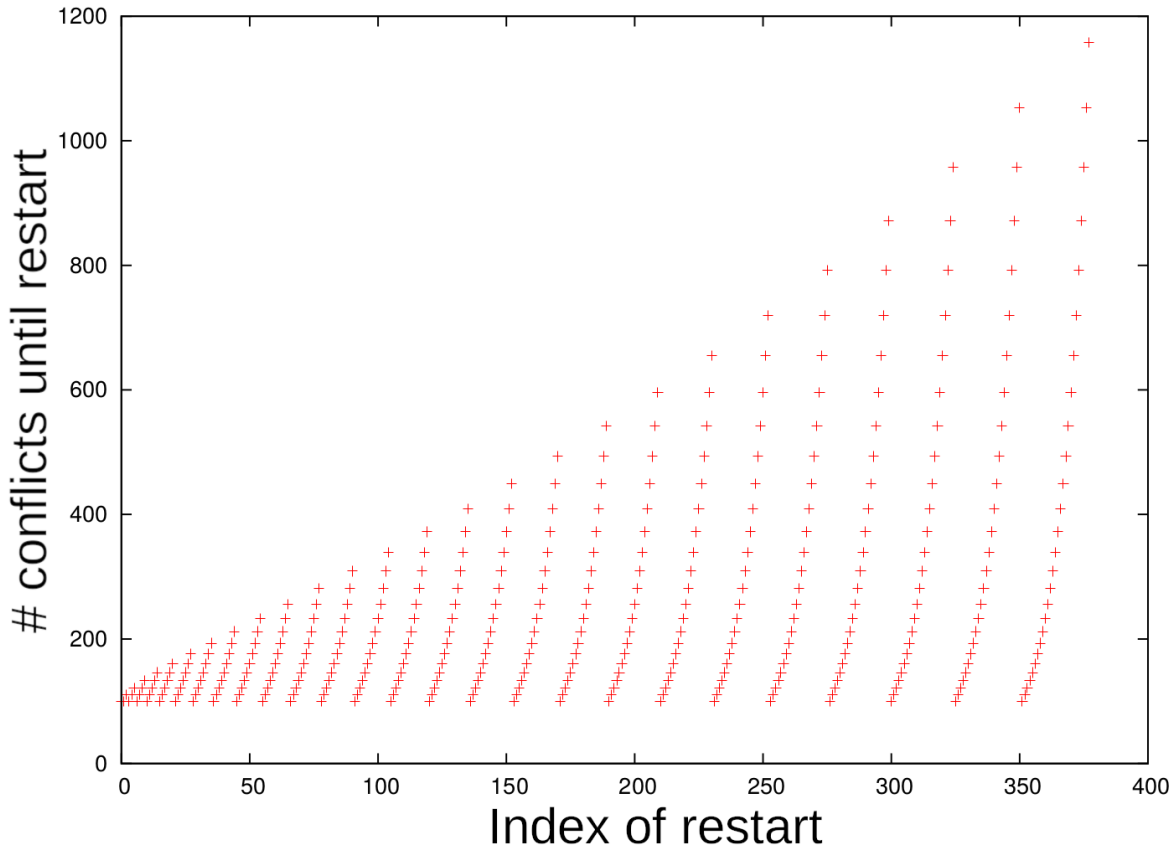
Restart Strategies: Inner / Outer Pattern (MiniSat)

Algorithm: Inner / Outer

Data: int inner = 100, outer = 100

```
1 while true do
2     run DPLL() with conflict-limit inner
3     restarts++
4     if inner ≥ outer then
5         outer *= 1.1
6         inner = 100
7     else
8         inner *= 1.1
```

Restart Strategies: Inner / Outer Pattern (MiniSat)



Algorithm: Inner / Outer

Data: int inner = 100, outer = 100

```
1 while true do
2     run DPLL() with conflict-limit inner
3     restarts++
4     if inner ≥ outer then
5         outer *= 1.1
6         inner = 100
7     else
8         inner *= 1.1
```

Restart Strategies: Luby Sequence

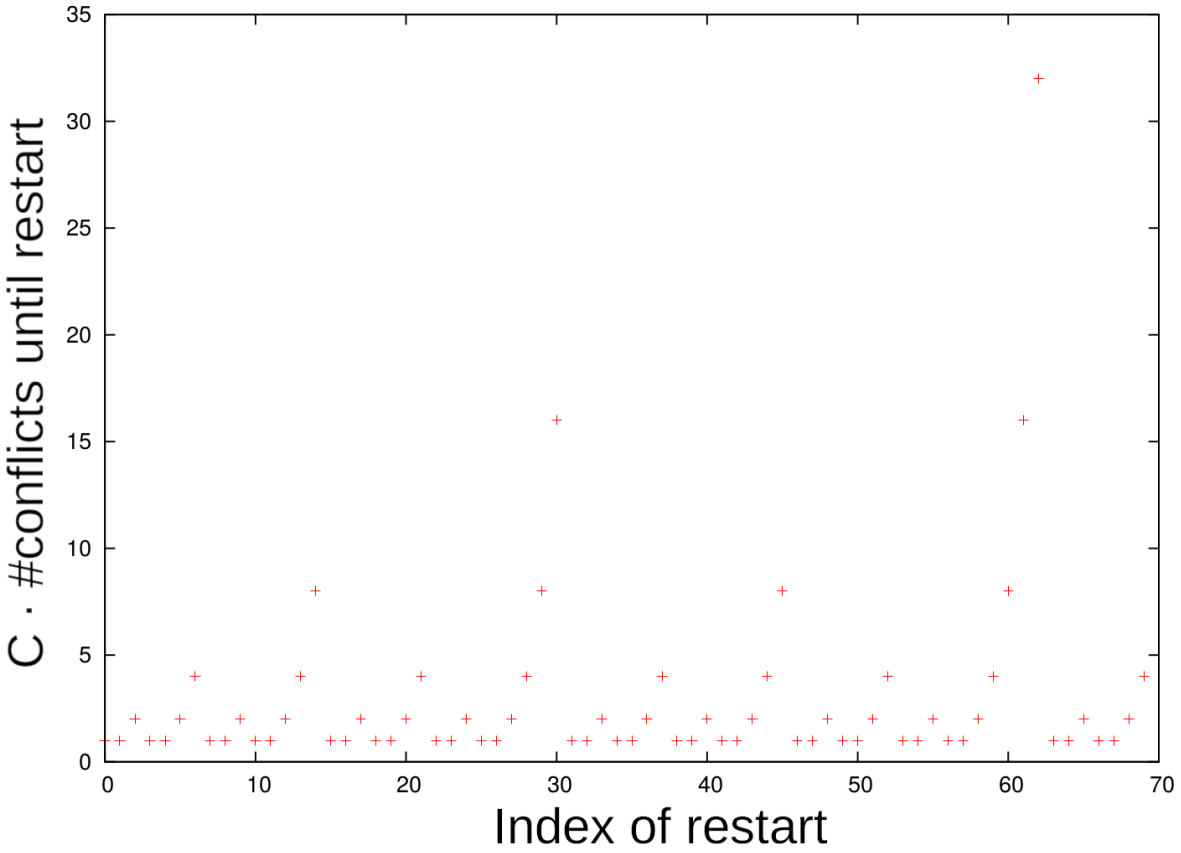
Theorem [Luby et al, 1993]

Consider a Las Vegas algorithm A (i.e., correct but with random run time) and a restart strategy $S = \langle t_1, t_2, \ldots \rangle$ (i.e., run A for time t_1, then for time t_2, etc.). Up to a constant factor, the Luby sequence is the best possible universal strategy to minimize the expected run time until a run is successful.

$$Luby = u \cdot (t_i)_{i \in \mathbb{N}} \quad \text{with} \quad t_i = \begin{cases} 2^{k-1} & \text{if } i = 2^k - 1 \\ t_{i-2^{k-1}+1} & \text{if } 2^{k-1} \leq i \leq 2^k - 1 \end{cases}$$

Example: $1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, \ldots$

Restart Strategies: Luby Sequence



Algorithm: Luby Sequence

Input: int $i \geq 1$ `// index of the` i`-th restart`

`/* determine smallest` k `with` $i \leq 2^k - 1$`,`
`i.e.` $k = \lceil \log_2(i+1) \rceil$ `*/`

$k \leftarrow 1$
while $i > (1 \ll k) - 1$ **do**
  $k \leftarrow k + 1$

`// emit peak`
if $i == (1 \ll k) - 1$ **then**
  return $1 \ll (k-1)$

`// recurse into first half`
return Luby$(i - (1 \ll (k-1)) + 1)$

Example usage in SAT solvers: Run DPLL() with conflict-limit 512· Luby(++restarts)

Restart Strategies: Luby Sequence

Luby Sequence: Reluctant Doubling

A more efficient implementation of the Luby sequence invented by Donald Knuth.

Use the v_n of the following pairs (u_n, v_n):

$$\begin{aligned}(u_1, v_1) &:= (1, 1)\\ (u_{n+1}, v_{n+1}) &:= \big(u_n \,\&\, (-u_n)\big) = v_n \;\; ? \;\; (u_n{+}1,\, 1) \;\; : \;\; (u_n,\, 2 \cdot v_n)\end{aligned}$$

Example: $(1, \mathbf{1}), (2, \mathbf{1}), (2, \mathbf{2}), (3, \mathbf{1}), (4, \mathbf{1}), (4, \mathbf{2}), (4, \mathbf{4}), (5, \mathbf{1}), \ldots$

Intuition: u_n counts which "run" we are in; v_n doubles inside that run until it reaches the lowest set bit of u_n, at which point Luby's recursive structure demands a reset. The $u \,\&\, (-u)$ part is a one-instruction way to isolate that bit.

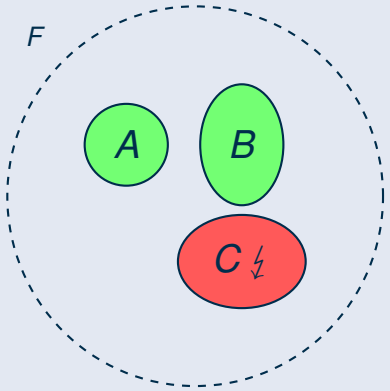
Branching Polarity: Phase Saving

Observation: Frequent restarts can destroy useful work on satisfiable instances — after a restart, the solver may pick the opposite polarity and re-explore parts of the search space it had already shown to be consistent.

Phase Saving / Assignment Caching [RSAT, 2006]

Idea: Remember the last assigned value of each variable and reuse it the next time the variable is branched on (after a restart or backtrack).

- Preserves consistent assignments across restarts
- Stabilizes the positive effect of frequent restarts
- Especially effective combined with non-chronological backtracking (next topic): the solver can jump back over a satisfied component and later restore it from cache



A, *B*, *C*: clusters of variables;
F: the whole formula.
green = satisfied component,
red = current conflict.
Phase cache restores *A*, *B* without re-search.

Recap

Decision Heuristics

Böhm, MOMS, Jeroslow-Wang, (R)DLCS / (R)DLIS

Restart Strategies

- Inner / Outer Pattern
- Luby Sequence / Reluctant Doubling
- Phase Saving / Assignment Caching

Next up

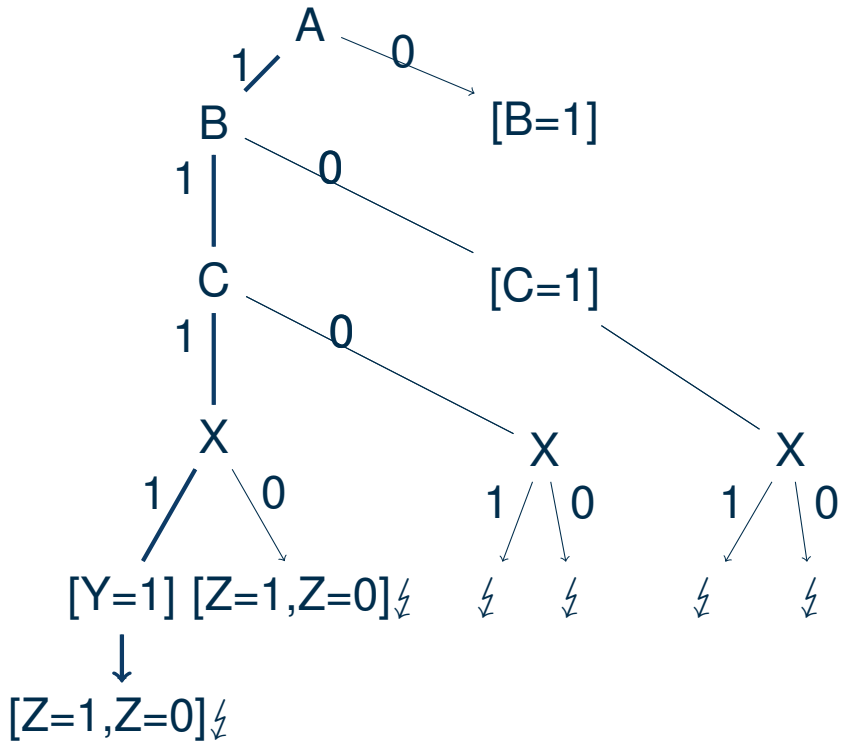
Clause Learning

DPLL: Chronological Backtracking

$\{\{A, B\}, \{B, C\}, \{\neg A, \neg X, Y\},$
$\{\neg A, X, Z\}, \{\neg A, X, \neg Z\},$
$\{\neg A, \neg Y, Z\}, \{\neg A, \neg Y, \neg Z\}\}$ (Formula)

A, B, C, X, Y, Z (Trail)

$\{\neg A, \neg Y, \neg Z\}$ (Conflicting Clause)

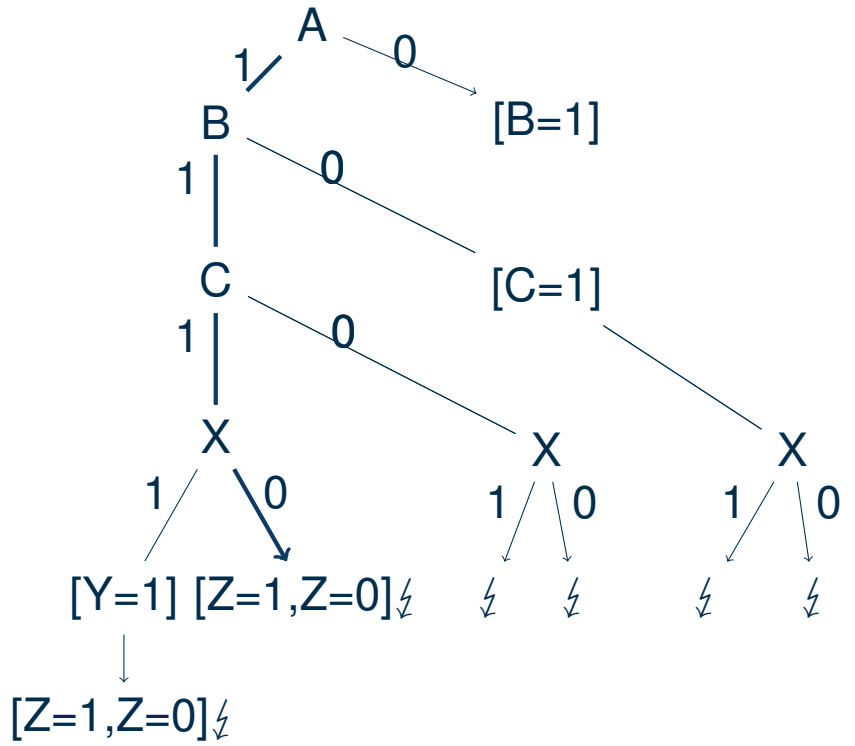


DPLL: Chronological Backtracking

$\{\{A, B\}, \{B, C\}, \{\neg A, \neg X, Y\},$
$\{\neg A, X, Z\}, \{\neg A, X, \neg Z\},$
$\{\neg A, \neg Y, Z\}, \{\neg A, \neg Y, \neg Z\}\}$ (Formula)

$A, B, C, \neg X, Z$ (Trail)

$\{\neg A, X, \neg Z\}$ (Conflicting Clause)

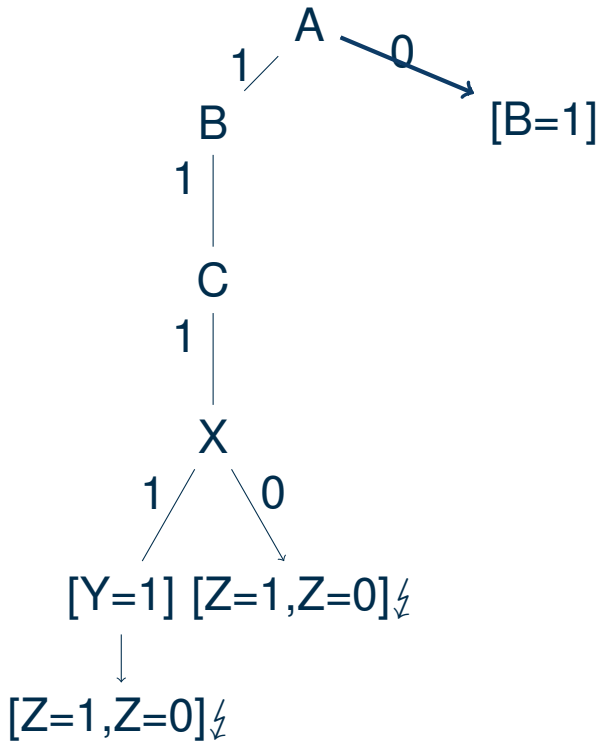


DPLL: Chronological Backtracking

$$\{\{A, B\}, \{B, C\}, \{\neg A, \neg X, Y\}, \{\neg A, X, Z\}, \{\neg A, X, \neg Z\}, \{\neg A, \neg Y, Z\}, \{\neg A, \neg Y, \neg Z\}\}$$ (Formula)

Observation: Conflicting clauses $\{\neg A, \neg Y, \neg Z\}$, $\{\neg A, X, \neg Z\}$ constrain only a fraction of the trail (B and C irrelevant)

How to find out which assignments on the trail are relevant for the actual conflict and immediately backtrack to A?



Implication Graph

Idea: Make the causal structure of unit propagation explicit, so we can read off *why* a conflict occurred and *which earlier decisions are to blame*.

Definition: Implication Graph

Given a formula F, assignment trail T, and conflicting clause C,
the implication graph is a DAG $G = (V \cup \{\lightning\}, E)$ where

- each vertex $[\ell_i, d_i]$ corresponds to a literal ℓ_i on the trail together with its decision level d_i
- the special vertex $\lightning$ represents the conflict;

  all literals of C have edges to $\lightning$
- there is an edge $\big([\ell_i, d_i], [u_{i,j}, d_i]\big)$ whenever $u_{i,j}$ was propagated by a clause containing $\neg\ell_i$

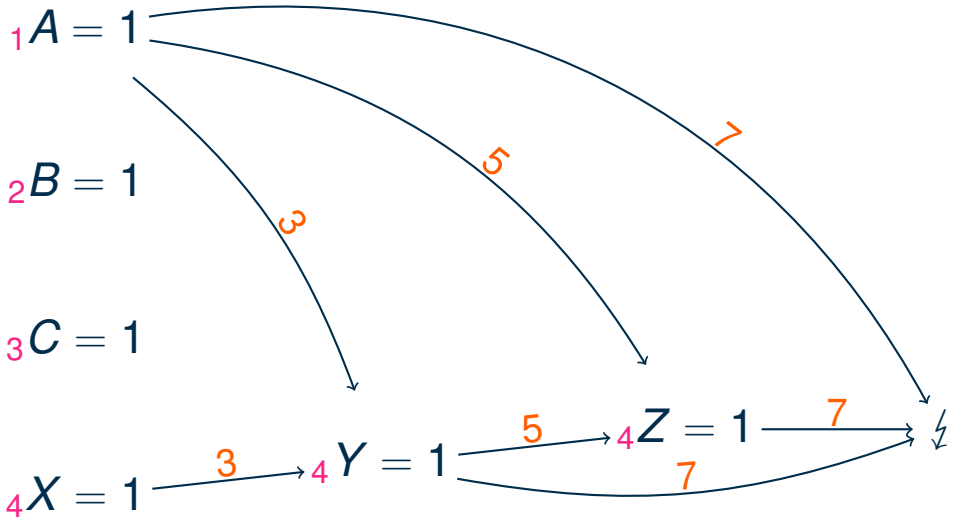
  (i.e., ℓ_i is one of the reasons for $u_{i,j}$)

Example: Implication Graph

Implication graph for the conflicting state under the trail A, B, C, X, Y, Z.
Node labels show a variable assignment with its decision level; edge labels denote the propagating clause.

Reading the graph:

- Source A is the only decision literal causing the conflict (B, C are irrelevant).
- Sink $\lightning$ is the conflict; its in-edges come from the conflicting clause $\{\neg A, \neg Y, \neg Z\}$ (clause 7).
- Any cut separating A from $\lightning$ yields a candidate learned clause.



$$
\begin{aligned}
&\{\{A, B\}, && (1)\\
&\ \{B, C\}, && (2)\\
&\ \{\neg A, \neg X, Y\}, && (3)\\
&\ \{\neg A, X, Z\}, && (4)\\
&\ \{\neg A, \neg Y, Z\}, && (5)\\
&\ \{\neg A, X, \neg Z\}, && (6)\\
&\ \{\neg A, \neg Y, \neg Z\}\} && (7)
\end{aligned}
$$

Example cut $\{\neg A, \neg X\}$: derive by resolution $(7 \circ_Z 5) \circ_Y 3$;
Learning it forbids the same partial assignment from being revisited.

Conflict Analysis: Implementation

Implement the trail as a stack of literals, each paired with a pointer to its reason clause (*null* for decisions) and its decision level. On a conflict, walk the trail top-down and resolve the conflicting clause against the reason clauses of the involved literals until only one literal of the current decision level remains.

Example: Conflict on clause $\{\neg A, \neg Y, \neg Z\}$ at decision level 4.

Var.	Lvl.	Reason
$\lightning$	4	$\{\neg A, \neg Y, \neg Z\}$
Z	4	$\{\neg A, \neg Y, Z\}$
Y	4	$\{\neg A, \neg X, Y\}$
X	4	*null*
C	3	*null*
B	2	*null*
A	1	*null*

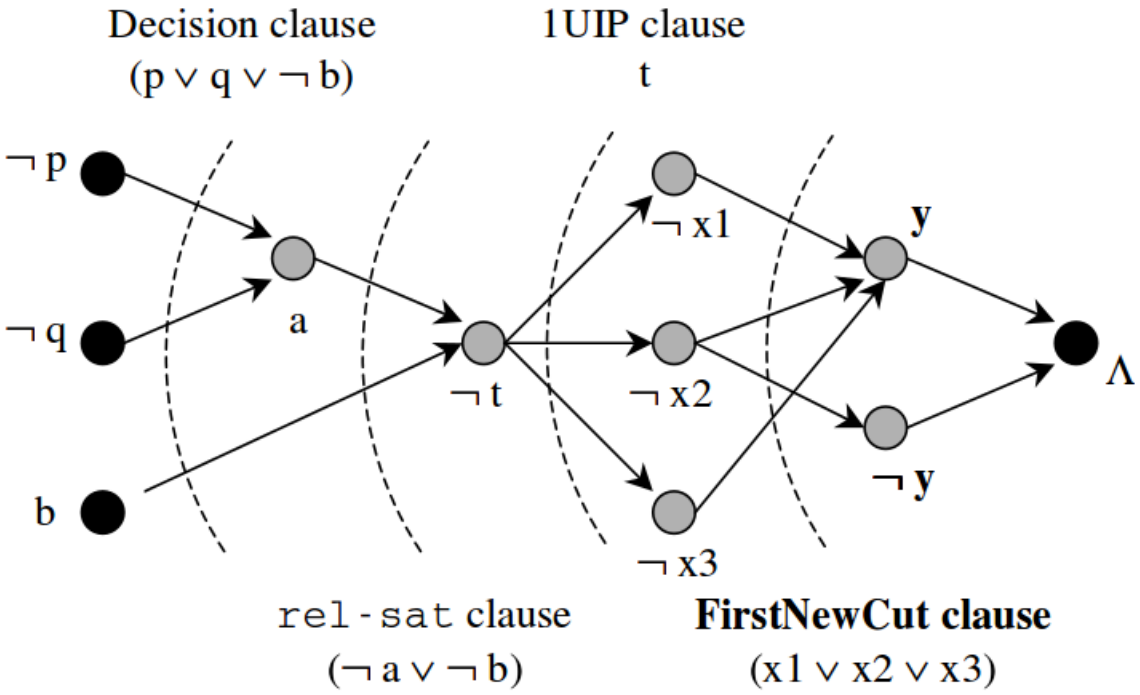
Trail Resolution:

- Start from the conflicting clause:
  $\{\neg A, \neg Y, \neg Z\}$
- Resolve on Z with reason of Z:
  $\{\neg A, \neg Y, \neg Z\} \otimes_Z \{\neg A, \neg Y, Z\} = \{\neg A, \neg Y\}$
- Resolve on Y with reason of Y:
  $\{\neg A, \neg Y\} \otimes_Y \{\neg A, \neg X, Y\} = \{\neg A, \neg X\}$
- Stop: only one literal ($\neg X$) of level 4 remains.
  Learned clause: $C = \{\neg A, \neg X\}$

Conflict Analysis: Unit Implication Points (UIP)

Several possibilities to learn a clause from an implication graph exist.

- UIP is a dominator in the implication graph (restricted to variables assigned at the current decision level)
- A node v is a dominator for $\lightning$, if all paths to $\lightning$ contain v
- FirstUIP: “first” dominator (seen from conflict side)



[Beame et al, 2003]

Conflict Analysis: 1-UIP Learning

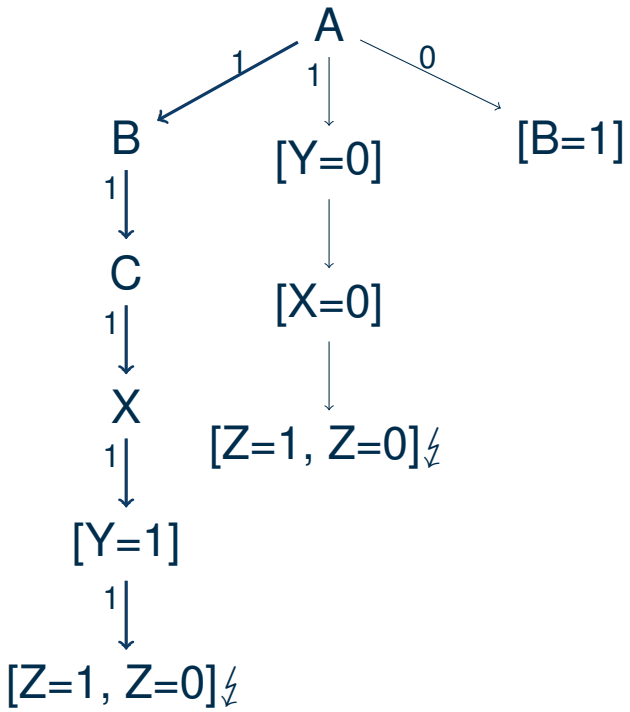
Resolve the conflicting clause and reason clauses until only a single literal of the current decision level remains.

Advantage:

- Stopping at a UIP always leads to an asserting clause.
- A clause is asserting if all literals are false except one, which is unassigned.
- Algorithm becomes simpler: backtrack until clause becomes asserting.
- The backtrack level is always the second highest level in a conflict clause.

Non-chronological Backtracking

1-UIP learning changes the decision tree in our example like this:

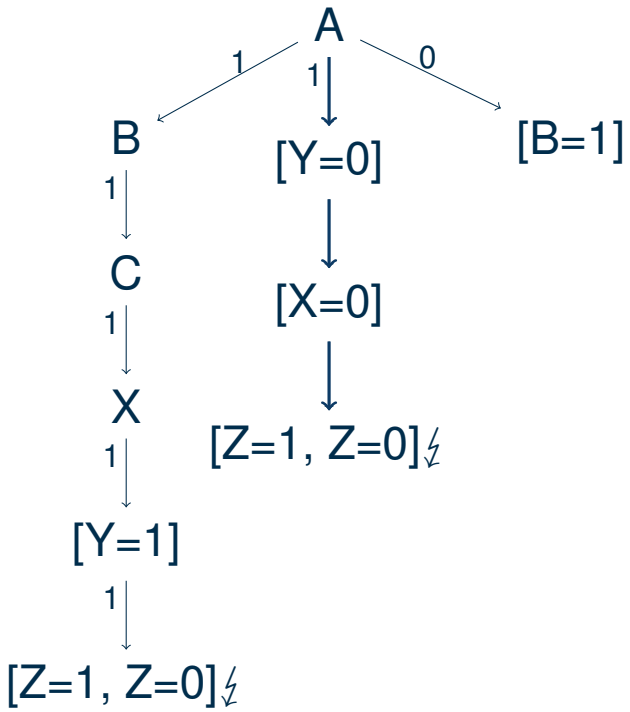


$$F = \{\{A, B\}, \{B, C\}, \{\neg A, \neg X, Y\}, \{\neg A, X, Z\}, \{\neg A, \neg Y, Z\}, \{\neg A, X, \neg Z\}, \{\neg A, \neg Y, \neg Z\}\}$$

Trail: A, B, C, X, Y, Z
Conflicting Clause: $\{\neg A, \neg Y, \neg Z\}$
Conflict Clause (1UIP): $\{\neg A, \neg Y\}$

Non-chronological Backtracking

1-UIP learning changes the decision tree in our example like this:



$$F = \{\{A, B\}, \{B, C\}, \{\neg A, \neg X, Y\}, \{\neg A, X, Z\}, \{\neg A, \neg Y, Z\}, \{\neg A, X, \neg Z\}, \{\neg A, \neg Y, \neg Z\} \{\neg A, \neg Y\}\}$$

Trail: $A, \neg Y, \neg X, Z$

Conflicting Clause: $\{\neg A, X, \neg Z\}$

Clause Learning: Further Reading

Beyond 1-UIP: apply the UIP idea on every decision level (all-UIP) so that each level contributes at most one literal to the learned clause. This can produce much shorter clauses. But, unlike 1-UIP, it may also introduce new literals on lower levels, so the clause could actually grow.

- Feng & Bacchus, SAT 2020 – Clause Size Reduction with all-UIP Learning

  First good background overview of learning variants, then introduces all-UIP shrinking. Naively too expensive and prone to enlarging the clause, so they guard it with several heuristics.

- Fleury & Biere, SAT 2021 – Efficient All-UIP Learned Clause Minimization

  Reformulate all-UIP shrinking. Their algorithm is linear in the size of the implication graph – cheap enough to run unconditionally to completion, no heuristic gating needed.

From Clause Learning to Resolution Proofs

Key observation: Each learned clause C is obtained by a chain of resolutions on reason clauses (cf. trail resolution). So CDCL is, at heart, a resolution-proof generator.

Properties of every learned clause C

- Sound: $F \models C$ (resolution is sound)
- Propagation-derivable: $F \cup \neg C \vdash_{UP} \bot$ ($\neg C$ triggers a conflict by unit propagation alone)
- Non-redundant: no $D \subsetneq C$ is already in F

Certificates of Unsatisfiability

- If CDCL derives the empty clause $\bot$, the sequence of learned clauses is a resolution refutation of F.
- Emitted as a DRAT / LRAT proof and independently checked (e.g. `drat-trim`, verified checkers in Coq/Isabelle).
- Standard in SAT competitions and high-assurance applications (verification, certified mathematics, etc.).

The End.

Recap

- Decision Heuristics
  - Böhm's Heuristic
  - Mom's Heuristic
  - Jeroslow-Wang Heuristic
  - (R)DLCS and (R)DLIS Heuristics
- Restart Strategies
  - Inner / Outer Pattern
  - Luby Sequence / Reluctant Doubling
  - Branching Heuristic: Phase Saving
- Conflict Analysis, Clause Learning, and Non-chronological Backtracking
- Resolution Proofs