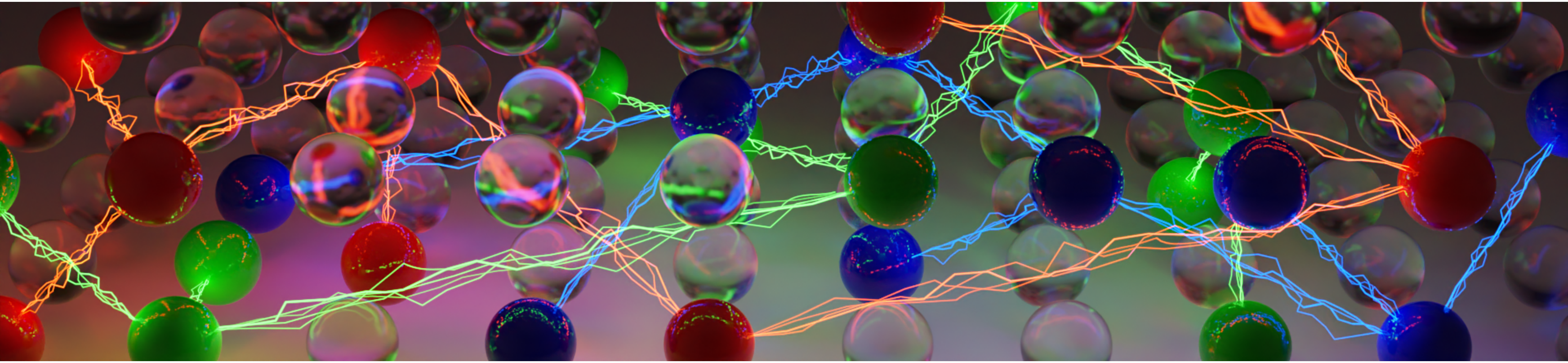


Practical SAT Solving

Lecture 3 – Elementary SAT Solving Heuristics, Conflict-Driven Clause Learning

Markus Iser, Dominik Schreiber | May 12, 2025

Overview

Recap. Lecture 3: Elementary SAT Solving Algorithms

- Local Search
- Resolution
- DP Algorithm
- DPLL Algorithm

Overview

Recap. Lecture 3: Elementary SAT Solving Algorithms

- Local Search
- Resolution
- DP Algorithm
- DPLL Algorithm

Today’s Topics

- Elementary SAT Solving Heuristics
  - Branching Order
  - Branching Polarity
  - Restart Strategies
- Conflict-Driven Clause Learning

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)

while *not all variables assigned by Trail* **do**
    if *unitPropagation(F, Trail) has CONFLICT* **then**
        $L \leftarrow$ last literal not tried both True and False
        if *no such L* **then return** UNSAT
        pop all literals after and including L from Trail
        push $\{L = 0\}$ on Trail
    else
        $L \leftarrow$ pick an unassigned literal
        push $\{L = 1\}$ on Trail
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

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

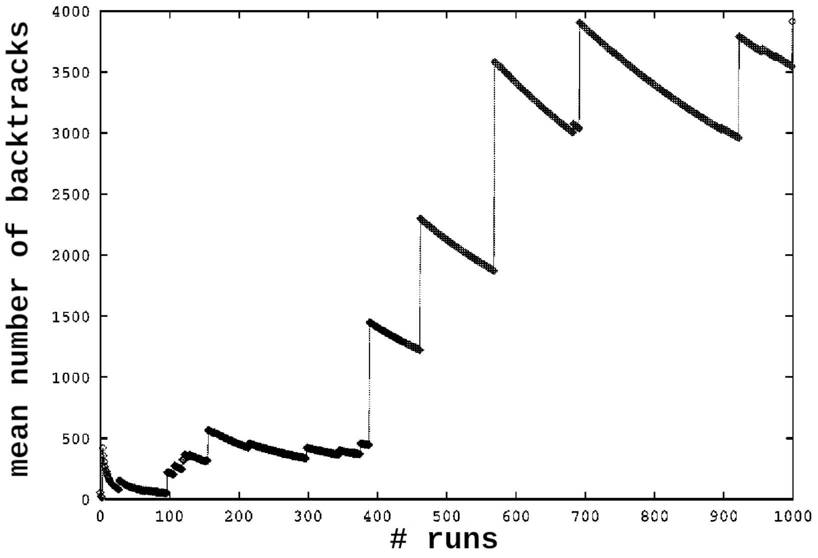
- Böhm’s Heuristic
- Mom’s Heuristic
- Jeroslow-Wang Heuristic
- (R)DLCS and (R)DLIS Heuristics

Next up

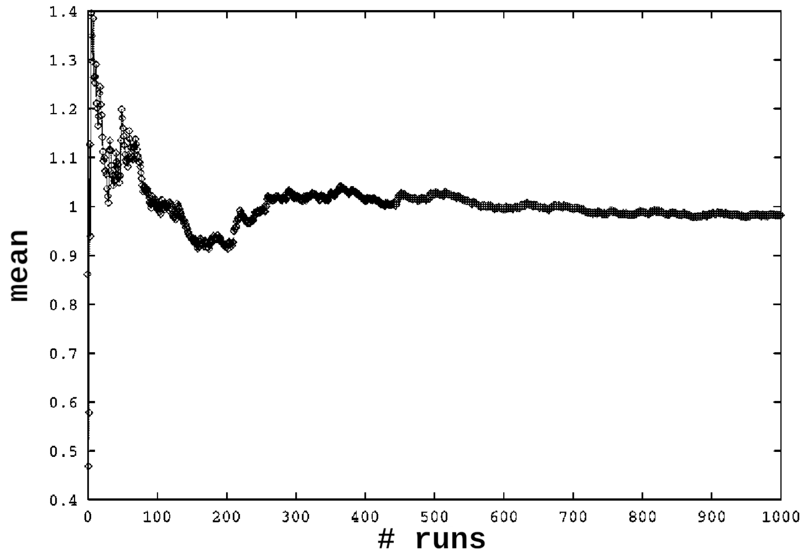
Restart Strategies

Restarts Strategies: Motivation

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



Heavy-tailed distribution

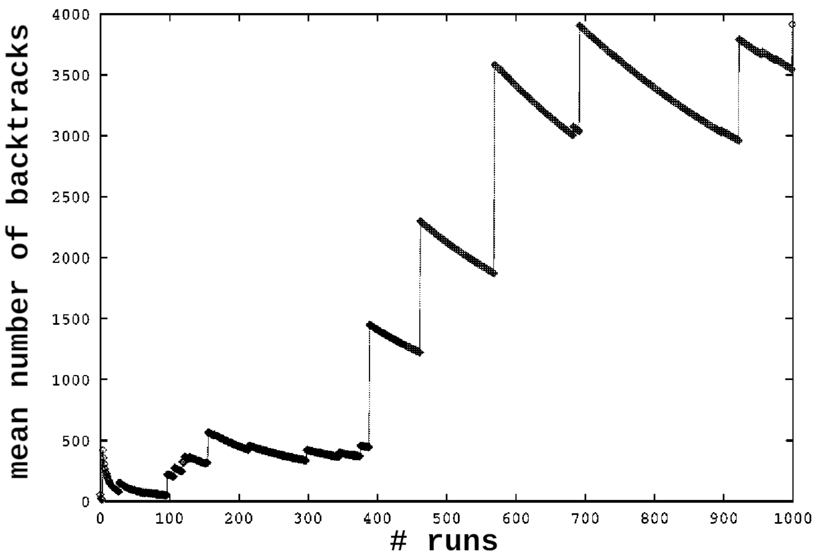


Standard distribution

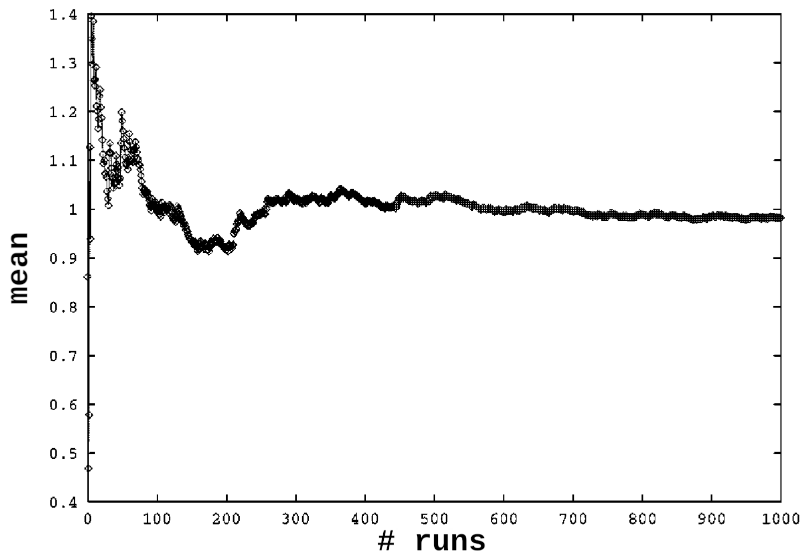
[Gomes et al. 2000]

Restarts Strategies: Motivation

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



Heavy-tailed distribution



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

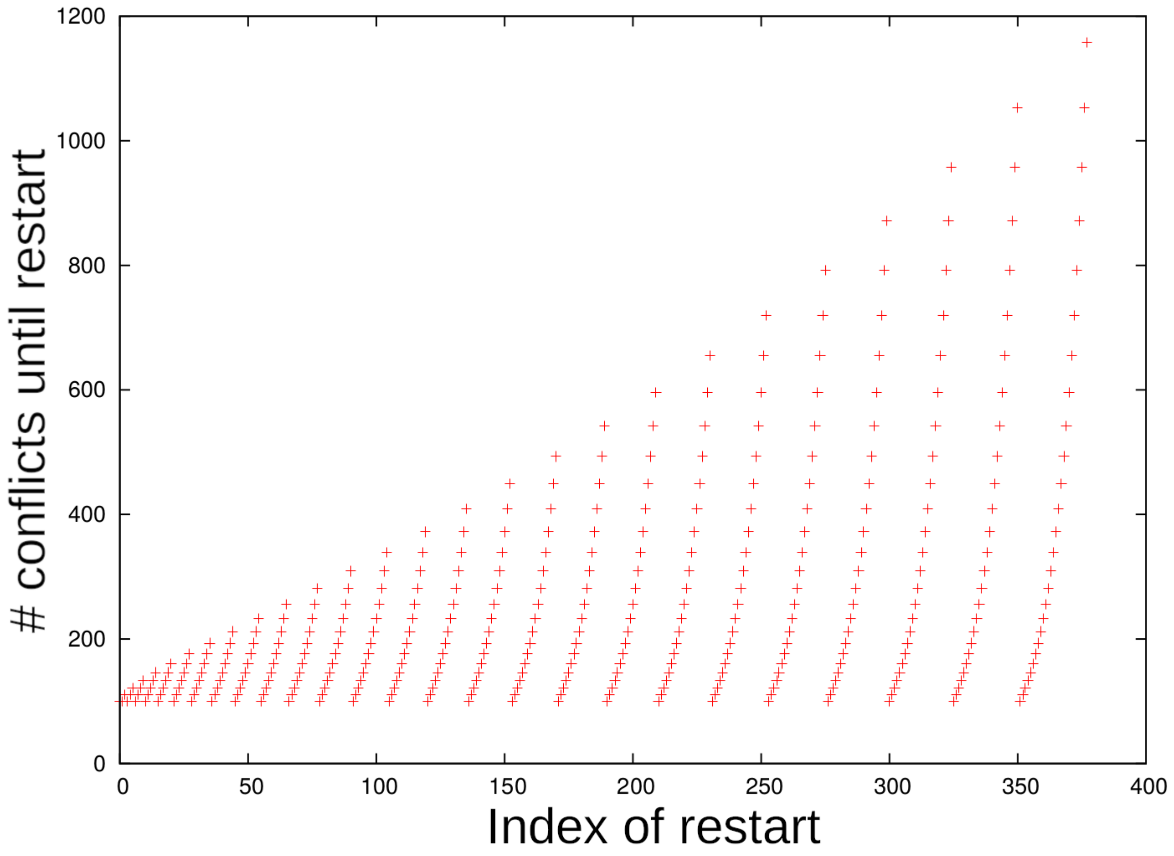
Restart Strategies: Inner / Outer Pattern (MiniSat)

Algorithm: Inner / Outer

Data: int inner = 100, outer = 100

```
while true do
    run DPLL() with conflict-limit inner
    restarts++
    if inner ≥ outer then
        outer *= 1.1
        inner = 100
    else
        inner *= 1.1
```

Restart Strategies: Inner / Outer Pattern (MiniSat)



Algorithm: Inner / Outer

Data: int inner = 100, outer = 100

while *true* **do**
  run DPLL() with conflict-limit *inner*
  restarts++
  if *inner* $\geq$ *outer* **then**
    outer *= 1.1
    inner = 100
  else
    inner *= 1.1

Restart Strategies: Luby Sequence

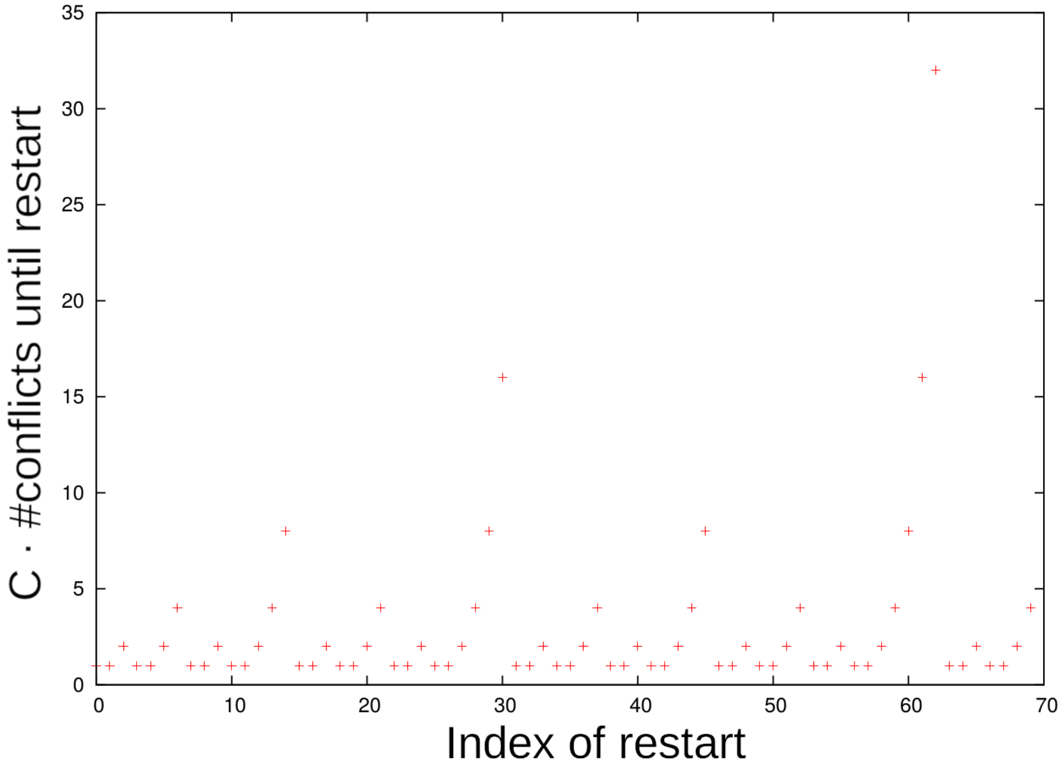
Theorem [Luby et al, 1993]

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

for $k = 1$ **to** 32 **do**
  if $i == (1 \ll k) - 1$ **then**
    return $1 \ll (k - 1)$

for $k = 1$ **to** ∞ **do**
  if $(1 \ll (k - 1)) \leq i \leq (1 \ll k) - 1$ **then**
    return Luby($i - (1 \ll (k - 1)) + 1$)

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):

$(u_1,\ v_1)\ =\ (1,\ 1);$
$(u_{n+1},\ v_{n+1})\ =\ u_n\ \&\ \text{-}u_n\ ==\ v_n\ ?\ (u_n+1,\ 1)\ :\ (u_n,\ 2v_n);$

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

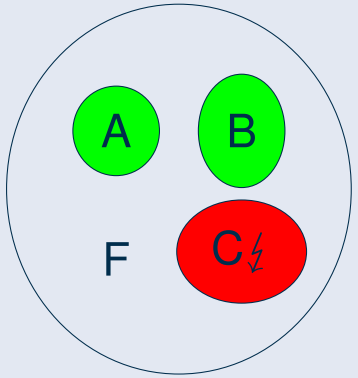
Branching Polarity: Phase Saving

Observation: Frequent restarts decrease performance on some satisfiable instances

Phase Saving / Assignment Caching

Idea: Remember last assignment of each variable and use it first in branching

- First implemented in RSAT (2006)
- Result: Phase saving stabilizes positive effect of restarts
- Best results in combination with non-chronological backtracking (follows)



Example: A and B are satisfied, searching in component C

Recap

Decision Heuristics

Restart Strategies

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

Next up

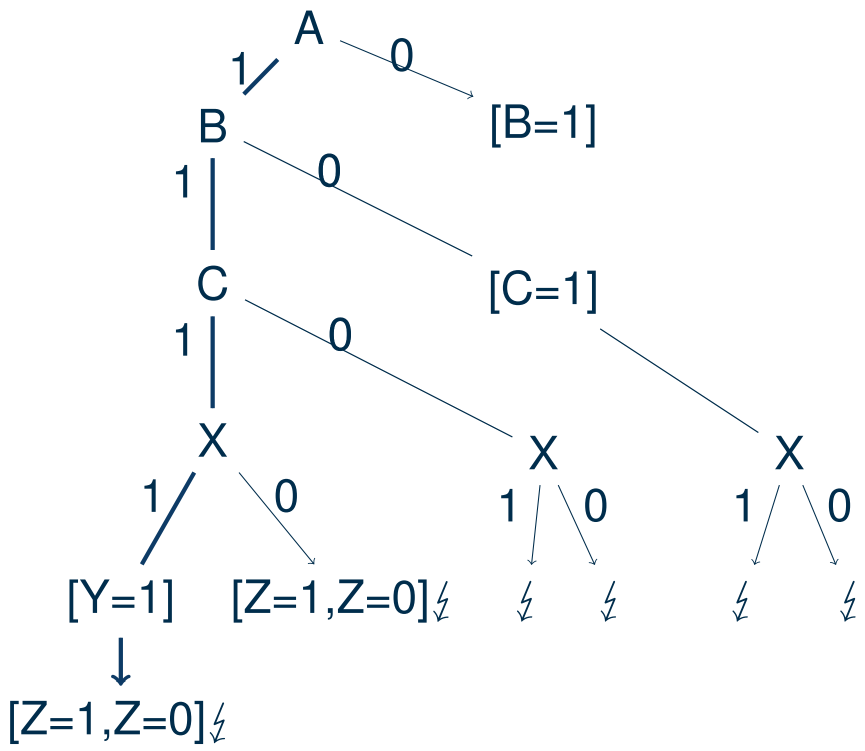
Clause Learning

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)

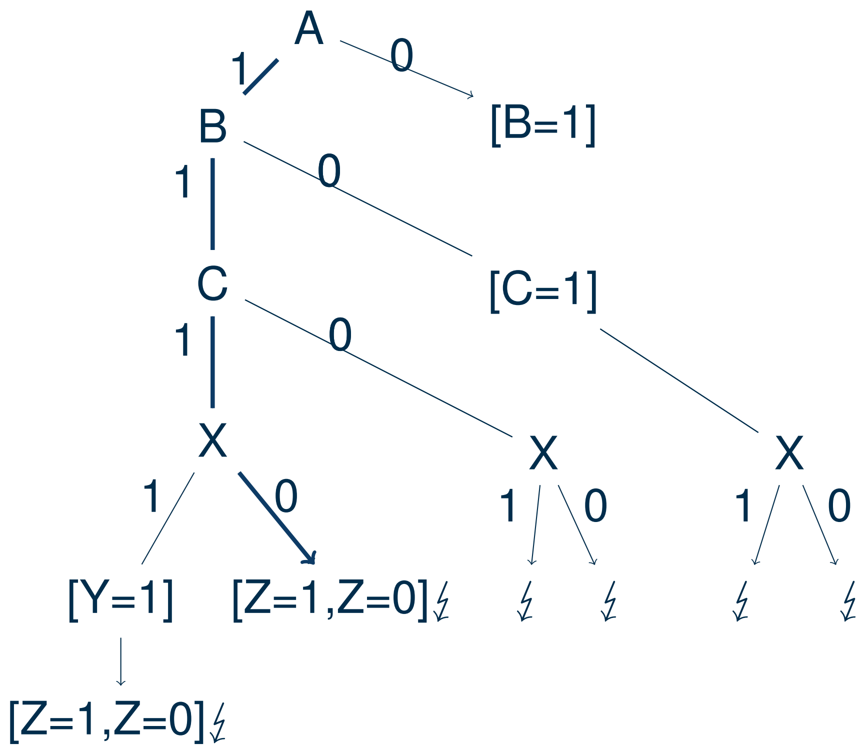


DPLL: Chronological Backtracking

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

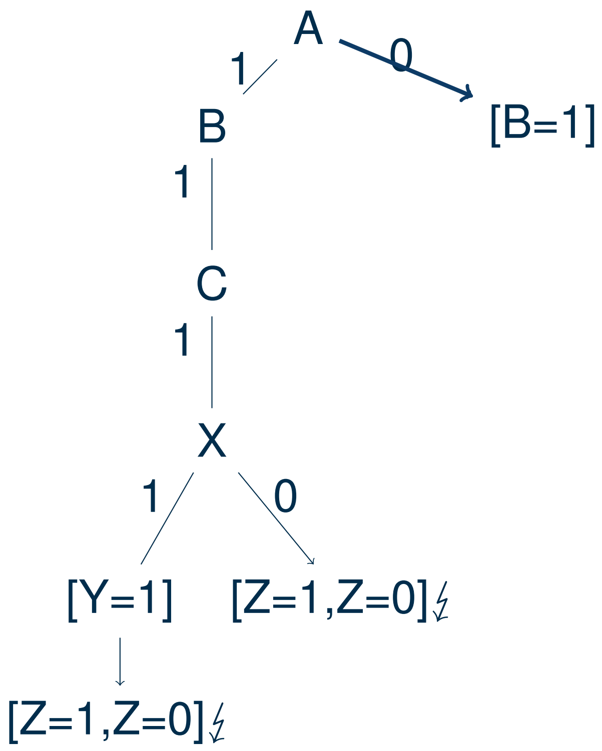
$$A, B, C, \neg X, Z \qquad \text{(Trail)}$$

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



DPLL: Chronological Backtracking

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



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

Definition: Implication Graph

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

- vertices $[\ell_i, d_i]$ for each literal ℓ_i with decision level d_i on the trail
- vertex $\lightning$ representing the conflicting assignment

  Note: all literals of C have edges to $\lightning$
- edges $([\ell_i, d_i], [u_{i,j}, d_i])$ for each propagated literal $u_{i,j}$ at decision level d_i

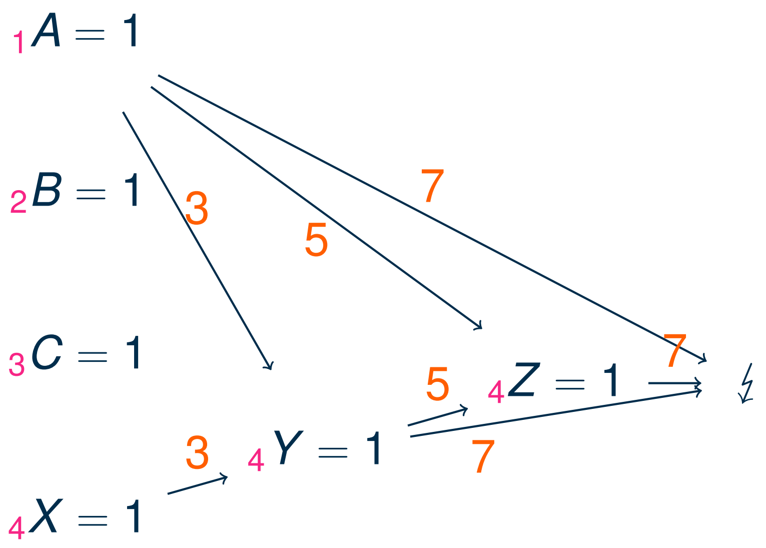
The sink is always the conflicting assignment, and the sources are the desicion literals involved in the conflict.

We can use this to determine the reasons for the conflict.

Example: Implication Graph

Implication graph for the conflicting state under the trail A, B, C, X, Y, Z.
The edge labels denote clauses, node labels indicate a variable assignment and its decision level.

- Consider inferring the clause $\{\neg A, \neg X\}$ by the following resolution steps: $(7 \circ_Z 5) \circ_Y 3$.
- Learning of $\{\neg A, \neg X\}$ prevents the solver choosing the same partial assignment again.



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

Conflict Analysis: Implementation

Implement trail as stack of literals together with a pointer to the reason clause (*null* for decisions) and the decision level. On each conflict, use the trail to trace back the implications to the conflict sources.

Example: Trail with conflicting clause $\{\neg A, \neg Y, \neg Z\}$

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

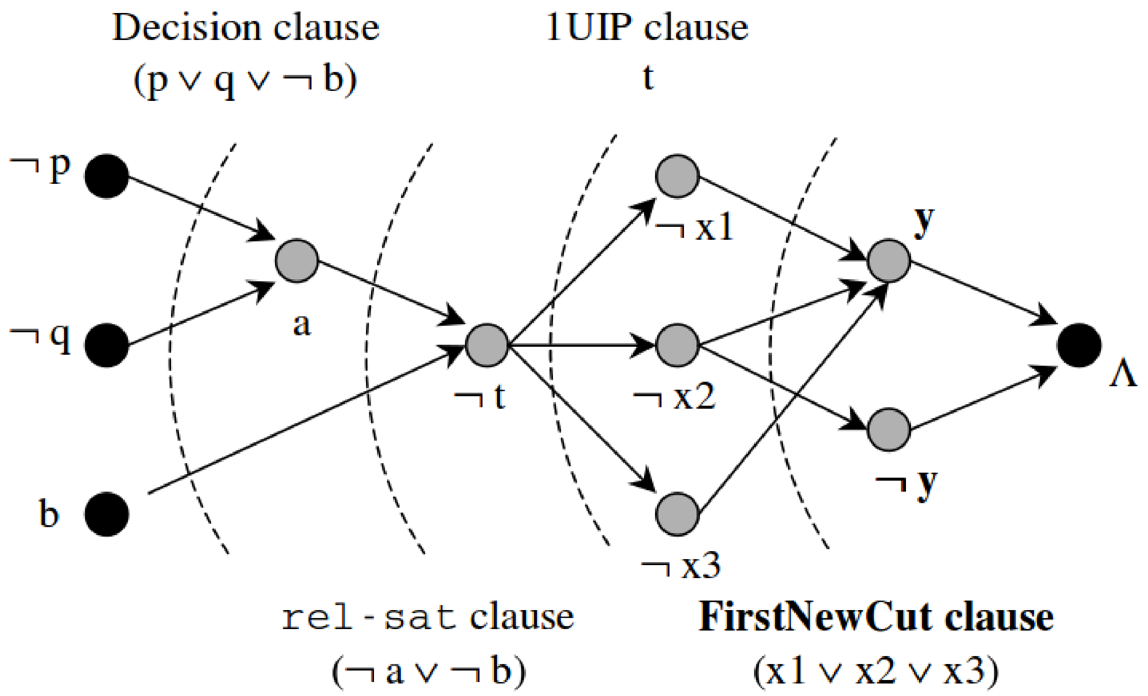
Trail Resolution:

- $\{\neg A, \neg Y, \neg Z\} \otimes_Z \{\neg A, \neg Y, Z\} = \{\neg A, \neg Y\}$
- $\{\neg A, \neg Y\} \otimes_Y \{\neg A, \neg X, Y\} = \{\neg A, \neg X\}$
- Conflict 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 ↯, if all paths to ↯ contain *v*
- FirstUIP: “first” dominator (seen from conflict side)



[Beame et al, 2003]

Conflict Analysis: 1-UIP Learning

- FirstUIP Clause:

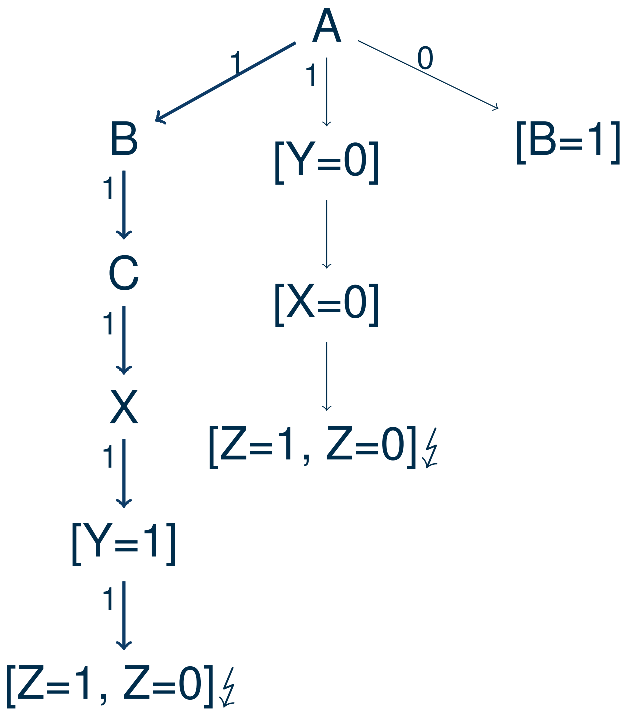
  Resolve the conflicting clause and reason clauses until only a single literal of the current decision level remains.

- 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.

- In 1-UIP learning, 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:

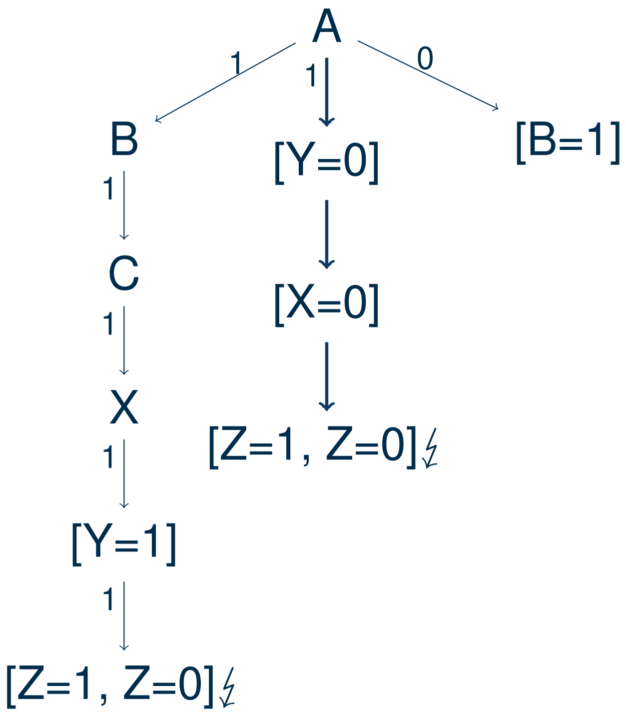


$$F = \{\{A, B\}, \{B, C\},\\ \{\neg A, \neg X, Y\},\\ \{\neg A, X, Z\},\\ \{\neg A, \neg Y, Z\},\\ \{\neg A, X, \neg Z\},\\ \{\neg A, \neg Y, \neg Z\}\}$$

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:



$$
\begin{aligned}
F = \{&\{A, B\}, \{B, C\},\\
&\{\neg A, \neg X, Y\},\\
&\{\neg A, X, Z\},\\
&\{\neg A, \neg Y, Z\},\\
&\{\neg A, X, \neg Z\},\\
&\{\neg A, \neg Y, \neg Z\}\\
&\{\neg A, \neg Y\}\}
\end{aligned}
$$

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

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

Clause Learning: Further Reading

- Clause Size Reduction with all-UIP Learning (Feng and Bacchus, 2020)
- Efficient All-UIP Learned Clause Minimization (Fleury and Biere, 2021)

Resolution Proof

Properties of conflict clause C

- $F \models C$
- $F \cup \neg C \vdash_{UP} \bot$
- $\forall D \subseteq C, D \notin F$

Certificates for Unsatisfiability

- sequence of learned clauses serves as a proof of unsatisfiability
- can be used to validate the correctness of the SAT result in high risk applications

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