

Practical SAT Solving

Lecture 4

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



Recap. Lecture 3: Classic Algorithms

- Local Search
- Resolution
- DP Algorithm
- DPLL Algorithm

Overview

Recap. Lecture 3: Classic Algorithms

- Local Search
- Resolution
- DP Algorithm
- DPLL Algorithm

Today's Topics

- Classic Heuristics: Branching Order, Branching Polarity, Restart Strategies
- Modern SAT Solving 1: Conflict Analysis, Clause Learning

DPLL Algorithm: Iterative Variant

Decision Heuristics:

- Branching Order:
Which variable to choose?
- Branching Polarity:
Which value to assign?

Algorithm 1: 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      $L \leftarrow$  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      $L \leftarrow$  pick an unassigned literal
9     push  $\{L = 1\}$  on Trail
10 return SAT

```

Decision Heuristics

Properties of Decision Heuristics

- Desired properties:
 - Fast to compute
 - Yields easy sub-problems
→ Maximize unit propagations
- 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**
- $H_i(x) := \alpha \max(h_i(x), h_i(\bar{x})) + \beta \min(h_i(x), h_i(\bar{x}))$ (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 $H_i(x) = H_i(\bar{x})$)
→ select x if $\sum_i h_i(x) \geq \sum_i h_i(\bar{x})$
- 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 $S(x) = (f^*(x) + f^*(\bar{x})) \cdot 2^k + f^*(x) \cdot f^*(\bar{x})$ (let $k := 10$, for example)

Properties of Mom's Heuristic

Goal: assign variables with **high occurrence in short clauses**

- Separate polarity heuristic
→ for example, select x if $f^*(\bar{x}) \geq f^*(x)$
- depends on literal occurrence counts over the not yet satisfied clauses
- Popular in the mid 90s (Find some variants in **Freeman 1995**)

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
→ for example, use conflict-seeking polarity heuristic
- **Two-sided variant**: choose variable x with maximum $J(x) + J(\bar{x})$
→ 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 occurrences $C_N(x)$ 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

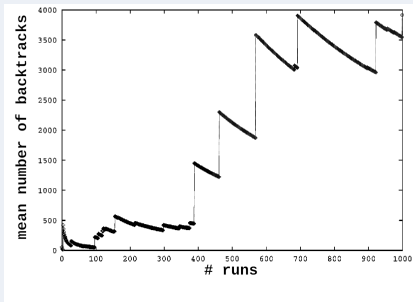
Next up

Restart Strategies

Restarts Strategies: Motivation

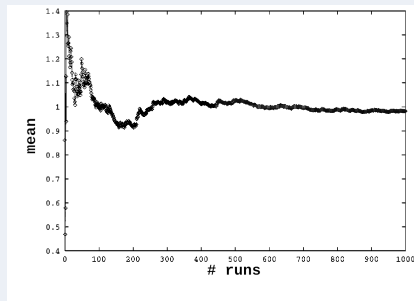
Given n runs of randomized DPLL search, what is the average number of backtracks per run (relative to n)?

Heavy-tailed Distribution



⇒ Heavy-tailed distribution:

$$P[X > x] \sim C \cdot x^{-\alpha}, \quad C > 0, \alpha \in (0, 2)$$



— vs. Standard distribution: $P[X > x] \sim \frac{1}{x\sqrt{2\pi}} e^{-x^2/2}$

[From: Gomes et al. 2000]

Restart Strategies

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

Why Restart?

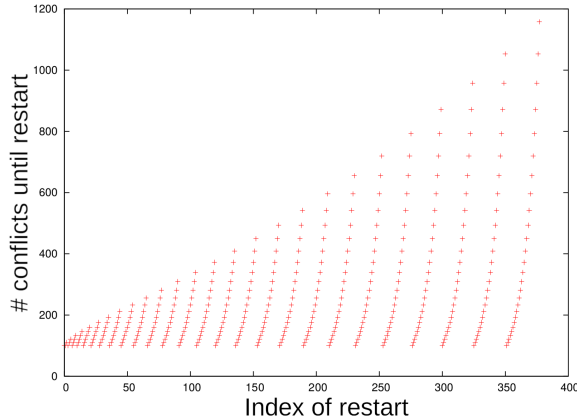
- To recover from **bad branching decisions** and solve more instances
- Might decrease performance on **easy instances**

When to Restart?

- After **some number of conflicts / backtracks**
- The intervals between restarts should increase **to guarantee completeness**
- How much increase?
 - 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

Inner / Outer Pattern (MiniSat)



Algorithm 2: 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, Sinclair, Zuckerman 1993)

Consider a *Las Vegas* algorithm A (i.e., correct but with *random run time*) and a *restart strategy* $S = \langle t_1, t_2, \dots \rangle$ (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.

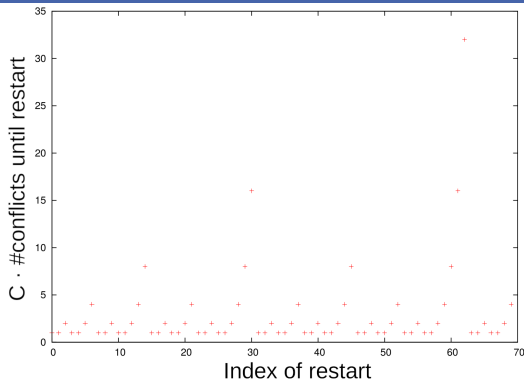
$$\text{Luby} = u \cdot (t_i)_{i \in \mathbb{N}} \quad \text{with} \quad t_i = \begin{cases} 2^{k-1} & \text{if } i = 2^k - 1 \\ t_{i-2^{k-1}+1} & \text{if } 2^{k-1} \leq i \leq 2^k - 1 \end{cases}$$

Example (Luby Sequence)

1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, ...

Restart Strategies: Luby Sequence

Luby Sequence: Naive Implementation



Algorithm 3: Luby Sequence

Input: int i

```

1 for  $k = 1$  to 32 do
2   if  $i == (1 \ll k) - 1$  then
3     return  $1 \ll (k - 1)$ 
4 for  $k = 1$  to  $\infty$  do
5   if  $(1 \ll (k - 1)) \leq i \leq (1 \ll k) - 1$  then
6     return Luby( $i - (1 \ll (k - 1)) + 1$ )

```

run DPLL() with conflict-limit $512 \cdot \text{Luby}(++\text{restarts})$

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) :

$$(u_1, v_1) = (1, 1);$$

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

Example (Luby Sequence)

$(1,1), (2,1), (2,2), (3,1), (4,1), (4,2), (4,4), (5,1), \dots$

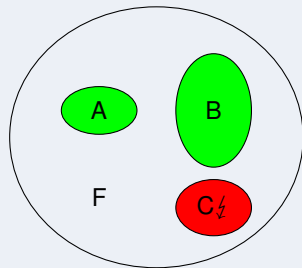
Branching Polarity: Phase Saving

Observation: Frequent restarts **decrease performance** on some satisfiable instances

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**



Example: *A* and *B* are satisfied, search works on component *C*

Recap

Decision Heuristics

Restart Strategies

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

Next up

Clause Learning

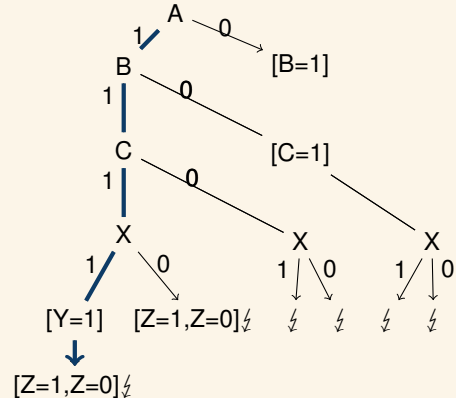
DPLL: Backtracking

Example: 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)



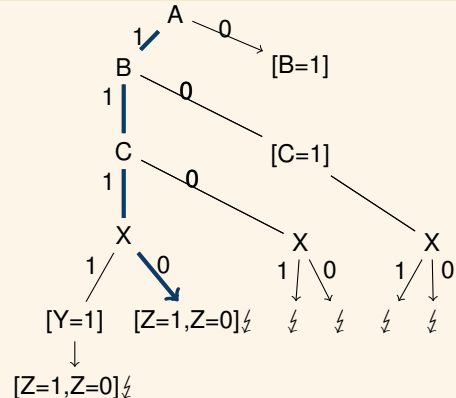
DPLL: Backtracking

Example: 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)



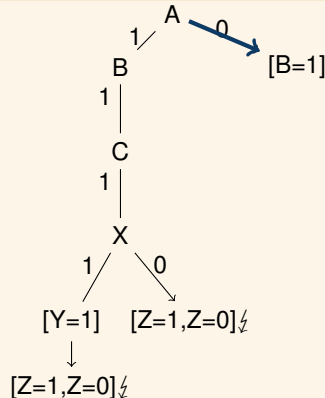
DPLL: Backtracking

Example: 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

Given: Formula F , assignment trail T and conflicting clause C .

Definition: Implication Graph

The implication graph is a DAG $G = (V \cup \{\bot\}, E)$ of

- vertices $[\ell_i, d_i]$ for each literal ℓ_i with decision level d_i on the trail
- vertex \bot representing the conflicting assignment
- Note: all literals of C have edges to \bot
- edges $([\ell_i, d_i], [u_{i,j}, d_i])$ for each propagated literal $u_{i,j}$ at decision level d_i

Observations:

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

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

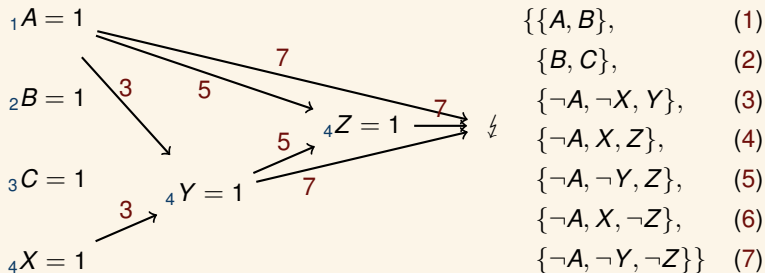
Conflict Analysis

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.



Conflict Analysis: Implementation

Implement trail as **stack of literals** together with a pointer to the **reason clause** (nullptr 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 $\{\neg A, \neg Y, \neg Z\}$)

\downarrow	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

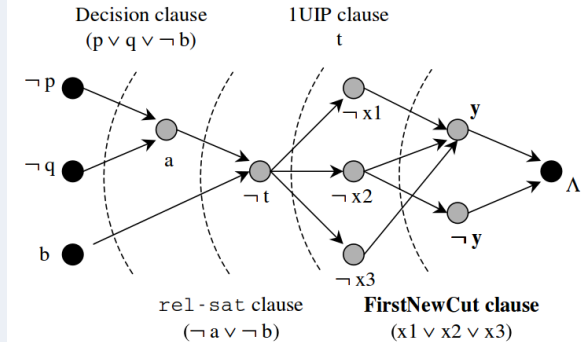
Trail Resolution:

- $\{\neg A, \neg Y, \neg Z\} \otimes_Z \{\neg A, \neg Y, Z\} = \{\neg A, \neg Y\}$
- $\{\neg A, \neg Y\} \otimes_Y \{\neg A, \neg X, Y\} = \{\neg A, \neg X\}$
- Conflict Clause $C = \{\neg A, \neg X\}$
- Backtrack Level $b = 1$

Conflict Analysis: Unit Implication Points (UIP)

Several possibilities to learn a clause from an implication graph

- UIP is a dominator in the implication graph
- A node v is a dominator for $\frac{\perp}{\perp}$, if all paths to $\frac{\perp}{\perp}$ contain v
- FirstUIP: “first” dominator (seen from conflict side)



*[From: Beame et al. 2003]

Conflict Analysis: Unit Implication Points (UIP)

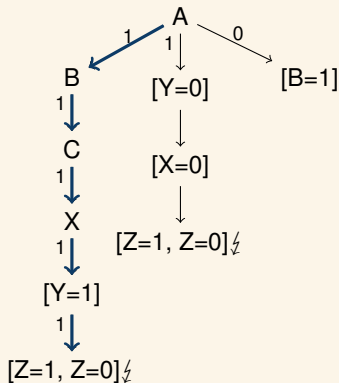
1-UIP Learning

- FirstUIP-clause: resolve 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. Algorithm becomes easier: backtrack until clause becomes asserting
- The assertion level is the second highest level in a conflict clause

Backtracking with Clause Learning

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

Example: Non-chronological Backtracking



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

Trail: A, B, C, X, Y, Z

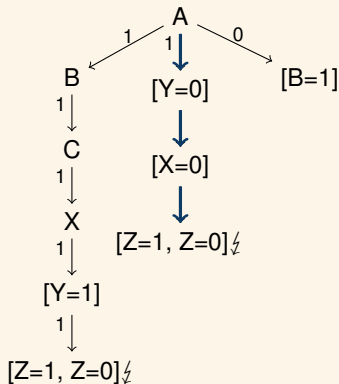
Conflicting Clause: $\{ \neg A, \neg Y, \neg Z \}$

Conflict Clause (1UIP): $\{ \neg A, \neg Y \}$

Backtracking with Clause Learning

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

Example: Non-chronological Backtracking



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

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

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

Resolution Proof

Properties of conflict clause C

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

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** (e.g., verification)

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 Polarity: Phase Saving
- Conflict Analysis, Clause Learning