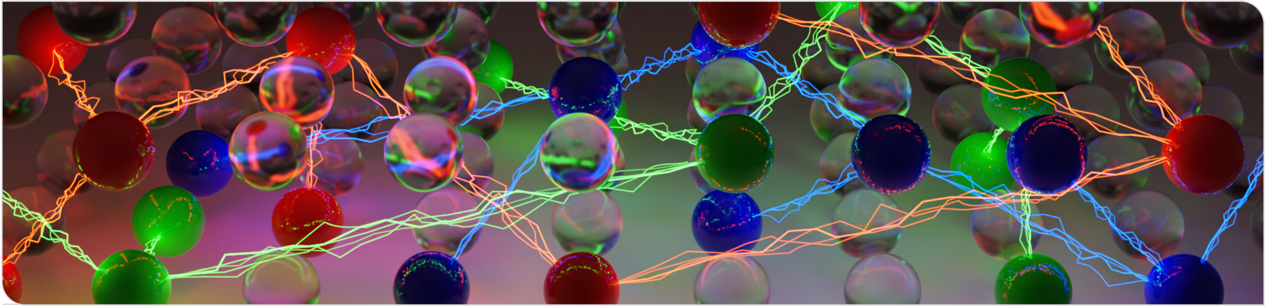
Karlsruher Institut für Technologie

Practical SAT Solving

Lecture 4

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



Overview

Recap. Lecture 3: Classic Algorithms

- Local Search
- Resolution
- DP Algorithm
- DPLL Algorithm

Overview

Karlsruher Institut für Technologie

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)

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

Decision Heuristics

Properties of Decision Heuristics

- Desired properties:
  - Fast to compute
  - Yields easy sub-problems
    $\rightarrow$ 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
- $H_i(x) := \alpha \max(h_i(x), h_i(\overline{x})) + \beta \min(h_i(x), h_i(\overline{x}))$ (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)

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

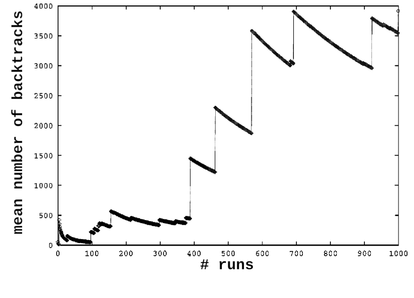
Next up

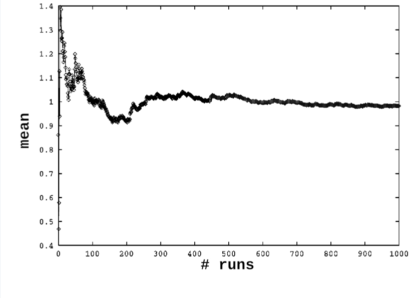
Restart Strategies

Restarts Strategies: Motivation

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

Heavy-tailed Distribution





$\Rightarrow$ 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?

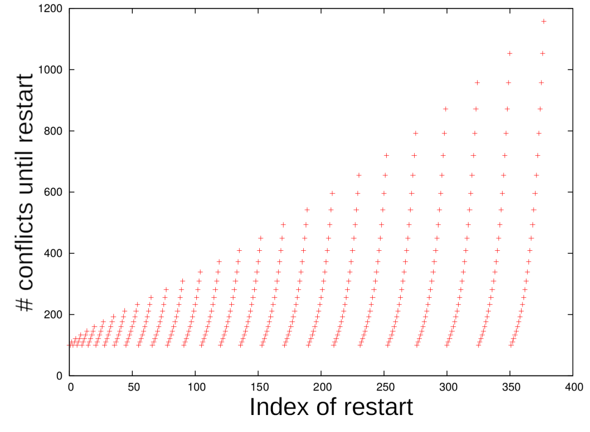
- To recover from bad branching decisions and solve more instances
- Might decrease performance on easy instances

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

```
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: 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 $S = \langle t_1, t_2, \ldots \rangle$ (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.

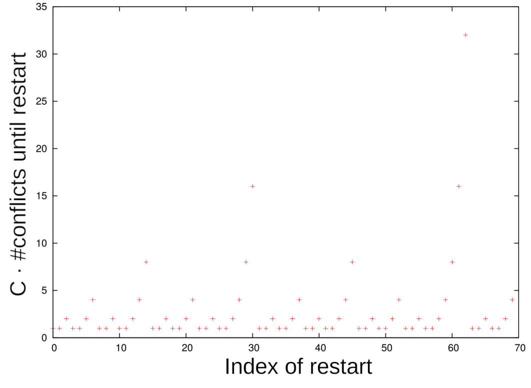
$$Luby = u \cdot (t_i)_{i \in \mathbb{N}} \quad \text{with} \quad t_i = \begin{cases} 2^{k-1} & \text{if } i = 2^k - 1 \\ t_{i-2^{k-1}+1} & \text{if } 2^{k-1} \leq i \leq 2^k - 1 \end{cases}$$

Example (Luby Sequence)

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

Restart Strategies: Luby Sequence

Luby Sequence: Naive Implementation



Algorithm 3: 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):

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

Example (Luby Sequence)

(1,1), (2,1), (2,2), (3,1), (4,1), (4,2), (4,4), (5,1), . . .

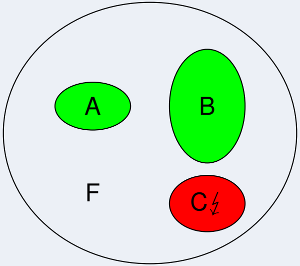
Branching Polarity: Phase Saving

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

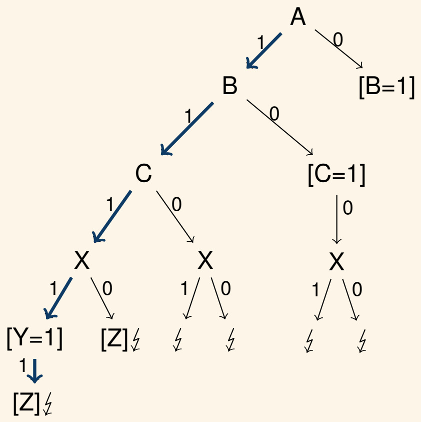
- Inner / Outer Pattern
- Luby Sequence / Reluctant Doubling
- Phase Saving / Assignment Caching

Next up

Clause Learning

DPLL: Backtracking

Example: Chronological Backtracking

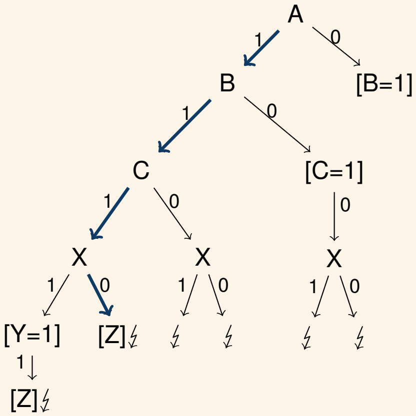


$$F = \{\{A, B\},\\ \{B, C\},\\ \{\neg A, \neg X, Y\},\\ \{\neg A, X, Z\},\\ \{\neg A, \neg Y, Z\},\\ \{\neg A, X, \neg Z\},\\ \{\neg A, \neg Y, \neg Z\}\}$$

- Trail: A, B, C, X, Y, Z
- Conflicting Clause: $\{\neg A, \neg Y, \neg Z\}$

DPLL: Backtracking

Example: Chronological Backtracking

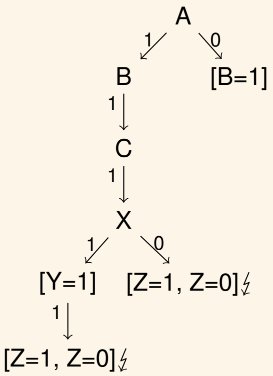


$$F = \{\{A, B\},\ \{B, C\},\ \{\neg A, \neg X, Y\},\ \{\neg A, X, Z\},\ \{\neg A, \neg Y, Z\},\ \{\neg A, X, \neg Z\},\ \{\neg A, \neg Y, \neg Z\}\}$$

- Trail: $A, B, C, \neg X, Z$
- Conflicting Clause: $\{\neg A, X, \neg Z\}$

DPLL: Backtracking

Example: Backjumping



- The first two conflicting clauses $\{\neg A, \neg Y, \neg Z\}$, $\{\neg A, X, \neg Z\}$ contain only a fraction of the assignments on the trail
- Assignments to B and C obviously play no role in the present conflicting state and we could immediately backtrack to flip the assignment to A
- How to find out which assignments on the trail are relevant for the actual conflict?

Implication Graph

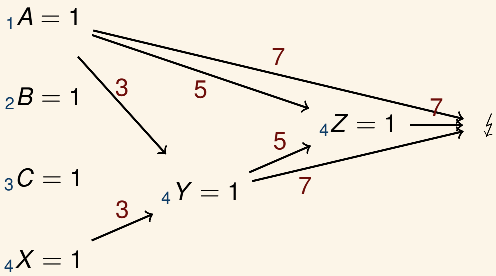
Given: Formula F, assignment trail T and conflicting clause C.
The implication graph is a DAG $G = (V \cup \{\lightning\}, E)$ of

- vertex $\lightning$ representing the conflicting assignment (all literals of C have edges to $\lightning$)
- vertices $v := [x = b, d]$ for each assignment $x = b$ (with decision level d)
- edges $e := ([l_i = 0, d_i], [u = 1, d_i])$ for each unit propagation of a literal u from $[l_i = 0, d_i]$ (due to a clause $\{l_1, \ldots, l_k, u\}$)

Conflict Analysis

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

Example: Implication Graph

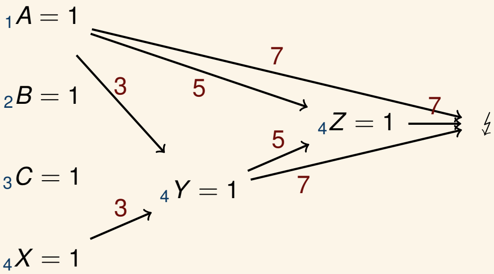


$$
\begin{aligned}
F = \{\{A, B\}, \quad & =: 1 \\
\{B, C\}, \quad & =: 2 \\
\{\neg A, \neg X, Y\}, \quad & =: 3 \\
\{\neg A, X, Z\}, \quad & =: 4 \\
\{\neg A, \neg Y, Z\}, \quad & =: 5 \\
\{\neg A, X, \neg Z\}, \quad & =: 6 \\
\{\neg A, \neg Y, \neg Z\}\} \quad & =: 7
\end{aligned}
$$

Conflict Analysis

- the sink is always the conflicting assignment
- the sources are the desicion literals that take part in the conflict
- we can use it to detect the *reasons* for a conflict

Example: Implication Graph

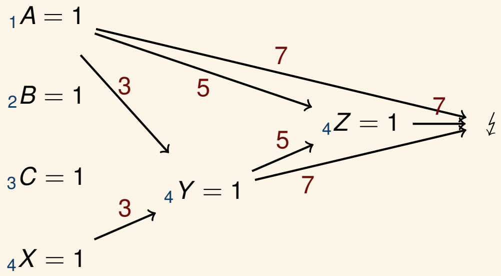


$$
\begin{aligned}
F = \{&\{A, B\}, && =: 1\\
&\{B, C\}, && =: 2\\
&\{\neg A, \neg X, Y\}, && =: 3\\
&\{\neg A, X, Z\}, && =: 4\\
&\{\neg A, \neg Y, Z\}, && =: 5\\
&\{\neg A, X, \neg Z\}, && =: 6\\
&\{\neg A, \neg Y, \neg Z\}\} && =: 7
\end{aligned}
$$

Conflict Analysis

In our example we can learn the clause $\{\neg A, \neg X\}$ in order to prevent the solver to pick the same partial assignment again. This can also be expressed as a sequence of resolution steps: $C = (7 \circ_Z 5) \circ_Y 3$

Example: Implication Graph



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

Implementation: Conflict Analysis

- for each assignment store a pointer to the *reason clause* and the *decision level* on a stack
- decision literals store a null pointer
- with the clause pointers and the trail we can trace all the implications back to their sources

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

$\lightning$	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

Implementation: Conflict Analysis

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

$\lightning$	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 $C = \{\neg A, \neg X\}$
- Backtrack Level $b = 1$

Properties of conflict clause C

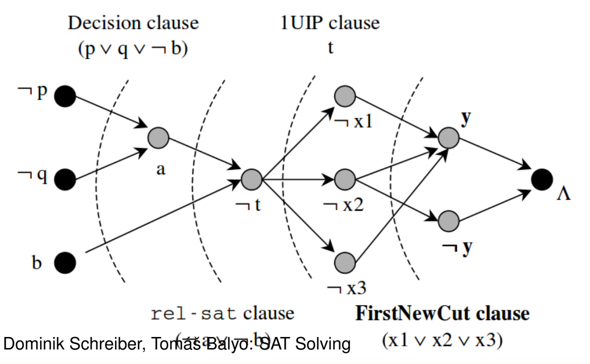
- $F \models C$
- $F \cup \neg C \vdash_{UP} \bot$
- $D \notin F, \forall D \subseteq C$

Unit Implication Points (UIP)

- UIP is a dominator in the implication graph
- A node *v* is a dominator for ↯, if all paths to ↯ contain *v*
- FirstUIP: “first” dominator (seen from conflict side)

Several possibilities to learn a clause from an implication graph

A clause can be learned for every cut in the the implication graph.[a]



Unit Implication Points (UIP)

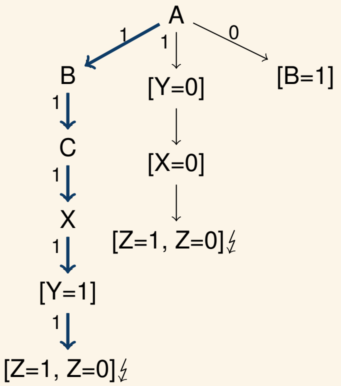
1-UIP Learning

- 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



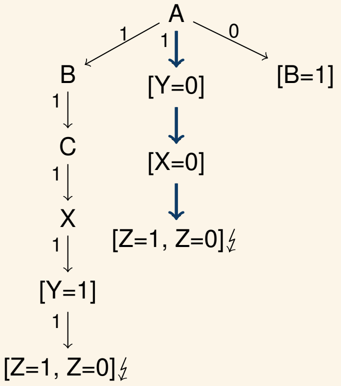
$$
\begin{aligned}
F = \{&\{A, B\}, \{B, C\},\\
&\{\neg A, \neg X, Y\},\\
&\{\neg A, X, Z\},\\
&\{\neg A, \neg Y, Z\},\\
&\{\neg A, X, \neg Z\},\\
&\{\neg A, \neg Y, \neg Z\}\}
\end{aligned}
$$

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

Karlsruher Institut für Technologie

CDCL can derive a complete resolution refutation

- proof can serve as a certificate for validating the correctness of the SAT solver[a]
- resolution refutations based on clause learning find key practical applications (e.g. model checking)
- can help to determine minimally unsatisfiable subsets in an unsatisfiable formula

[a] http://www.cs.utexas.edu/~marijn/drat-trim/