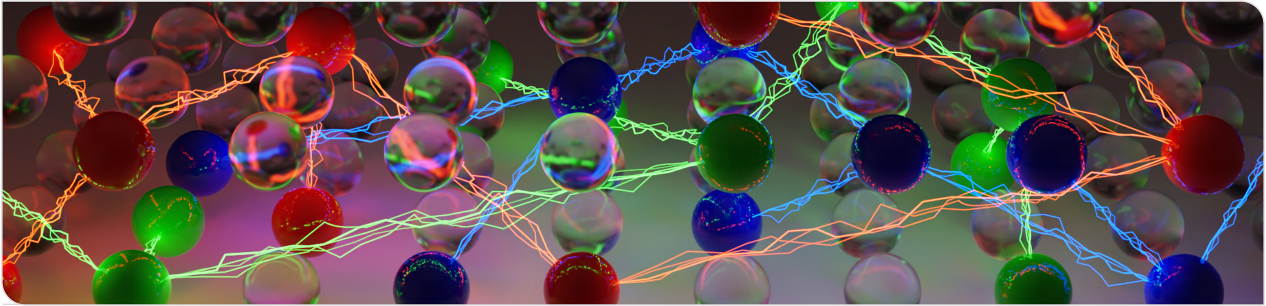
Karlsruher Institut für Technologie

# Practical SAT Solving

## Lecture 3

Markus Iser, Dominik Schreiber, Tomáš Balyo | April 29, 2024



# Overview

## Recap. Lecture 2

- Tractable Subclasses
- Constraint Encodings and their Properties

# Overview

Karlsruher Institut für Technologie

## Recap. Lecture 2

- Tractable Subclasses
- Constraint Encodings and their Properties

## Today's Topics: Elementary SAT Algorithms

- Local Search
- Resolution
- DP Algorithm
- DPLL Algorithm

# Stochastic Local Search (SLS)

## Minimize the Number of Unsatisfied Clauses

Start with a random complete variable assignment $\alpha$:

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 |

Repeatedly flip variables in $\alpha$ to decrease the number of unsatisfied clauses:

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 |

# Stochastic Local Search (SLS)

## Properties of SLS Algorithms

Local search algorithms are incomplete: They cannot show unsatisfiability!

Challenges:

- Which variable should be flipped next?

# Stochastic Local Search (SLS)

## Properties of SLS Algorithms

Local search algorithms are incomplete: They cannot show unsatisfiability!

Challenges:

- Which variable should be flipped next?
  - select variable from an unsatisfied clause
  - select variable that maximizes the number of satisfied clauses
- How to avoid getting stuck in local minima?

# Stochastic Local Search (SLS)

## Properties of SLS Algorithms

Local search algorithms are incomplete: They cannot show unsatisfiability!

Challenges:

- Which variable should be flipped next?
  - select variable from an unsatisfied clause
  - select variable that maximizes the number of satisfied clauses
- How to avoid getting stuck in local minima?
  - randomization

# Classic SLS Algorithms

## GSAT (Selman et al., 1992)

Greedy local search algorithm

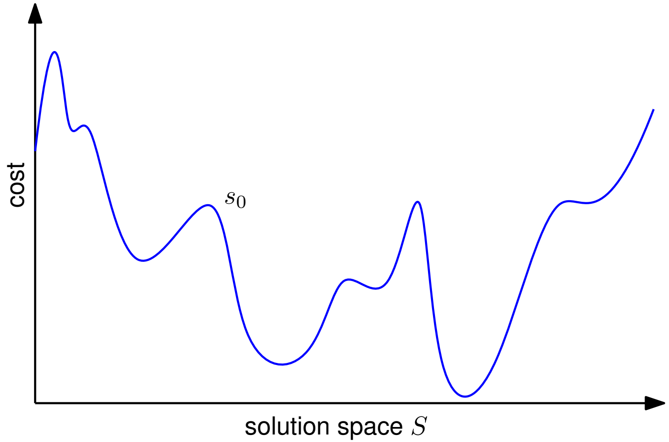
**Algorithm 1:** GSAT

**Input:** ClauseSet *S*

**Output:** Assignment $\alpha$, or Nothing

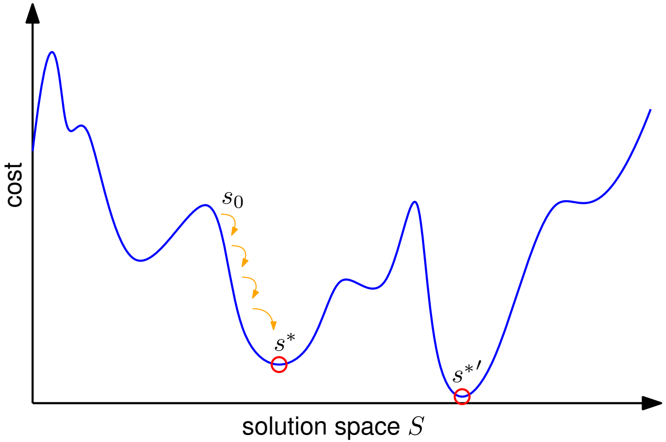
**for** $i = 1$ **to** *MAX_TRIES* **do**<br>
    $\alpha$ = random-assignment to variables in *S*<br>
    **for** $j = 1$ **to** *MAX_FLIPS* **do**<br>
        **if** $\alpha$ *satisfies all clauses in S* **then** **return** $\alpha$<br>
        *x* = variable that produces least number of unsatisfied clauses when flipped<br>
        flip *x*<br>
**return** Nothing `// no solution found`

# SLS: Local Minima



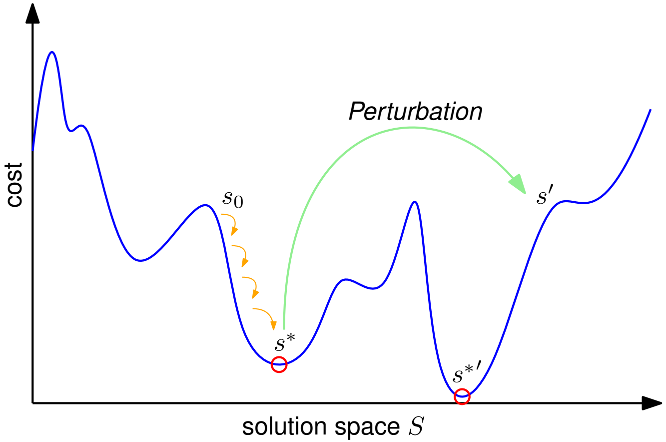

[Illustration Adapted from: Alan Mackworth, UBC, Canada]

# SLS: Local Minima




[Illustration Adapted from: Alan Mackworth, UBC, Canada]

# SLS: Local Minima




[Illustration Adapted from: Alan Mackworth, UBC, Canada]

# Classic SLS Algorithms

## WalkSAT (Selman et al., 1993)

Variant of GSAT

Try to avoid local minima by introducing random noise.

**Algorithm 2:** WalkSAT($S$)

**for** $i = 1$ **to** *MAX_TRIES* **do**
    $\alpha$ = random-assignment to variables in $S$
    **for** $j = 1$ **to** *MAX_FLIPS* **do**
        **if** $\alpha$ *satisfies all clauses in S* **then** **return** $\alpha$
        $C$ = random unsatisfied clause in $S$
        **if** *by flipping an* $x \in C$ *no new unsatisfied clauses emerges* **then** flip $x$
        **else** with probability $p$ flip an $x \in C$ at random
        otherwise, flip a variable that changes the least number of clauses from satisfied to unsatisfied
**return** Nothing `// no solution found`

# SLS: Important Notions

Karlsruher Institut für Technologie

**Consider a flip taking $\alpha$ to $\alpha'$**

| | |
|---|---|
| **breakcount** | number of clauses satisfied in $\alpha$, but not satisfied in $\alpha'$ |
| **makecount** | number of clauses not satisfied in $\alpha$, but satisfied in $\alpha'$ |
| **diffscore** | # unsatisfied clauses in $\alpha$ $-$ # unsatisfied clauses in $\alpha'$ |

Typically, breakcount, makecount, and/or diffscore are used to select the variable to flip.

# SLS: Important Notions

Karlsruher Institut für Technologie

## Consider a flip taking $\alpha$ to $\alpha'$

**breakcount** number of clauses satisfied in $\alpha$, but not satisfied in $\alpha'$

**makecount** number of clauses not satisfied in $\alpha$, but satisfied in $\alpha'$

**diffscore** # unsatisfied clauses in $\alpha$ $-$ # unsatisfied clauses in $\alpha'$

Typically, breakcount, makecount, and/or diffscore are used to select the variable to flip.

## Recap using new nomenclature

**GSAT** select variable with highest diffscore

**WalkSAT** select variable with minimal breakcount

# Stochastic Local Search (SLS)

## Legacy of SLS

- Extremely successful and popular in early days of SAT
  - SLS outperformed early resolution-based solvers, e.g., based on DP or DPLL
  - for example, state of the art engine for automated planning in the 90s

# Stochastic Local Search (SLS)

Karlsruher Institut für Technologie

## Legacy of SLS

- Extremely successful and popular in early days of SAT
  - SLS outperformed early resolution-based solvers, e.g., based on DP or DPLL
  - for example, state of the art engine for automated planning in the 90s
- Today, sophisticated resolution-based systematic search solvers dominate in most practical applications
  - Faster, more reliable, and complete!

# Stochastic Local Search (SLS)

## Legacy of SLS

- Extremely successful and popular in early days of SAT
  - SLS outperformed early resolution-based solvers, e.g., based on DP or DPLL
  - for example, state of the art engine for automated planning in the 90s
- Today, sophisticated resolution-based systematic search solvers dominate in most practical applications
  - Faster, more reliable, and complete!
- Still useful as a component in more complex solvers
  - Part of (parallel) algorithm portfolios
  - Control branching heuristics in complete search algorithms
  - Detection of autarkies in formula simplification algorithms
  - In combination with complete solvers for optimization problems (e.g., MaxSAT)

# Recap

## Elementary Algorithms

- Local Search
  - Examples: GSAT, WalkSAT
  - Terminology: breakcount, makecount, diffscore

# Recap

## Elementary Algorithms

- Local Search
  - Examples: GSAT, WalkSAT
  - Terminology: breakcount, makecount, diffscore

## Next Up

- Resolution

# Resolution

Karlsruher Institut für Technologie

## The Resolution Rule

$$\frac{P_1 \cup \{x\}, \quad P_2 \cup \{\neg x\}}{P_1 \cup P_2}$$

Resolution is a logical inference rule to infer a conclusion (resolvent) from given premises (input clauses).

## Example (Resolution)

$$\{x_1, x_3, \neg x_7\}, \{\neg x_1, x_2\} \vdash \{x_3, \neg x_7, x_2\}$$

$$\{x_4, x_5\}, \{\neg x_5\} \vdash \{x_4\}$$ (Fact)

$$\{x_1, x_2\}, \{\neg x_1, \neg x_2\} \vdash \{x_1, \neg x_1\}$$ (Tautological Resolvent)

$$\{x_1\}, \{\neg x_1\} \vdash \{\}$$ (Empty Clause)

# Resolution

## Theorem: Resolution is Sound

Given a CNF formula $F$ with two resolvable clauses $C_1, C_2 \subseteq F$ with resolvent $\mathrm{R}(C_1, C_2)$, the following holds:

$$F \quad \equiv \quad F \wedge \mathrm{R}(C_1, C_2)$$

## Proof

Let $C_1 := \{x\} \cup P_1$ and $C_2 := \{\neg x\} \cup P_2$ such that $\mathrm{R}(C_1, C_2) = P_1 \cup P_2 =: D$.

**Soundness:** $F \vdash F \wedge D \implies F \models F \wedge D$

Any satisfying assignment $\phi$ of $F$ is also a satisfying assignment of $D$: Since $\phi$ satisfies both $C_1$ and $C_2$, it necessarily satisfies at least one literal in $D$. If $\phi$ satisfies $x$ then it satisfies some literal in $P_2$. Otherwise, if $\phi$ satisfies $\neg x$ then it satisfies some literal in $P_1$.

# Resolution

Karlsruher Institut für Technologie

## Theorem: Resolution is Sound

Given a CNF formula $F$ with two resolvable clauses $C_1, C_2 \subseteq F$ with resolvent $\mathrm{R}(C_1, C_2)$, the following holds:

$$F \quad \equiv \quad F \wedge \mathrm{R}(C_1, C_2)$$

## Proof

Let $C_1 := \{x\} \cup P_1$ and $C_2 := \{\neg x\} \cup P_2$ such that $\mathrm{R}(C_1, C_2) = P_1 \cup P_2 =: D$.

**Soundness:** $F \vdash F \wedge D \implies F \models F \wedge D$

Any satisfying assignment $\phi$ of $F$ is also a satisfying assignment of $D$: Since $\phi$ satisfies both $C_1$ and $C_2$, it necessarily satisfies at least one literal in $D$. If $\phi$ satisfies $x$ then it satisfies some literal in $P_2$. Otherwise, if $\phi$ satisfies $\neg x$ then it satisfies some literal in $P_1$.

**Equivalence:** $F \vdash F \wedge D \implies F \wedge D \models F$

Since $D$ does not introduce new variables, $F \wedge D$ can not have more satisfying assignments than $F$.

# Resolution

## Resolution is Sound and Refutation Complete

- If we manage to infer the empty clause from a CNF formula $F$, then $F$ is unsatisfiable. (sound)
- If $F$ is unsatisfiable, then there exists a refutation by resolution. (complete)
- Not all possible consequences of $F$ can be derived by resolution. ("only" refutation complete)

## Resolution Proof

A resolution proof for $F$ is a sequence of clauses $\langle C_1, C_2, \ldots, C_{k-1}, C_k = \emptyset \rangle$ where each $C_i$ is either an original clause of $F$ or a resolvent of two earlier clauses.

## Example (Resolution Proof)

$$F = \{x_1, x_2\}, \{\neg x_1, x_2\}, \{x_1, \neg x_2\}, \{\neg x_1, \neg x_2\} \qquad \text{(Formula)}$$

(Refutation)

# Resolution

## Resolution is Sound and Refutation Complete

- If we manage to infer the empty clause from a CNF formula $F$, then $F$ is unsatisfiable. (sound)
- If $F$ is unsatisfiable, then there exists a refutation by resolution. (complete)
- Not all possible consequences of $F$ can be derived by resolution. ("only" refutation complete)

## Resolution Proof

A resolution proof for $F$ is a sequence of clauses $\langle C_1, C_2, \ldots, C_{k-1}, C_k = \emptyset \rangle$ where each $C_i$ is either an original clause of $F$ or a resolvent of two earlier clauses.

## Example (Resolution Proof)

$$F = \{x_1, x_2\}, \{\neg x_1, x_2\}, \{x_1, \neg x_2\}, \{\neg x_1, \neg x_2\} \quad \text{(Formula)}$$

$$\equiv \{x_1, x_2\}, \{\neg x_1, x_2\}, \{x_1, \neg x_2\}, \{\neg x_1, \neg x_2\}, \{x_2\}, \{\neg x_2\}, \{\} \quad \text{(Refutation)}$$

# Saturation Algorithm

### Properties

- sound and complete – always terminates and answers correctly
- exponential time and space complexity

**Algorithm 3:** Saturation Algorithm

**Input:** CNF formula $F$
**Output:** $\{$SAT, UNSAT$\}$

```
while true do
    R := resolveAll(F)
    if R ∩ F ≠ R then  F := F ∪ R
    else  break
if ⊥ ∈ F then  return UNSAT
else  return SAT
```

# Unit Propagation

## Unit Resolution

Resolution where at least one of the resolved clauses is a unit clause, i.e. has size one.

## Example (Unit Resolution)

$$R((x_1 \lor x_7 \lor \neg x_2 \lor x_4), (x_2)) = (x_1 \lor x_7 \lor x_4)$$

# Unit Propagation

## Unit Resolution

Resolution where at least one of the resolved clauses is a unit clause, i.e. has size one.

## Example (Unit Resolution)

$$R((x_1 \lor x_7 \lor \neg x_2 \lor x_4), (x_2)) = (x_1 \lor x_7 \lor x_4)$$

## Unit Propagation

Apply unit resolution until fixpoint is reached.

## Example (Unit Propagation)

Usually, we are only interested in the inferred facts (unit clauses) and conflicts (empty clauses).

$$\{x_1, x_2, x_3\}, \{x_1, \neg x_2\}, \{\neg x_1\} \quad \vdash_1 \quad \{\neg x_2\}, \{x_3\}$$

# Recap

## Elementary Algorithms

- Local Search
  - Examples: GSAT, WalkSAT
  - Terminology: breakcount, makecount, diffscore
- Resolution
  - Soundness and Completeness
  - Saturation Algorithm (Exponential Complexity)
  - Unit Propagation

# Recap

## Elementary Algorithms

- Local Search
  - Examples: GSAT, WalkSAT
  - Terminology: breakcount, makecount, diffscore
- Resolution
  - Soundness and Completeness
  - Saturation Algorithm (Exponential Complexity)
  - Unit Propagation

## Next Up

Davis Putnam (DP) Algorithm (Improving upon saturation-based resolution)

# Davis-Putnam Algorithm (Davis & Putnam, 1960)

Presented in 1960 as a SAT procedure for first-order logic.

## Deduction Rules of DP Algorithm

- **Unit Resolution:** If there is a unit clause $C = \{\overline{x}\} \in F$, simplify all other clauses containing $x$
- **Pure Literal Elimination:** If a literal $x$ never occurs negated in $F$, add clause $\{x\}$ to $F$
- **Case Splitting:** Put $F$ in the form $(A \vee x) \wedge (B \vee \neg x) \wedge R$, where $A$, $B$, and $R$ are clause sets free of $x$. Replace $F$ by the clausification of $(A \vee B) \wedge R$

Apply above deduction rules (prioritizing rules 1 and 2) until one of the following situations occurs:

- $F = \emptyset \quad \rightarrow$ SAT
- $\emptyset \in F \quad \rightarrow$ UNSAT

# Davis-Putnam Algorithm

Karlsruher Institut für Technologie

## Example (DP Algorithm)

$F = \{\{x, y, \neg z, u\}, \{\neg x, y, u\}, \{x, \neg y, \neg z\}, \{z, v\}, \{z, \neg v\}, \{\neg z, \neg u\}, \{\neg x, \neg y, u\}\}$ (Split by $x$)

# Davis-Putnam Algorithm

Karlsruher Institut für Technologie

## Example (DP Algorithm)

$F = \{\{x, y, \neg z, u\}, \{\neg x, y, u\}, \{x, \neg y, \neg z\}, \{z, v\}, \{z, \neg v\}, \{\neg z, \neg u\}, \{\neg x, \neg y, u\}\}$ (Split by $x$)

$A = \{\{y, \neg z, u\}, \{\neg y, \neg z\}\}$ $B = \{\{y, u\}, \{\neg y, u\}\}$ $R = \{\{z, v\}, \{z, \neg v\}, \{\neg z, \neg u\}\}$ $((A \vee B) \wedge R)$

# Davis-Putnam Algorithm

Karlsruher Institut für Technologie

## Example (DP Algorithm)

$F = \{\{x, y, \neg z, u\}, \{\neg x, y, u\}, \{x, \neg y, \neg z\}, \{z, v\}, \{z, \neg v\}, \{\neg z, \neg u\}, \{\neg x, \neg y, u\}\}$ (Split by $x$)

$A = \{\{y, \neg z, u\}, \{\neg y, \neg z\}\}$ $B = \{\{y, u\}, \{\neg y, u\}\}$ $R = \{\{z, v\}, \{z, \neg v\}, \{\neg z, \neg u\}\}$ $((A \vee B) \wedge R)$

$F_1 = \{\{y, \neg z, u\}, \{\neg y, \neg z, u\}, \{z, v\}, \{z, \neg v\}, \{\neg z, \neg u\}\}$ (Split by $y$)

# Davis-Putnam Algorithm

**Example (DP Algorithm)**

$F = \{\{x, y, \neg z, u\}, \{\neg x, y, u\}, \{x, \neg y, \neg z\}, \{z, v\}, \{z, \neg v\}, \{\neg z, \neg u\}, \{\neg x, \neg y, u\}\}$ (Split by $x$)

$A = \{\{y, \neg z, u\}, \{\neg y, \neg z\}\}$ $B = \{\{y, u\}, \{\neg y, u\}\}$ $R = \{\{z, v\}, \{z, \neg v\}, \{\neg z, \neg u\}\}$ $((A \vee B) \wedge R)$

$F_1 = \{\{y, \neg z, u\}, \{\neg y, \neg z, u\}, \{z, v\}, \{z, \neg v\}, \{\neg z, \neg u\}\}$ (Split by $y$)

$A_1 = \{\{\neg z, u\}\}$ $B_1 = \{\{\neg z, u\}\}$ $R_1 = \{\{z, v\}, \{z, \neg v\}, \{\neg z, \neg u\}\}$ $((A_1 \vee B_1) \wedge R_1)$

# Davis-Putnam Algorithm

**Example (DP Algorithm)**

$F = \{\{x, y, \neg z, u\}, \{\neg x, y, u\}, \{x, \neg y, \neg z\}, \{z, v\}, \{z, \neg v\}, \{\neg z, \neg u\}, \{\neg x, \neg y, u\}\}$ (Split by $x$)

$A = \{\{y, \neg z, u\}, \{\neg y, \neg z\}\}$ $B = \{\{y, u\}, \{\neg y, u\}\}$ $R = \{\{z, v\}, \{z, \neg v\}, \{\neg z, \neg u\}\}$ $((A \vee B) \wedge R)$

$F_1 = \{\{y, \neg z, u\}, \{\neg y, \neg z, u\}, \{z, v\}, \{z, \neg v\}, \{\neg z, \neg u\}\}$ (Split by $y$)

$A_1 = \{\{\neg z, u\}\}$ $B_1 = \{\{\neg z, u\}\}$ $R_1 = \{\{z, v\}, \{z, \neg v\}, \{\neg z, \neg u\}\}$ $((A_1 \vee B_1) \wedge R_1)$

$F_2 = \{\{\neg z, u\}, \{z, v\}, \{z, \neg v\}, \{\neg z, \neg u\}\}$ (Split by $z$)

# Davis-Putnam Algorithm

Karlsruher Institut für Technologie

## Example (DP Algorithm)

$$F = \{\{x, y, \neg z, u\}, \{\neg x, y, u\}, \{x, \neg y, \neg z\}, \{z, v\}, \{z, \neg v\}, \{\neg z, \neg u\}, \{\neg x, \neg y, u\}\} \qquad \text{(Split by } x\text{)}$$

$$A = \{\{y, \neg z, u\}, \{\neg y, \neg z\}\} \quad B = \{\{y, u\}, \{\neg y, u\}\} \quad R = \{\{z, v\}, \{z, \neg v\}, \{\neg z, \neg u\}\} \qquad ((A \vee B) \wedge R)$$

$$F_1 = \{\{y, \neg z, u\}, \{\neg y, \neg z, u\}, \{z, v\}, \{z, \neg v\}, \{\neg z, \neg u\}\} \qquad \text{(Split by } y\text{)}$$

$$A_1 = \{\{\neg z, u\}\} \quad B_1 = \{\{\neg z, u\}\} \quad R_1 = \{\{z, v\}, \{z, \neg v\}, \{\neg z, \neg u\}\} \qquad ((A_1 \vee B_1) \wedge R_1)$$

$$F_2 = \{\{\neg z, u\}, \{z, v\}, \{z, \neg v\}, \{\neg z, \neg u\}\} \qquad \text{(Split by } z\text{)}$$

$$A_2 = \{\{v\}, \{\neg v\}\} \quad B_2 = \{\{u\}, \{\neg u\}\} \quad R_2 = \{\} \qquad ((A_2 \vee B_2) \wedge R_2)$$

# Davis-Putnam Algorithm

Karlsruher Institut für Technologie

**Example (DP Algorithm)**

$F = \{\{x, y, \neg z, u\}, \{\neg x, y, u\}, \{x, \neg y, \neg z\}, \{z, v\}, \{z, \neg v\}, \{\neg z, \neg u\}, \{\neg x, \neg y, u\}\}$ (Split by $x$)

$A = \{\{y, \neg z, u\}, \{\neg y, \neg z\}\}$ $B = \{\{y, u\}, \{\neg y, u\}\}$ $R = \{\{z, v\}, \{z, \neg v\}, \{\neg z, \neg u\}\}$ $((A \vee B) \wedge R)$

$F_1 = \{\{y, \neg z, u\}, \{\neg y, \neg z, u\}, \{z, v\}, \{z, \neg v\}, \{\neg z, \neg u\}\}$ (Split by $y$)

$A_1 = \{\{\neg z, u\}\}$ $B_1 = \{\{\neg z, u\}\}$ $R_1 = \{\{z, v\}, \{z, \neg v\}, \{\neg z, \neg u\}\}$ $((A_1 \vee B_1) \wedge R_1)$

$F_2 = \{\{\neg z, u\}, \{z, v\}, \{z, \neg v\}, \{\neg z, \neg u\}\}$ (Split by $z$)

$A_2 = \{\{v\}, \{\neg v\}\}$ $B_2 = \{\{u\}, \{\neg u\}\}$ $R_2 = \{\}$ $((A_2 \vee B_2) \wedge R_2)$

$F_3 = \{\{u, v\}, \{u, \neg v\}, \{\neg u, v\}, \{\neg u, \neg v\}\}$ (Split by $u$)

# Davis-Putnam Algorithm

Karlsruher Institut für Technologie

## Example (DP Algorithm)

$F = \{\{x, y, \neg z, u\}, \{\neg x, y, u\}, \{x, \neg y, \neg z\}, \{z, v\}, \{z, \neg v\}, \{\neg z, \neg u\}, \{\neg x, \neg y, u\}\}$ (Split by $x$)

$A = \{\{y, \neg z, u\}, \{\neg y, \neg z\}\}$ $B = \{\{y, u\}, \{\neg y, u\}\}$ $R = \{\{z, v\}, \{z, \neg v\}, \{\neg z, \neg u\}\}$ $((A \vee B) \wedge R)$

$F_1 = \{\{y, \neg z, u\}, \{\neg y, \neg z, u\}, \{z, v\}, \{z, \neg v\}, \{\neg z, \neg u\}\}$ (Split by $y$)

$A_1 = \{\{\neg z, u\}\}$ $B_1 = \{\{\neg z, u\}\}$ $R_1 = \{\{z, v\}, \{z, \neg v\}, \{\neg z, \neg u\}\}$ $((A_1 \vee B_1) \wedge R_1)$

$F_2 = \{\{\neg z, u\}, \{z, v\}, \{z, \neg v\}, \{\neg z, \neg u\}\}$ (Split by $z$)

$A_2 = \{\{v\}, \{\neg v\}\}$ $B_2 = \{\{u\}, \{\neg u\}\}$ $R_2 = \{\}$ $((A_2 \vee B_2) \wedge R_2)$

$F_3 = \{\{u, v\}, \{u, \neg v\}, \{\neg u, v\}, \{\neg u, \neg v\}\}$ (Split by $u$)

$A_3 = \{\{v\}, \{\neg v\}\}$ $B_3 = \{\{v\}, \{\neg v\}\}$ $R_3 = \{\}$ $((A_3 \vee B_3) \wedge R_3)$

# Davis-Putnam Algorithm

Karlsruher Institut für Technologie

## Example (DP Algorithm)

$F = \{\{x, y, \neg z, u\}, \{\neg x, y, u\}, \{x, \neg y, \neg z\}, \{z, v\}, \{z, \neg v\}, \{\neg z, \neg u\}, \{\neg x, \neg y, u\}\}$ (Split by $x$)

$A = \{\{y, \neg z, u\}, \{\neg y, \neg z\}\}$ $B = \{\{y, u\}, \{\neg y, u\}\}$ $R = \{\{z, v\}, \{z, \neg v\}, \{\neg z, \neg u\}\}$ $((A \vee B) \wedge R)$

$F_1 = \{\{y, \neg z, u\}, \{\neg y, \neg z, u\}, \{z, v\}, \{z, \neg v\}, \{\neg z, \neg u\}\}$ (Split by $y$)

$A_1 = \{\{\neg z, u\}\}$ $B_1 = \{\{\neg z, u\}\}$ $R_1 = \{\{z, v\}, \{z, \neg v\}, \{\neg z, \neg u\}\}$ $((A_1 \vee B_1) \wedge R_1)$

$F_2 = \{\{\neg z, u\}, \{z, v\}, \{z, \neg v\}, \{\neg z, \neg u\}\}$ (Split by $z$)

$A_2 = \{\{v\}, \{\neg v\}\}$ $B_2 = \{\{u\}, \{\neg u\}\}$ $R_2 = \{\}$ $((A_2 \vee B_2) \wedge R_2)$

$F_3 = \{\{u, v\}, \{u, \neg v\}, \{\neg u, v\}, \{\neg u, \neg v\}\}$ (Split by $u$)

$A_3 = \{\{v\}, \{\neg v\}\}$ $B_3 = \{\{v\}, \{\neg v\}\}$ $R_3 = \{\}$ $((A_3 \vee B_3) \wedge R_3)$

$F_4 = \{\{v\}, \{\neg v\}\} \vdash_1 \{\emptyset\}$ (Unit Resolution)

# DP Variant: Bucket Elimination

## Bucket Elimination

- Fix order $\prec$ on variables.
- **Bucket:** set of clauses with same $\prec$-maximal variable
- **Bucket Elimination:** process buckets in decreasing $\prec$-order
  - resolve all clauses in bucket
  - put resolvents in fitting bucket

# DP Variant: Bucket Elimination

## Example (Bucket Elimination)

$$F = \{(x, y, \overline{z}, u), (\overline{x}, y, u), (x, \overline{y}, \overline{z}), (z, v), (z, \overline{v}), (\overline{z}, \overline{u}), (\overline{x}, \overline{y}, u)\} \qquad (x \succ y \succ z \succ u \succ v)$$

| Variable | Bucket |
|---|---|
| $x$ | $(x, y, \overline{z}, u), (\overline{x}, y, u), (x, \overline{y}, \overline{z}), (\overline{x}, \overline{y}, u)$ |
| $y$ | |
| $z$ | $(z, v), (z, \overline{v}), (\overline{z}, \overline{u})$ |
| $u$ | |
| $v$ | |

# DP Variant: Bucket Elimination

Karlsruher Institut für Technologie

## Example (Bucket Elimination)

$$F = \{(x, y, \overline{z}, u), (\overline{x}, y, u), (x, \overline{y}, \overline{z}), (z, v), (z, \overline{v}), (\overline{z}, \overline{u}), (\overline{x}, \overline{y}, u)\} \qquad (x \succ y \succ z \succ u \succ v)$$

| Variable | Bucket |
|---|---|
| $x$ | processed |
| $y$ | $(y, \overline{z}, u), (\overline{y}, \overline{z}, u)$ |
| $z$ | $(z, v), (z, \overline{v}), (\overline{z}, \overline{u})$ |
| $u$ | |
| $v$ | |

# DP Variant: Bucket Elimination

Karlsruher Institut für Technologie

## Example (Bucket Elimination)

$$F = \{(x, y, \overline{z}, u), (\overline{x}, y, u), (x, \overline{y}, \overline{z}), (z, v), (z, \overline{v}), (\overline{z}, \overline{u}), (\overline{x}, \overline{y}, u)\} \qquad (x \succ y \succ z \succ u \succ v)$$

| Variable | Bucket |
|---|---|
| $x$ | processed |
| $y$ | processed |
| $z$ | $(z, v), (z, \overline{v}), (\overline{z}, \overline{u}), (\overline{z}, u)$ |
| $u$ | |
| $v$ | |

# DP Variant: Bucket Elimination

Karlsruher Institut für Technologie

## Example (Bucket Elimination)

$$F = \{(x, y, \overline{z}, u), (\overline{x}, y, u), (x, \overline{y}, \overline{z}), (z, v), (z, \overline{v}), (\overline{z}, \overline{u}), (\overline{x}, \overline{y}, u)\} \qquad (x \succ y \succ z \succ u \succ v)$$

| Variable | Bucket |
|---|---|
| $x$ | processed |
| $y$ | processed |
| $z$ | processed |
| $u$ | $(\overline{u}, v), (u, v), (\overline{u}, \overline{v}), (u, \overline{v})$ |
| $v$ | |

# DP Variant: Bucket Elimination

Karlsruher Institut für Technologie

## Example (Bucket Elimination)

$$F = \{(x, y, \overline{z}, u), (\overline{x}, y, u), (x, \overline{y}, \overline{z}), (z, v), (z, \overline{v}), (\overline{z}, \overline{u}), (\overline{x}, \overline{y}, u)\} \qquad (x \succ y \succ z \succ u \succ v)$$

| Variable | Bucket |
|---|---|
| $x$ | processed |
| $y$ | processed |
| $z$ | processed |
| $u$ | processed |
| $v$ | $(v), (\overline{v})$ |

# DP: Discussion

*The superiority of the present procedure over those previously available is indicated in part by the fact that a formula on which Gilmore's routine for the IBM 704 causes the machine to compute for 21 minutes without obtaining a result was worked successfully by hand computation using [DP] in 30 minutes.*

—from Davis' and Putnam's Paper

- Does DP improve on saturation's average time complexity?

# DP: Discussion

*The superiority of the present procedure over those previously available is indicated in part by the fact that a formula on which Gilmore's routine for the IBM 704 causes the machine to compute for 21 minutes without obtaining a result was worked successfully by hand computation using [DP] in 30 minutes.*

—from Davis' and Putnam's Paper

- Does DP improve on saturation's average time complexity?

  $\Rightarrow$ yes — if we split over the right variables

# DP: Discussion

> *The superiority of the present procedure over those previously available is indicated in part by the fact that a formula on which Gilmore's routine for the IBM 704 causes the machine to compute for 21 minutes without obtaining a result was worked successfully by hand computation using [DP] in 30 minutes.*
>
> —from Davis' and Putnam's Paper

- Does DP improve on saturation's average time complexity?

  $\Rightarrow$ yes — if we split over the right variables

- Does DP avoid saturation's exponential space complexity?

# DP: Discussion

*The superiority of the present procedure over those previously available is indicated in part by the fact that a formula on which Gilmore's routine for the IBM 704 causes the machine to compute for 21 minutes without obtaining a result was worked successfully by hand computation using [DP] in 30 minutes.*

—from Davis' and Putnam's Paper

- Does DP improve on saturation's average time complexity?

  $\Rightarrow$ yes — if we split over the right variables

- Does DP avoid saturation's exponential space complexity?

  $\Rightarrow$ no — quadratic blowup in size for eliminating one variable

# DPLL Algorithm (Davis et al., 1962)

## Davis Putnam Logemann Loveland (DPLL) Algorithm

- DPLL is a backtracking search over partial variable assignments.
- Case splitting over a variable $x$ branches the search over two cases $x$ and $\neg x$: resulting in the simplified formulas $F_{|x=\texttt{true}}$ and $F_{|x=\texttt{false}}$
- Simplification rules:
  - **Unit Propagation:** If $\{l\} \in F$, $l$ must be set to $\texttt{true}$.
  - **Pure Literal Elimination:** If $x$ occurs only positively (or only negatively), it may be fixed to the respective value.

# DPLL Algorithm

start with simplifications

recurse on subformulas obtained by case-splitting

stop if satisfying assignment found or all branches are unsatisfiable

**Algorithm 4:** DPLL(ClauseSet $S$)

```
while S contains a unit clause {L} do
    delete from S clauses containing L           // unit-subsumption
    delete ¬L from all clauses in S              // unit-resolution
if ∅ ∈ S then return false                       // empty clause
while S contains a pure literal L do
    delete from S all clauses containing L       // pure literal elimination
if S = ∅ then return true                        // no clauses
choose a literal L occurring in S                // case-splitting
if DPLL(S ∪ {{L}}) then return true              // first branch
else if DPLL(S ∪ {{¬L}}) then return true        // second branch
else return false
```

# DPLL Algorithm with Trail

$(S, \alpha)$ is the clause set $S$ as "seen" under partial assignment $\alpha$

No explicit pure literal elimination (it is too slow for the benefit it provides)

trailDPLL() leads to efficient iterative DPLL implementation

**Algorithm 5:** trailDPLL(ClauseSet $S$, PartialAssignment $\alpha$)

```
while (S, α) contains a unit clause {L} do
    add {L = 1} to α                                   // Unit Propagation
if a literal is assigned both 0 and 1 in α then
    return false                                       // Conflict
if all literals assigned then
    return true                                        // Assignment found
choose a literal L not assigned in α occurring in S    // Case Splitting
if trailDPLL(S, α ∪ {{L = 1}}) then
    return true                                        // first branch
else if trailDPLL(S, α ∪ {{L = 0}}) then
    return true                                        // second branch
else return false
```

# DPLL Algorithm

Karlsruher Institut für Technologie

## Properties

- DPLL always terminates
  - Each recursion eliminates one variable
  - Worst case: binary tree search of depth $|V|$
- DPLL is sound and complete
  - If clause set $S$ is SAT, we eventually find a satisfying $\alpha$
  - If clause set $S$ is UNSAT, the entire space of (partial) variable assignments is searched (but variable selection still matters!)
- Space complexity: linear!
  — systematic search avoids blowup of “unfocused” DP

# Recap

Karlsruher Institut für Technologie

## Elementary Algorithms

- Local Search
  - Examples: GSAT, WalkSAT
  - Terminology: breakcount, makecount, diffscore
- Resolution
  - Soundness and Completeness
  - Saturation Algorithm (Exponential Complexity)
- DP Algorithm
  - Systematized Resolution
  - Improved Average Time Complexity
- DPLL Algorithm
  - Case Splitting and Unit Propagation
  - Linear Space Complexity

# Next Steps

Karlsruher Institut für Technologie

## Coming Lectures

- How can we implement unit propagation efficiently?
- Which literal $L$ to use for case splitting?
- How can we efficiently implement the case splitting step?