CS498: Algorithmic Engineering

Lecture 18: How SAT Solvers Actually Work

Elfarouk Harb

University of Illinois Urbana-Champaign

Week 10 – 03/26/2026

Outline

Where We Left Off

Lecture 17: we introduced SAT and Z3.

Where We Left Off

Lecture 17: we introduced SAT and Z3.

- Encoded puzzles (Sudoku, N-Queens) as **boolean formulas in CNF**.

Where We Left Off

Lecture 17: we introduced SAT and Z3.

- Encoded puzzles (Sudoku, N-Queens) as **boolean formulas in CNF**.
- Fed them to Z3, called `check()`, got solutions back.

Where We Left Off

Lecture 17: we introduced SAT and Z3.

- Encoded puzzles (Sudoku, N-Queens) as **boolean formulas in CNF**.
- Fed them to Z3, called `check()`, got solutions back.
- Z3 felt like magic: declare constraints, get answers.

Where We Left Off

Lecture 17: we introduced SAT and Z3.

- Encoded puzzles (Sudoku, N-Queens) as **boolean formulas in CNF**.
- Fed them to Z3, called `check()`, got solutions back.
- Z3 felt like magic: declare constraints, get answers.

Today: we open the black box. How does a SAT solver actually work?

The Brute Force Baseline

A SAT instance: n boolean variables, m clauses in CNF.

The Brute Force Baseline

A SAT instance: n boolean variables, m clauses in CNF.
Naive approach: try all 2^n assignments.

The Brute Force Baseline

A SAT instance: n boolean variables, m clauses in CNF.
Naive approach: try all 2^n assignments.

n	2^n
10	1,024
20	$\approx 10^6$
50	$\approx 10^{15}$
100	$\approx 10^{30}$
1,000	$\approx 10^{301}$

The Brute Force Baseline

A SAT instance: n boolean variables, m clauses in CNF.
Naive approach: try all 2^n assignments.

n	2^n
10	1,024
20	$\approx 10^6$
50	$\approx 10^{15}$
100	$\approx 10^{30}$
1,000	$\approx 10^{301}$

Real SAT instances have $n > 1{,}000{,}000$. Brute force is hopeless.

The Brute Force Baseline

A SAT instance: n boolean variables, m clauses in CNF.
Naive approach: try all 2^n assignments.

n	2^n
10	1,024
20	$\approx 10^6$
50	$\approx 10^{15}$
100	$\approx 10^{30}$
1,000	$\approx 10^{301}$

Real SAT instances have $n > 1{,}000{,}000$. Brute force is hopeless.
The question is: can we do better?

Starting Simple: Backtracking Search

1. Pick an unassigned variable x_i.

Starting Simple: Backtracking Search

1. Pick an unassigned variable x_i.
2. Set $x_i = \texttt{True}$. Simplify. Recurse.

Starting Simple: Backtracking Search

1. Pick an unassigned variable x_i.
2. Set $x_i = \texttt{True}$. Simplify. Recurse.
3. If SAT: done. If UNSAT: try $x_i = \texttt{False}$. Recurse.

Starting Simple: Backtracking Search

1. Pick an unassigned variable x_i.
2. Set $x_i = \texttt{True}$. Simplify. Recurse.
3. If SAT: done. If UNSAT: try $x_i = \texttt{False}$. Recurse.
4. If both fail: return UNSAT.

Starting Simple: Backtracking Search

1. Pick an unassigned variable x_i.
2. Set $x_i = \texttt{True}$. Simplify. Recurse.
3. If SAT: done. If UNSAT: try $x_i = \texttt{False}$. Recurse.
4. If both fail: return UNSAT.

"Simplify": remove satisfied clauses, drop false literals from remaining clauses.

Starting Simple: Backtracking Search

1. Pick an unassigned variable x_i.
2. Set $x_i = \texttt{True}$. Simplify. Recurse.
3. If SAT: done. If UNSAT: try $x_i = \texttt{False}$. Recurse.
4. If both fail: return UNSAT.

"Simplify": remove satisfied clauses, drop false literals from remaining clauses.
Still 2^n worst-case, but we prune early $\Rightarrow$ skip entire subtrees.

Unit Propagation: The Key Idea

Observation: if a clause has only **one unset literal** left, that literal **must** be True.

Unit Propagation: The Key Idea

Observation: if a clause has only **one unset literal** left, that literal **must** be True.

Unit Propagation (UP)

A clause with one remaining literal is a **unit clause**. Force that literal. Repeat until no more unit clauses exist.

Unit Propagation: The Key Idea

Observation: if a clause has only **one unset literal** left, that literal **must** be True.

Unit Propagation (UP)

A clause with one remaining literal is a **unit clause**. Force that literal. Repeat until no more unit clauses exist.

Example: $(\underbrace{\neg x_1}_{\text{False}} \vee \underbrace{x_2}_{\text{False}} \vee x_3) \Rightarrow$ force $x_3 = \texttt{True}$.

Unit Propagation: The Key Idea

Observation: if a clause has only **one unset literal** left, that literal **must** be True.

Unit Propagation (UP)

A clause with one remaining literal is a **unit clause**. Force that literal. Repeat until no more unit clauses exist.

Example: $(\underbrace{\neg x_1}_{\text{False}} \vee \underbrace{x_2}_{\text{False}} \vee x_3) \Rightarrow$ force $x_3 = \texttt{True}$.

Key insight: UP *cascades*. Forcing x_3 might make another clause unit, which forces another variable, and so on.

Unit Propagation: Cascade Example

$$\underbrace{(x_1 \vee x_2)}_{C_1} \wedge \underbrace{(\neg x_1 \vee x_3)}_{C_2} \wedge \underbrace{(\neg x_2 \vee x_4)}_{C_3} \wedge \underbrace{(\neg x_3 \vee \neg x_4)}_{C_4} \wedge \underbrace{(x_2 \vee \neg x_3)}_{C_5}$$

Unit Propagation: Cascade Example

$$\underbrace{(x_1 \vee x_2)}_{C_1} \wedge \underbrace{(\neg x_1 \vee x_3)}_{C_2} \wedge \underbrace{(\neg x_2 \vee x_4)}_{C_3} \wedge \underbrace{(\neg x_3 \vee \neg x_4)}_{C_4} \wedge \underbrace{(x_2 \vee \neg x_3)}_{C_5}$$

Decide: $x_1 = \texttt{False}$.

Unit Propagation: Cascade Example

$$\underbrace{(x_1 \vee x_2)}_{C_1} \wedge \underbrace{(\neg x_1 \vee x_3)}_{C_2} \wedge \underbrace{(\neg x_2 \vee x_4)}_{C_3} \wedge \underbrace{(\neg x_3 \vee \neg x_4)}_{C_4} \wedge \underbrace{(x_2 \vee \neg x_3)}_{C_5}$$

Decide: $x_1 = \texttt{False}$.

- $C_1 = (\cancel{x_1} \vee x_2)$ is unit. **Force** $x_2 = \texttt{True}$.

Unit Propagation: Cascade Example

$$\underbrace{(x_1 \vee x_2)}_{C_1} \wedge \underbrace{(\neg x_1 \vee x_3)}_{C_2} \wedge \underbrace{(\neg x_2 \vee x_4)}_{C_3} \wedge \underbrace{(\neg x_3 \vee \neg x_4)}_{C_4} \wedge \underbrace{(x_2 \vee \neg x_3)}_{C_5}$$

Decide: $x_1 = \texttt{False}$.

- $C_1 = (\cancel{x_1} \vee x_2)$ is unit. **Force** $x_2 = \texttt{True}$.
- $C_3 = (\cancel{\neg x_2} \vee x_4)$ is unit. **Force** $x_4 = \texttt{True}$.

Unit Propagation: Cascade Example

$$\underbrace{(x_1 \vee x_2)}_{C_1} \wedge \underbrace{(\neg x_1 \vee x_3)}_{C_2} \wedge \underbrace{(\neg x_2 \vee x_4)}_{C_3} \wedge \underbrace{(\neg x_3 \vee \neg x_4)}_{C_4} \wedge \underbrace{(x_2 \vee \neg x_3)}_{C_5}$$

Decide: $x_1 = \texttt{False}$.

- $C_1 = (\cancel{x_1} \vee x_2)$ is unit. **Force** $x_2 = \texttt{True}$.
- $C_3 = (\cancel{\neg x_2} \vee x_4)$ is unit. **Force** $x_4 = \texttt{True}$.
- C_2: $\neg x_1 = \text{True} \Rightarrow$ satisfied.

Unit Propagation: Cascade Example

$$\underbrace{(x_1 \vee x_2)}_{C_1} \wedge \underbrace{(\neg x_1 \vee x_3)}_{C_2} \wedge \underbrace{(\neg x_2 \vee x_4)}_{C_3} \wedge \underbrace{(\neg x_3 \vee \neg x_4)}_{C_4} \wedge \underbrace{(x_2 \vee \neg x_3)}_{C_5}$$

Decide: $x_1 = \texttt{False}$.

- $C_1 = (\cancel{x_1} \vee x_2)$ is unit. **Force** $x_2 = \texttt{True}$.
- $C_3 = (\cancel{\neg x_2} \vee x_4)$ is unit. **Force** $x_4 = \texttt{True}$.
- C_2: $\neg x_1 = \text{True} \Rightarrow$ satisfied.
- $C_4 = (\neg x_3 \vee \cancel{\neg x_4})$ is unit. **Force** $x_3 = \texttt{False}$.

Unit Propagation: Cascade Example

$$\underbrace{(x_1 \vee x_2)}_{C_1} \wedge \underbrace{(\neg x_1 \vee x_3)}_{C_2} \wedge \underbrace{(\neg x_2 \vee x_4)}_{C_3} \wedge \underbrace{(\neg x_3 \vee \neg x_4)}_{C_4} \wedge \underbrace{(x_2 \vee \neg x_3)}_{C_5}$$

Decide: $x_1 = \texttt{False}$.

- $C_1 = (\cancel{x_1} \vee x_2)$ is unit. **Force** $x_2 = \texttt{True}$.
- $C_3 = (\cancel{\neg x_2} \vee x_4)$ is unit. **Force** $x_4 = \texttt{True}$.
- C_2: $\neg x_1 = \text{True} \Rightarrow$ satisfied.
- $C_4 = (\neg x_3 \vee \cancel{\neg x_4})$ is unit. **Force** $x_3 = \texttt{False}$.

One decision triggered a cascade assigning *all* variables. SAT.

Pure Literal Elimination

Observation: if x_i only appears positively (never $\neg x_i$) across all clauses, set $x_i = \texttt{True}$. Setting it True satisfies clauses; setting it False cannot help.

Pure Literal Elimination

Observation: if x_i only appears positively (never $\neg x_i$) across all clauses, set $x_i = \texttt{True}$. Setting it True satisfies clauses; setting it False cannot help.

Less impactful than UP in practice, but cheap and sometimes helpful.

The DPLL Algorithm

Putting it all together (Davis, Putnam, Logemann, Loveland, 1962):

The DPLL Algorithm

Putting it all together (Davis, Putnam, Logemann, Loveland, 1962):

DPLL(F)

1. **Unit Propagation:** while any unit clause exists, force and simplify.
2. **Pure Literal Elimination:** assign any pure literals.
3. If no clauses left: return SAT.
4. If any clause is empty: return UNSAT.
5. Pick an unassigned variable x.
6. If DPLL($F\,|_{x=\mathrm{T}}$) = SAT: return SAT.
7. Else: return DPLL($F\,|_{x=\mathrm{F}}$).

The DPLL Algorithm

Putting it all together (Davis, Putnam, Logemann, Loveland, 1962):

DPLL(F)

1. **Unit Propagation:** while any unit clause exists, force and simplify.
2. **Pure Literal Elimination:** assign any pure literals.
3. If no clauses left: return SAT.
4. If any clause is empty: return UNSAT.
5. Pick an unassigned variable x.
6. If DPLL($F\,|_{x=\mathrm{T}}$) = SAT: return SAT.
7. Else: return DPLL($F\,|_{x=\mathrm{F}}$).

In words: propagate for free, then guess, then recurse. Dead end $\Rightarrow$ backtrack.

DPLL Walkthrough: SAT Instance

$$\underbrace{(x_1 \vee x_2)}_{C_1} \wedge \underbrace{(\neg x_1 \vee x_3)}_{C_2} \wedge \underbrace{(\neg x_2 \vee \neg x_3)}_{C_3} \wedge \underbrace{(x_2 \vee x_3)}_{C_4} \wedge \underbrace{(\neg x_1 \vee \neg x_2)}_{C_5}$$

DPLL Walkthrough: SAT Instance

$$\underbrace{(x_1 \vee x_2)}_{C_1} \wedge \underbrace{(\neg x_1 \vee x_3)}_{C_2} \wedge \underbrace{(\neg x_2 \vee \neg x_3)}_{C_3} \wedge \underbrace{(x_2 \vee x_3)}_{C_4} \wedge \underbrace{(\neg x_1 \vee \neg x_2)}_{C_5}$$

Decide: $x_1 = \texttt{True}$.

DPLL Walkthrough: SAT Instance

$$\underbrace{(x_1 \vee x_2)}_{C_1} \wedge \underbrace{(\neg x_1 \vee x_3)}_{C_2} \wedge \underbrace{(\neg x_2 \vee \neg x_3)}_{C_3} \wedge \underbrace{(x_2 \vee x_3)}_{C_4} \wedge \underbrace{(\neg x_1 \vee \neg x_2)}_{C_5}$$

Decide: $x_1 = \texttt{True}$.

- C_5: $(\cancel{\neg x_1} \vee \neg x_2) \Rightarrow (\neg x_2)$. Force $x_2 = \texttt{False}$.

DPLL Walkthrough: SAT Instance

$$\underbrace{(x_1 \vee x_2)}_{C_1} \wedge \underbrace{(\neg x_1 \vee x_3)}_{C_2} \wedge \underbrace{(\neg x_2 \vee \neg x_3)}_{C_3} \wedge \underbrace{(x_2 \vee x_3)}_{C_4} \wedge \underbrace{(\neg x_1 \vee \neg x_2)}_{C_5}$$

Decide: $x_1 = \texttt{True}$.

- C_5: $(\cancel{\neg x_1} \vee \neg x_2) \Rightarrow (\neg x_2)$. Force $x_2 = \texttt{False}$.
- C_4: $(\cancel{x_2} \vee x_3) \Rightarrow (x_3)$. Force $x_3 = \texttt{True}$.

DPLL Walkthrough: SAT Instance

$$\underbrace{(x_1 \vee x_2)}_{C_1} \wedge \underbrace{(\neg x_1 \vee x_3)}_{C_2} \wedge \underbrace{(\neg x_2 \vee \neg x_3)}_{C_3} \wedge \underbrace{(x_2 \vee x_3)}_{C_4} \wedge \underbrace{(\neg x_1 \vee \neg x_2)}_{C_5}$$

Decide: $x_1 = \texttt{True}$.

- C_5: $(\cancel{\neg x_1} \vee \neg x_2) \Rightarrow (\neg x_2)$. Force $x_2 = \texttt{False}$.
- C_4: $(\cancel{x_2} \vee x_3) \Rightarrow (x_3)$. Force $x_3 = \texttt{True}$.

Check remaining: $C_1 = (\texttt{T} \vee \texttt{F})$ ✓ $C_2 = (\texttt{F} \vee \texttt{T})$ ✓ $C_3 = (\texttt{T} \vee \texttt{F})$ ✓

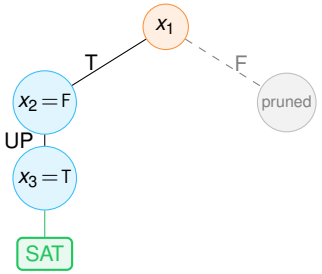
DPLL Walkthrough: SAT Instance

$$\underbrace{(x_1 \vee x_2)}_{C_1} \wedge \underbrace{(\neg x_1 \vee x_3)}_{C_2} \wedge \underbrace{(\neg x_2 \vee \neg x_3)}_{C_3} \wedge \underbrace{(x_2 \vee x_3)}_{C_4} \wedge \underbrace{(\neg x_1 \vee \neg x_2)}_{C_5}$$

Decide: $x_1 = \texttt{True}$.

- C_5: $(\cancel{\neg x_1} \vee \neg x_2) \Rightarrow (\neg x_2)$. Force $x_2 = \texttt{False}$.
- C_4: $(\cancel{x_2} \vee x_3) \Rightarrow (x_3)$. Force $x_3 = \texttt{True}$.

Check remaining: $C_1 = (\mathrm{T} \vee \mathrm{F})$ ✓ $C_2 = (\mathrm{F} \vee \mathrm{T})$ ✓ $C_3 = (\mathrm{T} \vee \mathrm{F})$ ✓



DPLL Walkthrough: UNSAT Instance

An unsatisfiable formula (4 variables, 8 clauses):

$$(x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_2 \vee x_4) \wedge (x_2 \vee \neg x_4) \wedge (\neg x_2 \vee x_4) \wedge (\neg x_2 \vee \neg x_4)$$

DPLL Walkthrough: UNSAT Instance

An unsatisfiable formula (4 variables, 8 clauses):

$$(x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_2 \vee x_4) \wedge (x_2 \vee \neg x_4) \wedge (\neg x_2 \vee x_4) \wedge (\neg x_2 \vee \neg x_4)$$

Try $x_1 = \mathsf{T}$: $C_3 \Rightarrow x_3 = \mathsf{T}$. But $C_4 = (\mathsf{F} \vee \mathsf{F})$. Conflict.

DPLL Walkthrough: UNSAT Instance

An unsatisfiable formula (4 variables, 8 clauses):

$$(x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_2 \vee x_4) \wedge (x_2 \vee \neg x_4) \wedge (\neg x_2 \vee x_4) \wedge (\neg x_2 \vee \neg x_4)$$

Try $x_1 = \mathtt{T}$: $C_3 \Rightarrow x_3 = \mathsf{T}$. But $C_4 = (\mathsf{F} \vee \mathsf{F})$. Conflict.
Backtrack, try $x_1 = \mathtt{F}$: $C_1 \Rightarrow x_2 = \mathsf{T}$. But $C_2 = (\mathsf{F} \vee \mathsf{F})$. Conflict.

DPLL Walkthrough: UNSAT Instance

An unsatisfiable formula (4 variables, 8 clauses):

$$(x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_2 \vee x_4) \wedge (x_2 \vee \neg x_4) \wedge (\neg x_2 \vee x_4) \wedge (\neg x_2 \vee \neg x_4)$$

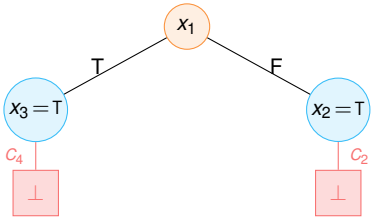
Try $x_1 = \mathtt{T}$: $C_3 \Rightarrow x_3 = \mathtt{T}$. But $C_4 = (\mathtt{F} \vee \mathtt{F})$. Conflict.
Backtrack, try $x_1 = \mathtt{F}$: $C_1 \Rightarrow x_2 = \mathtt{T}$. But $C_2 = (\mathtt{F} \vee \mathtt{F})$. Conflict.
Both branches failed $\Rightarrow$ UNSAT.

DPLL Walkthrough: UNSAT Instance

An unsatisfiable formula (4 variables, 8 clauses):

$$(x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_2 \vee x_4) \wedge (x_2 \vee \neg x_4) \wedge (\neg x_2 \vee x_4) \wedge (\neg x_2 \vee \neg x_4)$$

Try $x_1 = \mathtt{T}$: $C_3 \Rightarrow x_3 = \mathsf{T}$. But $C_4 = (\mathsf{F} \vee \mathsf{F})$. Conflict.
Backtrack, try $x_1 = \mathtt{F}$: $C_1 \Rightarrow x_2 = \mathsf{T}$. But $C_2 = (\mathsf{F} \vee \mathsf{F})$. Conflict.
Both branches failed $\Rightarrow$ UNSAT.



The Problem with Chronological Backtracking

DPLL backtracks to the **most recent decision**. This is called **chronological backtracking**.

The Problem with Chronological Backtracking

DPLL backtracks to the **most recent decision**. This is called **chronological backtracking**.

- Decide x_1 at level 1. Then $x_2, \ldots, x_9$ at levels 2–9.

The Problem with Chronological Backtracking

DPLL backtracks to the **most recent decision**. This is called **chronological backtracking**.

- Decide x_1 at level 1. Then $x_2, \ldots, x_9$ at levels 2–9.
- Conflict at level 10: caused by x_1 and x_{10}, nothing else.

The Problem with Chronological Backtracking

DPLL backtracks to the **most recent decision**. This is called **chronological backtracking**.

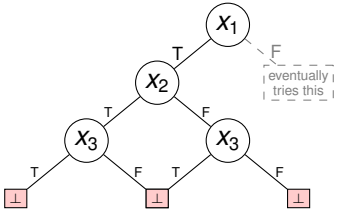
- Decide x_1 at level 1. Then $x_2, \ldots, x_9$ at levels 2–9.
- Conflict at level 10: caused by x_1 and x_{10}, nothing else.
- DPLL backtracks to level 9. But x_9 was irrelevant. Same conflict. Again at 8, 7, 6, ...

The Problem with Chronological Backtracking

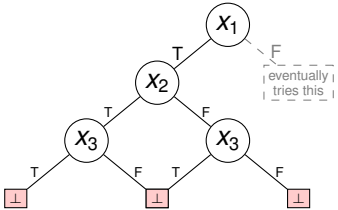
DPLL backtracks to the **most recent decision**. This is called **chronological backtracking**.

- Decide x_1 at level 1. Then $x_2, \ldots, x_9$ at levels 2–9.
- Conflict at level 10: caused by x_1 and x_{10}, nothing else.
- DPLL backtracks to level 9. But x_9 was irrelevant. Same conflict. Again at 8, 7, 6, ...
- Result: 2^8 useless combinations of $x_2, \ldots, x_9$ before reaching the real culprit.

DPLL Makes the Same Mistake Over and Over

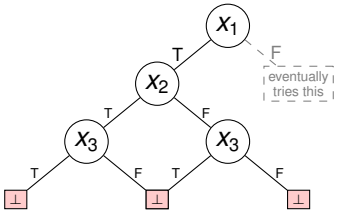


DPLL Makes the Same Mistake Over and Over



Every leaf conflict has the *same root cause*: $x_1 = \mathrm{T}$. DPLL does not know this.

DPLL Makes the Same Mistake Over and Over



Every leaf conflict has the *same root cause*: $x_1 = \mathrm{T}$. DPLL does not know this.

The gap

DPLL does not learn *why* conflicts happen. It makes the same mistake over and over.

The Question

When we hit a conflict, can we:

The Question

When we hit a conflict, can we:

1. Figure out *which decisions* caused it?

The Question

When we hit a conflict, can we:

1. Figure out *which decisions* caused it?
2. Record the mistake so we never repeat it?

The Question

When we hit a conflict, can we:

1. Figure out *which decisions* caused it?
2. Record the mistake so we never repeat it?
3. Jump directly to the guilty decision, skipping irrelevant levels?

The Question

When we hit a conflict, can we:

1. Figure out *which decisions* caused it?
2. Record the mistake so we never repeat it?
3. Jump directly to the guilty decision, skipping irrelevant levels?

This is exactly what **CDCL** (Conflict-Driven Clause Learning) does.

CDCL: Three Ideas on Top of DPLL

1. **Implication Graph.** Track *why* each variable was set.

CDCL: Three Ideas on Top of DPLL

1. **Implication Graph.** Track *why* each variable was set.
2. **Clause Learning.** When a conflict occurs, analyze the graph, derive a new clause that prevents this mistake forever.

CDCL: Three Ideas on Top of DPLL

1. **Implication Graph.** Track *why* each variable was set.
2. **Clause Learning.** When a conflict occurs, analyze the graph, derive a new clause that prevents this mistake forever.
3. **Non-chronological Backjumping.** Jump back to the *root cause*, not the most recent decision.

Decision Levels

CDCL organizes the search into **decision levels**:

Decision Levels

CDCL organizes the search into **decision levels**:

L0	No decisions yet. Propagate original unit clauses (and learned unit clauses).
L1	First decision ($x_a = \mathtt{T}$) + everything UP forces from it.
L2	Second decision ($x_b = \mathtt{F}$) + its propagations.
L3	Third decision ($x_c = \mathtt{T}$) + its propagations.

Decision Levels

CDCL organizes the search into **decision levels**:

L0	No decisions yet. Propagate original unit clauses (and learned unit clauses).
L1	First decision ($x_a = \mathtt{T}$) + everything UP forces from it.
L2	Second decision ($x_b = \mathtt{F}$) + its propagations.
L3	Third decision ($x_c = \mathtt{T}$) + its propagations.

- **Level 0** exists before any decision. It is the “ground truth.”

Decision Levels

CDCL organizes the search into **decision levels**:

L0	No decisions yet. Propagate original unit clauses (and learned unit clauses).
L1	First decision ($x_a = \mathtt{T}$) + everything UP forces from it.
L2	Second decision ($x_b = \mathtt{F}$) + its propagations.
L3	Third decision ($x_c = \mathtt{T}$) + its propagations.

- **Level 0** exists before any decision. It is the "ground truth."
- **Backjumping to level k:** undo all assignments from levels $> k$. Levels $0, \ldots, k$ stay intact.

The Implication Graph

Every time UP forces a variable, we record *why*:

The Implication Graph

Every time UP forces a variable, we record *why*:

- **Decision nodes** (orange): variables set by choice. Labeled with decision level.

The Implication Graph

Every time UP forces a variable, we record *why*:

- **Decision nodes** (orange): variables set by choice. Labeled with decision level.
- **Propagation edges** (blue): clause C_j forced variable y, so draw edges from the other literals in C_j to y.

The Implication Graph

Every time UP forces a variable, we record *why*:

- **Decision nodes** (orange): variables set by choice. Labeled with decision level.
- **Propagation edges** (blue): clause C_j forced variable y, so draw edges from the other literals in C_j to y.
- **Conflict node** ($\perp$, red): when propagation forces contradictory values.

CDCL Example: The Formula

$$C_1 = (x_1 \vee x_4)$$
$$C_2 = (x_1 \vee \neg x_3 \vee \neg x_5)$$
$$C_3 = (\neg x_1 \vee x_2)$$
$$C_4 = (\neg x_1 \vee x_3 \vee x_5)$$
$$C_5 = (\neg x_2 \vee \neg x_4)$$
$$C_6 = (\neg x_2 \vee x_3)$$
$$C_7 = (\neg x_3 \vee x_4)$$
$$C_8 = (x_2 \vee \neg x_4 \vee x_5)$$

CDCL Example: The Formula

$$C_1 = (x_1 \vee x_4) \qquad C_2 = (x_1 \vee \neg x_3 \vee \neg x_5)$$
$$C_3 = (\neg x_1 \vee x_2) \qquad C_4 = (\neg x_1 \vee x_3 \vee x_5)$$
$$C_5 = (\neg x_2 \vee \neg x_4) \qquad C_6 = (\neg x_2 \vee x_3)$$
$$C_7 = (\neg x_3 \vee x_4) \qquad C_8 = (x_2 \vee \neg x_4 \vee x_5)$$

5 variables, 8 clauses. Let's solve it with CDCL.

Step 1: Decide and Propagate

Level 1: Decide $x_1 = \mathsf{T}$.

Step 1: Decide and Propagate

Level 1: Decide $x_1 = \mathsf{T}$.
Unit propagation cascade:

1. $C_3 = (\neg x_1 \vee x_2)$: $\neg x_1 = \mathsf{F}$, unit. **Force** $x_2 = \mathsf{T}$.

Step 1: Decide and Propagate

Level 1: Decide $x_1 = \mathtt{T}$.
Unit propagation cascade:

1. $C_3 = (\neg x_1 \vee x_2)$: $\neg x_1 = \mathtt{F}$, unit. **Force** $x_2 = \mathtt{T}$.
2. $C_5 = (\neg x_2 \vee \neg x_4)$: $\neg x_2 = \mathtt{F}$, unit. **Force** $x_4 = \mathtt{F}$.

Step 1: Decide and Propagate

Level 1: Decide $x_1 = \mathtt{T}$.
Unit propagation cascade:

1. $C_3 = (\neg x_1 \vee x_2)$: $\neg x_1 = \mathtt{F}$, unit. **Force** $x_2 = \mathtt{T}$.
2. $C_5 = (\neg x_2 \vee \neg x_4)$: $\neg x_2 = \mathtt{F}$, unit. **Force** $x_4 = \mathtt{F}$.
3. $C_6 = (\neg x_2 \vee x_3)$: $\neg x_2 = \mathtt{F}$, unit. **Force** $x_3 = \mathtt{T}$.

Step 1: Decide and Propagate

Level 1: Decide $x_1 = \mathtt{T}$.
Unit propagation cascade:

1. $C_3 = (\neg x_1 \vee x_2)$: $\neg x_1 = \mathtt{F}$, unit. **Force** $x_2 = \mathtt{T}$.
2. $C_5 = (\neg x_2 \vee \neg x_4)$: $\neg x_2 = \mathtt{F}$, unit. **Force** $x_4 = \mathtt{F}$.
3. $C_6 = (\neg x_2 \vee x_3)$: $\neg x_2 = \mathtt{F}$, unit. **Force** $x_3 = \mathtt{T}$.
4. $C_7 = (\neg x_3 \vee x_4)$: $\neg x_3 = \mathtt{F}$, unit. Wants $x_4 = \mathtt{T}$.

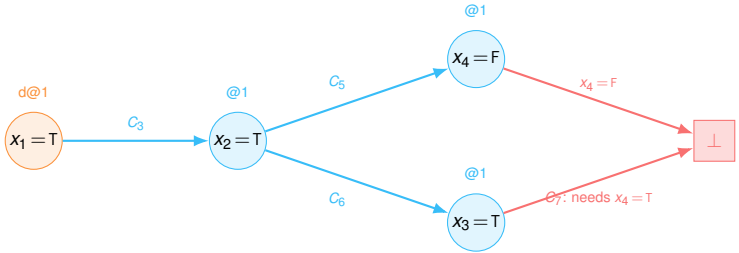
Step 1: Decide and Propagate

Level 1: Decide $x_1 = \mathtt{T}$.
Unit propagation cascade:

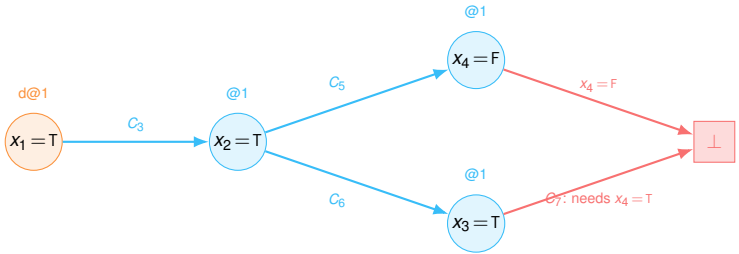
1. $C_3 = (\neg x_1 \vee x_2)$: $\neg x_1 = \mathtt{F}$, unit. **Force** $x_2 = \mathtt{T}$.
2. $C_5 = (\neg x_2 \vee \neg x_4)$: $\neg x_2 = \mathtt{F}$, unit. **Force** $x_4 = \mathtt{F}$.
3. $C_6 = (\neg x_2 \vee x_3)$: $\neg x_2 = \mathtt{F}$, unit. **Force** $x_3 = \mathtt{T}$.
4. $C_7 = (\neg x_3 \vee x_4)$: $\neg x_3 = \mathtt{F}$, unit. Wants $x_4 = \mathtt{T}$.

Conflict: step 2 forced $x_4 = \mathtt{F}$, but step 4 needs $x_4 = \mathtt{T}$.

Step 2: Draw the Implication Graph



Step 2: Draw the Implication Graph



The graph shows: x_2 is the **fork point**. It forced $x_4 = \mathsf{F}$ (via C_5) and $x_3 = \mathsf{T}$ (via C_6), and x_3 then demanded $x_4 = \mathsf{T}$ (via C_7).

Step 3: Conflict Analysis (Resolution)

We trace backwards from the conflict to find the root cause:

Step 3: Conflict Analysis (Resolution)

We trace backwards from the conflict to find the root cause:

Start: the two conflicting clauses are $C_5 = (\neg x_2 \vee \neg x_4)$ and $C_7 = (\neg x_3 \vee x_4)$.

Step 3: Conflict Analysis (Resolution)

We trace backwards from the conflict to find the root cause:

Start: the two conflicting clauses are $C_5 = (\neg x_2 \vee \neg x_4)$ and $C_7 = (\neg x_3 \vee x_4)$.

1. **Resolve** C_5 and C_7 on x_4:

$$(\neg x_2 \vee \cancel{\neg x_4}) \wedge (\neg x_3 \vee \cancel{x_4}) \vdash (\neg x_2 \vee \neg x_3)$$

Step 3: Conflict Analysis (Resolution)

We trace backwards from the conflict to find the root cause:

Start: the two conflicting clauses are $C_5 = (\neg x_2 \vee \neg x_4)$ and $C_7 = (\neg x_3 \vee x_4)$.

1. **Resolve** C_5 and C_7 on x_4:

$$(\neg x_2 \vee \cancel{\neg x_4}) \wedge (\neg x_3 \vee \cancel{x_4}) \vdash (\neg x_2 \vee \neg x_3)$$

Still 2 literals at level 1. Keep going.

Step 3: Conflict Analysis (Resolution)

We trace backwards from the conflict to find the root cause:

Start: the two conflicting clauses are $C_5 = (\neg x_2 \vee \neg x_4)$ and $C_7 = (\neg x_3 \vee x_4)$.

1. **Resolve** C_5 and C_7 on x_4:

$$(\neg x_2 \vee \cancel{\neg x_4}) \wedge (\neg x_3 \vee \cancel{x_4}) \vdash (\neg x_2 \vee \neg x_3)$$

   Still 2 literals at level 1. Keep going.

2. x_3 was forced by $C_6 = (\neg x_2 \vee x_3)$. **Resolve** on x_3:

$$(\neg x_2 \vee \cancel{\neg x_3}) \wedge (\neg x_2 \vee \cancel{x_3}) \vdash \boxed{(\neg x_2)}$$

Step 3: Conflict Analysis (Resolution)

We trace backwards from the conflict to find the root cause:

Start: the two conflicting clauses are $C_5 = (\neg x_2 \vee \neg x_4)$ and $C_7 = (\neg x_3 \vee x_4)$.

1. **Resolve** C_5 and C_7 on x_4:

$$(\neg x_2 \vee \cancel{\neg x_4}) \wedge (\neg x_3 \vee \cancel{x_4}) \vdash (\neg x_2 \vee \neg x_3)$$

   Still 2 literals at level 1. Keep going.

2. x_3 was forced by $C_6 = (\neg x_2 \vee x_3)$. **Resolve** on x_3:

$$(\neg x_2 \vee \cancel{\neg x_3}) \wedge (\neg x_2 \vee \cancel{x_3}) \vdash \boxed{(\neg x_2)}$$

One literal at level 1. We have found the **First UIP** (Unique Implication Point).

Step 3: Conflict Analysis (Resolution)

We trace backwards from the conflict to find the root cause:

Start: the two conflicting clauses are $C_5 = (\neg x_2 \vee \neg x_4)$ and $C_7 = (\neg x_3 \vee x_4)$.

1. **Resolve** C_5 and C_7 on x_4:

$$(\neg x_2 \vee \cancel{\neg x_4}) \wedge (\neg x_3 \vee \cancel{x_4}) \vdash (\neg x_2 \vee \neg x_3)$$

   Still 2 literals at level 1. Keep going.

2. x_3 was forced by $C_6 = (\neg x_2 \vee x_3)$. **Resolve** on x_3:

$$(\neg x_2 \vee \cancel{\neg x_3}) \wedge (\neg x_2 \vee \cancel{x_3}) \vdash \boxed{(\neg x_2)}$$

One literal at level 1. We have found the **First UIP** (Unique Implication Point).
Learned clause: $(\neg x_2)$. This clause is *permanently* added to the formula.

What Is the First UIP?

In the conflict analysis, we keep resolving until the clause has exactly **one literal from the current decision level**.

What Is the First UIP?

In the conflict analysis, we keep resolving until the clause has exactly **one literal from the current decision level**.

That literal is the **First UIP**: the closest “choke point” between the decision and the conflict such that ALL paths from decision to contradiction pass through it.

What Is the First UIP?

In the conflict analysis, we keep resolving until the clause has exactly **one literal from the current decision level**.

That literal is the **First UIP**: the closest “choke point” between the decision and the conflict such that ALL paths from decision to contradiction pass through it.

Why stop here? The First UIP gives the most general lesson from this conflict.

What Is the First UIP?

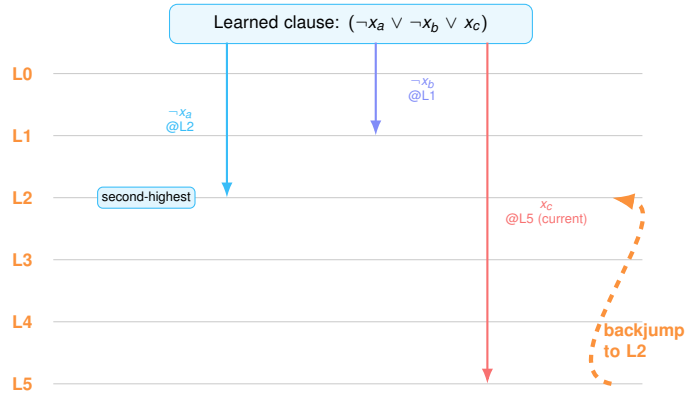
In the conflict analysis, we keep resolving until the clause has exactly **one literal from the current decision level**.

That literal is the **First UIP**: the closest “choke point” between the decision and the conflict such that ALL paths from decision to contradiction pass through it.

Why stop here? The First UIP gives the most general lesson from this conflict.

Takeaway: resolve backwards until one literal remains at the current level. That gives you the learned clause.

Where to Backjump: The General Rule



Backjump Rule

Find the **second-highest decision level** among all literals in learned clause. Backjump there. The clause is **unit** at that level, forcing the remaining literal.

Step 4: Backjump

Learned clause: $(\neg x_2)$.

Step 4: Backjump

Learned clause: $(\neg x_2)$.

DPLL would:

- Backtrack to level 1.
- Try $x_1 = \mathtt{F}$.
- No memory of why $x_1 = \mathtt{T}$ failed.

Step 4: Backjump

Learned clause: $(\neg x_2)$.

DPLL would:

- Backtrack to level 1.
- Try $x_1 = \texttt{F}$.
- No memory of why $x_1 = \texttt{T}$ failed.

CDCL does:

- **Backjump to level 0.**
- $(\neg x_2)$ becomes unit.
- Force $x_2 = \texttt{F}$.

Step 5: Solve to Completion

After backjumping to level 0, force $x_2 = \mathtt{F}$ (learned clause). Now UP cascades:

Step 5: Solve to Completion

After backjumping to level 0, force $x_2 = \texttt{F}$ (learned clause). Now UP cascades:

1. $C_3 = (\neg x_1 \vee \cancel{x_2}) \Rightarrow (\neg x_1)$. Force $x_1 = \texttt{F}$.

Step 5: Solve to Completion

After backjumping to level 0, force $x_2 = \texttt{F}$ (learned clause). Now UP cascades:

1. $C_3 = (\neg x_1 \vee \cancel{x_2}) \Rightarrow (\neg x_1)$. Force $x_1 = \texttt{F}$.
2. $C_1 = (\cancel{x_1} \vee x_4) \Rightarrow (x_4)$. Force $x_4 = \texttt{T}$.

Step 5: Solve to Completion

After backjumping to level 0, force $x_2 = \texttt{F}$ (learned clause). Now UP cascades:

1. $C_3 = (\neg x_1 \vee \cancel{x_2}) \Rightarrow (\neg x_1)$. Force $x_1 = \texttt{F}$.
2. $C_1 = (\cancel{x_1} \vee x_4) \Rightarrow (x_4)$. Force $x_4 = \texttt{T}$.
3. $C_8 = (\cancel{x_2} \vee \cancel{\neg x_4} \vee x_5) \Rightarrow (x_5)$. Force $x_5 = \texttt{T}$.

Step 5: Solve to Completion

After backjumping to level 0, force $x_2 = \texttt{F}$ (learned clause). Now UP cascades:

1. $C_3 = (\neg x_1 \vee \cancel{x_2}) \Rightarrow (\neg x_1)$. Force $x_1 = \texttt{F}$.
2. $C_1 = (\cancel{x_1} \vee x_4) \Rightarrow (x_4)$. Force $x_4 = \texttt{T}$.
3. $C_8 = (\cancel{x_2} \vee \cancel{\neg x_4} \vee x_5) \Rightarrow (x_5)$. Force $x_5 = \texttt{T}$.
4. $C_2 = (\cancel{x_1} \vee \neg x_3 \vee \cancel{\neg x_5}) \Rightarrow (\neg x_3)$. Force $x_3 = \texttt{F}$.

Step 5: Solve to Completion

After backjumping to level 0, force $x_2 = \mathtt{F}$ (learned clause). Now UP cascades:

1. $C_3 = (\neg x_1 \vee \cancel{x_2}) \Rightarrow (\neg x_1)$. Force $x_1 = \mathtt{F}$.
2. $C_1 = (\cancel{x_1} \vee x_4) \Rightarrow (x_4)$. Force $x_4 = \mathtt{T}$.
3. $C_8 = (\cancel{x_2} \vee \cancel{\neg x_4} \vee x_5) \Rightarrow (x_5)$. Force $x_5 = \mathtt{T}$.
4. $C_2 = (\cancel{x_1} \vee \neg x_3 \vee \cancel{\neg x_5}) \Rightarrow (\neg x_3)$. Force $x_3 = \mathtt{F}$.

All 8 clauses satisfied. **SAT:** $x_1 = \mathtt{F},\ x_2 = \mathtt{F},\ x_3 = \mathtt{F},\ x_4 = \mathtt{T},\ x_5 = \mathtt{T}$.

Step 5: Solve to Completion

After backjumping to level 0, force $x_2 = \mathtt{F}$ (learned clause). Now UP cascades:

1. $C_3 = (\neg x_1 \vee \cancel{x_2}) \Rightarrow (\neg x_1)$. Force $x_1 = \mathtt{F}$.
2. $C_1 = (\cancel{x_1} \vee x_4) \Rightarrow (x_4)$. Force $x_4 = \mathtt{T}$.
3. $C_8 = (\cancel{x_2} \vee \cancel{\neg x_4} \vee x_5) \Rightarrow (x_5)$. Force $x_5 = \mathtt{T}$.
4. $C_2 = (\cancel{x_1} \vee \neg x_3 \vee \cancel{\neg x_5}) \Rightarrow (\neg x_3)$. Force $x_3 = \mathtt{F}$.

All 8 clauses satisfied. **SAT:** $x_1 = \mathtt{F},\ x_2 = \mathtt{F},\ x_3 = \mathtt{F},\ x_4 = \mathtt{T},\ x_5 = \mathtt{T}$.

Observation: one conflict, one learned clause, *zero* further decisions. UP did all the work.

Why Non-Chronological Backjumping Matters

In bigger formulas, the payoff is dramatic:

Why Non-Chronological Backjumping Matters

In bigger formulas, the payoff is dramatic:

DPLL:

- Conflict at level 10.
- Backtrack: 9, 8, 7, . . .
- Waste time on irrelevant levels.

Why Non-Chronological Backjumping Matters

In bigger formulas, the payoff is dramatic:

DPLL:

- Conflict at level 10.
- Backtrack: 9, 8, 7, . . .
- Waste time on irrelevant levels.

CDCL:

- Conflict at level 10.
- Analyze: root cause is level 1.
- **Jump directly to level 1.**

Why Non-Chronological Backjumping Matters

In bigger formulas, the payoff is dramatic:

DPLL:

- Conflict at level 10.
- Backtrack: 9, 8, 7, ...
- Waste time on irrelevant levels.

CDCL:

- Conflict at level 10.
- Analyze: root cause is level 1.
- **Jump directly to level 1.**

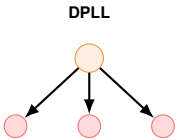
Key insight: CDCL might undo 9 decisions at once, because it *knows* the last 8 were irrelevant.

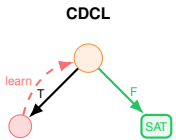
CDCL vs. DPLL: Summary

	DPLL	**CDCL**
Backtracking	Chronological	Non-chronological
Clause database	Static	Grows (learned clauses)
Conflict info	Discarded	Analyzed, recorded
Repeated mistakes	Yes	No

CDCL vs. DPLL: Summary

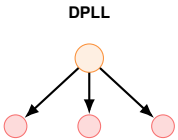
	DPLL	CDCL
Backtracking	Chronological	Non-chronological
Clause database	Static	Grows (learned clauses)
Conflict info	Discarded	Analyzed, recorded
Repeated mistakes	Yes	No

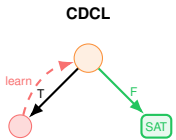




CDCL vs. DPLL: Summary

	DPLL	CDCL
Backtracking	Chronological	Non-chronological
Clause database	Static	Grows (learned clauses)
Conflict info	Discarded	Analyzed, recorded
Repeated mistakes	Yes	No





Takeaway: CDCL = DPLL + memory. It remembers its mistakes.

Two Ideas That Make Solvers Fast

CDCL gives the algorithmic framework. Two engineering tricks make it practical:

Two Ideas That Make Solvers Fast

CDCL gives the algorithmic framework. Two engineering tricks make it practical:

1. **VSIDS:** which variable to decide next.

Two Ideas That Make Solvers Fast

CDCL gives the algorithmic framework. Two engineering tricks make it practical:

1. **VSIDS:** which variable to decide next.
2. **Restarts:** when to abandon and start over.

VSIDS: Variable State Independent Decaying Sum

When CDCL needs to make a decision, which variable should it pick?

VSIDS: Variable State Independent Decaying Sum

When CDCL needs to make a decision, which variable should it pick?

- Every variable has an **activity score**.

VSIDS: Variable State Independent Decaying Sum

When CDCL needs to make a decision, which variable should it pick?

- Every variable has an **activity score**.
- When a variable appears in a **learned clause**: bump its score.

VSIDS: Variable State Independent Decaying Sum

When CDCL needs to make a decision, which variable should it pick?

- Every variable has an **activity score**.
- When a variable appears in a **learned clause**: bump its score.
- Periodically **decay** all scores ($\times 0.95$).

VSIDS: Variable State Independent Decaying Sum

When CDCL needs to make a decision, which variable should it pick?

- Every variable has an **activity score**.
- When a variable appears in a **learned clause**: bump its score.
- Periodically **decay** all scores ($\times 0.95$).
- Always decide the **highest-scoring** unassigned variable.

VSIDS: Variable State Independent Decaying Sum

When CDCL needs to make a decision, which variable should it pick?

- Every variable has an **activity score**.
- When a variable appears in a **learned clause**: bump its score.
- Periodically **decay** all scores ($\times 0.95$).
- Always decide the **highest-scoring** unassigned variable.

Key insight: variables in recent conflicts are "hot." VSIDS focuses the solver on the active part of the problem. Decay ensures old conflicts fade.

Restarts

Sometimes CDCL gets stuck in an unproductive subtree.

Restarts

Sometimes CDCL gets stuck in an unproductive subtree.

- **Restart:** abandon the search tree, start fresh.

Restarts

Sometimes CDCL gets stuck in an unproductive subtree.

- **Restart:** abandon the search tree, start fresh.
- **But keep all learned clauses.**

Restarts

Sometimes CDCL gets stuck in an unproductive subtree.

- **Restart:** abandon the search tree, start fresh.
- **But keep all learned clauses.**
- Each restart begins with a *smarter* formula.

Restarts

Sometimes CDCL gets stuck in an unproductive subtree.

- **Restart:** abandon the search tree, start fresh.
- **But keep all learned clauses.**
- Each restart begins with a *smarter* formula.
- Modern solvers restart every few hundred conflicts.

Restarts

Sometimes CDCL gets stuck in an unproductive subtree.

- **Restart:** abandon the search tree, start fresh.
- **But keep all learned clauses.**
- Each restart begins with a *smarter* formula.
- Modern solvers restart every few hundred conflicts.

Takeaway: restarts + clause learning = explore broadly without losing knowledge.

DPLL vs. CDCL: Complete Comparison

Feature	DPLL (1962)	CDCL (1996+)
Core idea	Backtracking + UP	Backtracking + UP
Conflict response	Backtrack one level	Analyze, learn, backjump
Clause database	Static	Grows
Decision heuristic	Any	VSIDS
Restarts	No	Yes
UP implementation	Scan all clauses	Uses Data-Structures (Chaff, 2001)
Practical scale	$\sim$100 variables	$\sim$1,000,000+ variables

DPLL vs. CDCL: Complete Comparison

Feature	DPLL (1962)	CDCL (1996+)
Core idea	Backtracking + UP	Backtracking + UP
Conflict response	Backtrack one level	Analyze, learn, backjump
Clause database	Static	Grows
Decision heuristic	Any	VSIDS
Restarts	No	Yes
UP implementation	Scan all clauses	Uses Data-Structures (Chaff, 2001)
Practical scale	$\sim$100 variables	$\sim$1,000,000+ variables

The deal: CDCL adds complexity, but the payoff is **four orders of magnitude** in scalability.