Satisifiability and SAT Solvers

Marcelo Finger

Department of Computer Science
Instituto de Matemática e Estatística
Universidade de São Paulo

2011

Topics

1. SAT
2. Empirical Properties
3. The DPLL Algorithm
4. Improvements to DPLL
5. Conclusion

Next Issue

The Setting: the language

- Atoms: $\mathcal{P} = \{p_1, \ldots, p_n\}$
- Literals: p_i and $\neg p_j$
- $\bar{p} = \neg p$, $\overline{\neg p} = p$
- A clause is a set of literals. Ex: $\{p, \bar{q}, r\}$ or $p \vee \neg q \vee r$
- A formula C is a set of clauses

The Setting: semantics

- Valuation for atoms $v : \mathcal{P} \to \{0, 1\}$

The Setting: semantics

- Valuation for atoms $v : \mathcal{P} \to \{0, 1\}$
- An atom p is **satisfied** if $v(p) = 1$

The Setting: semantics

- Valuation for atoms $v : \mathcal{P} \rightarrow \{0, 1\}$
- An atom p is **satisfied** if $v(p) = 1$
- Valuations are extended to all formulas

THE SETTING: SEMANTICS

- Valuation for atoms $v : \mathcal{P} \to \{0, 1\}$
- An atom p is **satisfied** if $v(p) = 1$
- Valuations are extended to all formulas
- $v(\bar{\lambda}) = 1 \Leftrightarrow v(\lambda) = 0$

The Setting: semantics

- Valuation for atoms $v : \mathcal{P} \to \{0, 1\}$
- An atom p is **satisfied** if $v(p) = 1$
- Valuations are extended to all formulas
- $v(\bar{\lambda}) = 1 \Leftrightarrow v(\lambda) = 0$
- A clause c is satisfied ($v(c) = 1$) if some literal $\lambda \in c$ is satisfied

The Setting: semantics

- Valuation for atoms $v : \mathcal{P} \to \{0, 1\}$
- An atom p is **satisfied** if $v(p) = 1$
- Valuations are extended to all formulas
- $v(\bar{\lambda}) = 1 \Leftrightarrow v(\lambda) = 0$
- A clause c is satisfied ($v(c) = 1$) if some literal $\lambda \in c$ is satisfied
- A formula C is satisfied ($v(C) = 1$) if all clauses in C are satisfied

The Problem

- A formula C is **satisfiable** if exits v, $v(C) = 1$.
- Otherwise, C is **unsatisfiable**

The Problem

- A formula C is **satisfiable** if exits v, $v(C) = 1$.
- Otherwise, C is **unsatisfiable**

The SAT Problem

Given a formula C, decide if C is satisfiable.

Witnesses: If C is satisfiable, provide a v such that $v(C) = 1$.

The Problem

- A formula C is **satisfiable** if exits v, $v(C) = 1$.
- Otherwise, C is **unsatisfiable**

The SAT Problem

Given a formula C, decide if C is satisfiable.

Witnesses: If C is satisfiable, provide a v such that $v(C) = 1$.

- SAT has small witnesses

An NP Algorithm for SAT

NP-SAT(C)

INPUT: C, a formula in clausal form

OUTPUT: v, if $v(C) = 1$; no, otherwise.

```
Guess a v
Show, in polynomial time, that v(C) = 1
return v
if no such v is guessable then
   return no
end if
```

A Naïve SAT Solver

NaiveSAT(C)

Input: C, a formula in clausal form

Output: v, if $v(C) = 1$; no, otherwise.

```
for every valuation v over p_1, ..., p_n do
    if v(C) = 1 then
        return v
    end if
end for
return no
```

Next Issue

A Brief History of SAT Solvers

- [Davis & Putnam, 1960; Davis, Longemann & Loveland, 1962] The DPLL Algorithm, a complete SAT Solver

A Brief History of SAT Solvers

- [Davis & Putnam, 1960; Davis, Longemann & Loveland, 1962] The DPLL Algorithm, a complete SAT Solver
- [Tseitin, 1966] DPLL has exponential lower bound

A Brief History of SAT Solvers

- [Davis & Putnam, 1960; Davis, Longemann & Loveland, 1962] The DPLL Algorithm, a complete SAT Solver
- [Tseitin, 1966] DPLL has exponential lower bound
- [Cook 1971] SAT is NP-complete

Incomplete SAT methods

Incomplete methods compute valuation if C is SAT; if C is unSAT, no answer.

- [Selman, Levesque & Mitchell, 1992] GSAT, a local search algorithm for SAT

Incomplete SAT methods

Incomplete methods compute valuation if C is SAT; if C is unSAT, no answer.

- [Selman, Levesque & Mitchell, 1992] GSAT, a local search algorithm for SAT
- [Mitchell, Levesque & Selman, 1992] Hard and easy SAT problems

Incomplete SAT methods

Incomplete methods compute valuation if C is SAT; if C is unSAT, no answer.

- [Selman, Levesque & Mitchell, 1992] GSAT, a local search algorithm for SAT
- [Mitchell, Levesque & Selman, 1992] Hard and easy SAT problems
- [Kautz & Selman, 1992] SAT planning

Incomplete SAT methods

Incomplete methods compute valuation if C is SAT; if C is unSAT, no answer.

- [Selman, Levesque & Mitchell, 1992] GSAT, a local search algorithm for SAT
- [Mitchell, Levesque & Selman, 1992] Hard and easy SAT problems
- [Kautz & Selman, 1992] SAT planning
- [Kautz & Selman, 1993] WalkSAT Algorithm

Incomplete SAT methods

Incomplete methods compute valuation if C is SAT; if C is unSAT, no answer.

- [Selman, Levesque & Mitchell, 1992] GSAT, a local search algorithm for SAT
- [Mitchell, Levesque & Selman, 1992] Hard and easy SAT problems
- [Kautz & Selman, 1992] SAT planning
- [Kautz & Selman, 1993] WalkSAT Algorithm
- [Gent & Walsh, 1994] SAT phase transition

DPLL: Second Generation

- Second Generation of DPLL SAT Solvers: Posit [1995], SATO [1997]. Heuristics but no learning.

DPLL: Second Generation

- Second Generation of DPLL SAT Solvers: Posit [1995], SATO [1997]. Heuristics but no learning.
- SAT competitions since 2002: `http://www.satcompetition.org/`

DPLL: Second Generation

- Second Generation of DPLL SAT Solvers: Posit [1995], SATO [1997]. Heuristics but no learning.
- SAT competitions since 2002: `http://www.satcompetition.org/`
- Aggregation of several techniques to SAT, such as learning, unlearning, backjumping, watched literal, special heuristics.

DPLL: Second Generation

- Second Generation of DPLL SAT Solvers: Posit [1995], SATO [1997]. Heuristics but no learning.
- SAT competitions since 2002: `http://www.satcompetition.org/`
- Aggregation of several techniques to SAT, such as learning, unlearning, backjumping, watched literal, special heuristics.
- Very competitive SAT solvers: GRASP [1999], Chaff [2001], BerkMin [2002], zChaff [2004], MiniSAT[2003].

DPLL: Second Generation

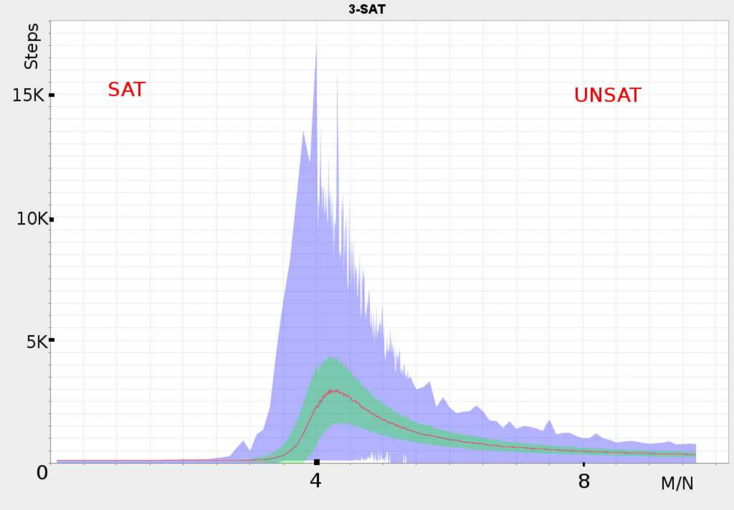
- Second Generation of DPLL SAT Solvers: Posit [1995], SATO [1997]. Heuristics but no learning.
- SAT competitions since 2002: `http://www.satcompetition.org/`
- Aggregation of several techniques to SAT, such as learning, unlearning, backjumping, watched literal, special heuristics.
- Very competitive SAT solvers: GRASP [1999], Chaff [2001], BerkMin [2002], zChaff [2004], MiniSAT[2003].
- Applications to planning, microprocessor test and verification, software design and verification, AI search, games, etc.

DPLL: Second Generation

- Second Generation of DPLL SAT Solvers: Posit [1995], SATO [1997]. Heuristics but no learning.
- SAT competitions since 2002: `http://www.satcompetition.org/`
- Aggregation of several techniques to SAT, such as learning, unlearning, backjumping, watched literal, special heuristics.
- Very competitive SAT solvers: GRASP [1999], Chaff [2001], BerkMin [2002], zChaff [2004], MiniSAT[2003].
- Applications to planning, microprocessor test and verification, software design and verification, AI search, games, etc.
- Some non-DPLL SAT solvers incorporate all those techniques: [Dixon 2004]

Next Issue

The SAT Phase Transition



The Phase Transition Diagram

- 3-SAT, N is fixed
- Higher N, more abrupt transition
- M/N: low (SAT); high (UNSAT)
- Phase transition point: $M/N \approx 4.3$, 50%SAT [Toby & Walsh 1994]
- Invariant with N
- Invariant with algorithm!

The Phase Transition Diagram

- 3-SAT, N is fixed
- Higher N, more abrupt transition
- M/N: low (SAT); high (UNSAT)
- Phase transition point: $M/N \approx 4.3$, 50%SAT [Toby & Walsh 1994]
- Invariant with N
- Invariant with algorithm!
- No theoretical explanation
- There is another phase-transition for SAT based on "Impurity" [Lozinskii 2006]

Deolalikar's $P = NP$ Proof Stratgy

- Prove theoretically the existence of phase transition
  - Uses Statistical Phisics
- Model P using Immerman-Vardi LFP-Logic
  - Show that for every p^k, some problem exists in a phase transition above p^k.

Deolalikar's $P = NP$ Proof Stratgy

- Prove theoretically the existence of phase transition
  - Uses Statistical Phisics
  - Critics found a problem here.
- Model P using Immerman-Vardi LFP-Logic
  - Show that for every p^k, some problem exists in a phase transition above p^k.
  - Critics found a problem here.
  - Uses a 2-variable fragment of LFP.

Next Issue

DPLL Through Examples

$$p \lor q$$
$$p \lor \neg q$$
$$\neg p \lor t \lor s$$
$$\neg p \lor \neg t \lor s$$
$$\neg p \lor \neg s$$
$$\neg p \lor s \lor \neg a$$

Initial Simplifications

Delete all clauses that contain λ, if $\bar{\lambda}$ does not occur.

$$p \vee q$$
$$p \vee \neg q$$
$$\neg p \vee t \vee s$$
$$\neg p \vee \neg t \vee s$$
$$\neg p \vee \neg s$$
$$\cancel{\neg p \vee s \vee \neg a}$$

Construction of a Partial Valuation

Choose a literal: s. $V = \{\mathbf{s}\}$
Propagate choice: Delete clauses containing s. Delete $\bar{s}$ from other clauses.

$$p \vee q$$
$$p \vee \neg q$$
$$\cancel{\neg p \vee t \vee s}$$
$$\cancel{\neg p \vee \neg t \vee s}$$
$$\neg p \cancel{\vee \neg s}$$

Unit Propagation

Enlarge the partial valuation with unit clauses.

$V = \{\mathbf{s}, \bar{p}\}$

Propagate unit clauses as before.

$$\cancel{p \vee} q$$

$$\cancel{p \vee} \neg q$$

$$\cancel{\neg p}$$

Another propagation step leads to $V = \{\mathbf{s}, \bar{p}, q, \bar{q}\}$

Backtracking

Unit propagation may lead to contradictory valuation:
$V = \{\mathbf{s}, \bar{p}, q, \bar{q}\}$
Backtrack to the previous choice, and propagate: $V = \{\bar{s}\}$

$$
\begin{array}{l}
p \lor q \\
p \lor \neg q \\
\neg p \lor t \cancel{\lor s} \\
\neg p \lor \neg t \cancel{\lor s} \\
\cancel{\neg p \lor \neg s}
\end{array}
$$

New Choice

When propagation finishes, a new choice is made: p.
$V = \{\bar{s}, \mathbf{p}\}$.
This leads to an inconsistent valuation: $V = \{\bar{s}, \mathbf{p}, t, \bar{t}\}$
Backtrack to last choice: $V = \{\bar{s}, \bar{p}\}$

$$\cancel{p \vee}\ q$$

$$\cancel{p \vee}\ \neg q$$

$$\cancel{\neg p \vee t}$$

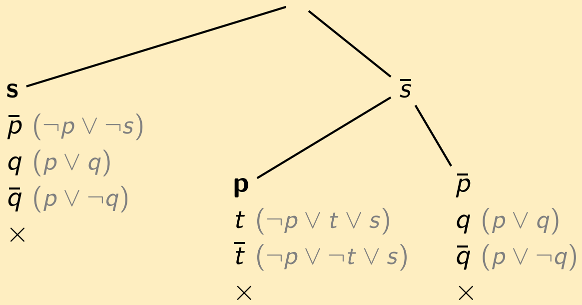
$$\cancel{\neg p \vee \neg t}$$

Propagation leads to another contradiction: $V = \{\bar{s}, \bar{p}, q, \bar{q}\}$

The Formula is UnSAT

There is nowhere to backtrack to now!

The formula is unsatisfiable, with a proof sketched below.



Next Issue

The Resolution Inference For Clauses

Usual Resolution

$$\frac{C \vee \lambda \quad \bar{\lambda} \vee D}{C \vee D}$$

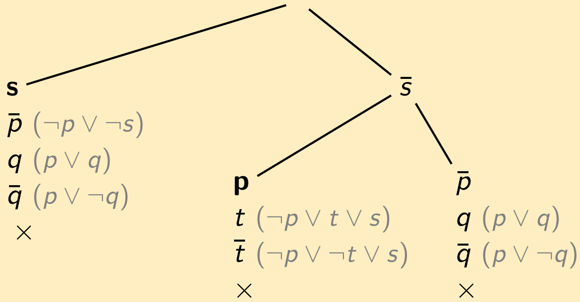
Clauses as Sets

$$\frac{\Gamma \cup \{\lambda\} \quad \{\bar{\lambda}\} \cup \Delta}{\Gamma \cup \Delta}$$

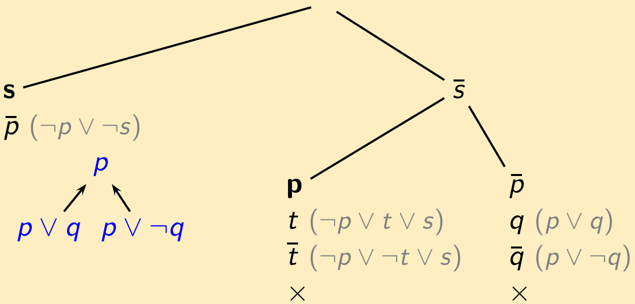
Note that, as clauses are sets

$$\frac{\Gamma \cup \{\mu, \lambda\} \quad \{\bar{\lambda}, \mu\} \cup \Delta}{\Gamma \cup \Delta \cup \{\mu\}}$$

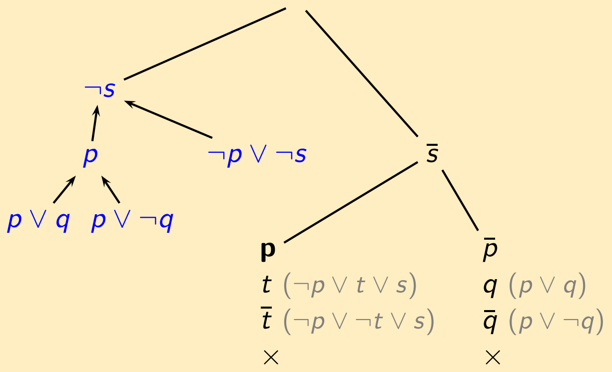
DPLL Proofs and Resolution



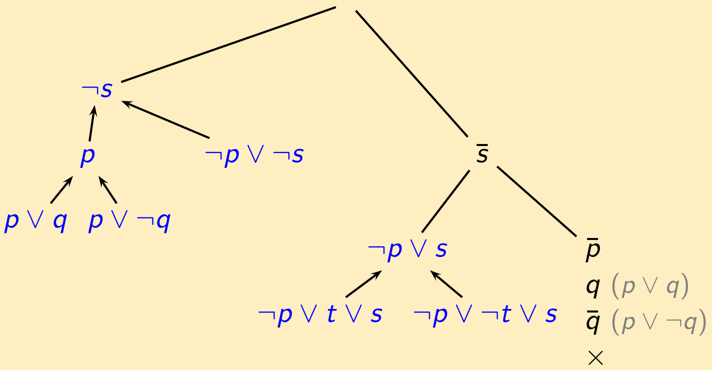
DPLL Proofs and Resolution



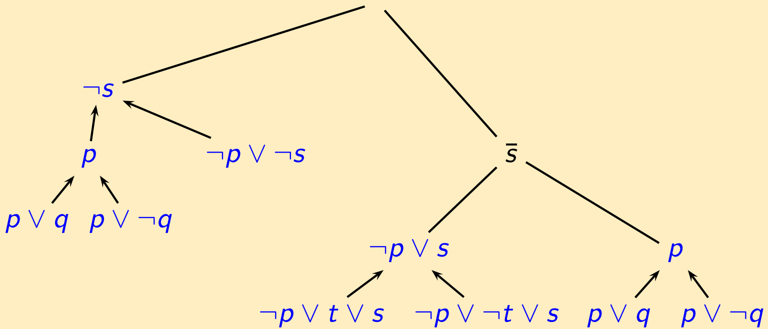
DPLL Proofs and Resolution



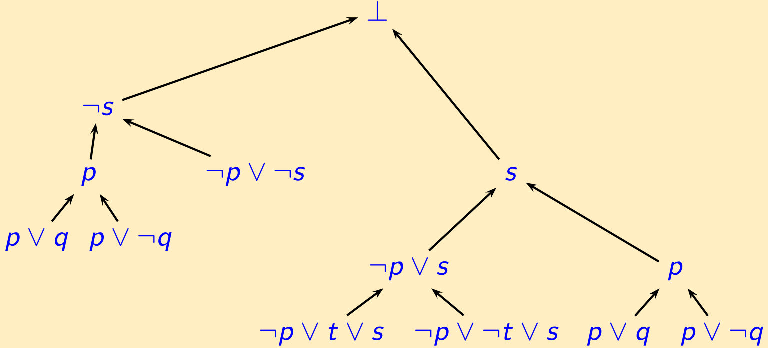
DPLL Proofs and Resolution



DPLL Proofs and Resolution



DPLL Proofs and Resolution



Conclusion

- DPLL is **isomorphic** to (a restricted form of) resolution

Conclusion

- DPLL is **isomorphic** to (a restricted form of) resolution
- DPLL inherits all properties of this (restricted form of resolution

Conclusion

- DPLL is **isomorphic** to (a restricted form of) resolution
- DPLL inherits all properties of this (restricted form of resolution
- In particular, DPLL inherits the exponential lower bounds

Enhancing DPLL

For the reasons discussed, DPLL needs to be improved to achieve better efficiency. Several techniques have been applied:

- Learning
- Unlearning
- Backjumping
- Watched literals
- Heuristics for choosing literals

Enhancing DPLL

For the reasons discussed, DPLL needs to be improved to achieve better efficiency. Several techniques have been applied:

- Learning
- Unlearning
- Backjumping
- **Watched literals**
- Heuristics for choosing literals

Next Issue

Next Issue

Watched Literals

The Cost of Unit Propagation

- Empirical measures show that **80% of time** DPLL is doing Unit Propagation
- Propagation is the main target for optimization
- Chaff introduced the technique of **Watched Literals**
  - Unit Propagation speed up
  - No need to delete literals or clauses
  - No need to watch all literals in a clause
  - Constant time backtracking (very fast)

DPLL and 3-valued Logic

- DPLL underlying logic is 3-valued
- Given a partial valuation

$$V = \{\lambda_1, \ldots, \lambda_k\}$$

- Let λ be any literal.

$$V(\lambda) = \begin{cases} 1(\text{true}) & \text{if } \lambda \in V \\ 0(\text{false}) & \text{if } \bar{\lambda} \in V \\ *(\text{undefined}) & \text{otherwise} \end{cases}$$

The Watched Literal Data Structure

- Every clause c has two selected literals: $\lambda_{c1}, \lambda_{c2}$
- For each c, $\lambda_{c1}, \lambda_{c2}$ are dynamically chosen and varies with time
- $\lambda_{c1}, \lambda_{c2}$ are **properly watched** under partial valuation V if:
  - they are both undefined; or
  - at least one of them is true

Dynamics of Watched Literals

- Initially, $V = \emptyset$
- A pair of watched literals is chosen for each clause. It is proper.
- Literal choice and unit propagation expand V
- One or both watched literals may be falsified
- If $\lambda_{c1}, \lambda_{c2}$ become improper then
  - The falsified watched literal is changed
- if no proper pair of watched literals can be found, two things may occur to alter V
  - Unit propagation (V is expanded)
  - Backtracking (V is reduced)

Example

clause	λ_{c1}	λ_{c2}
$p \vee q \vee r$	$p = *$	$q = *$
$p \vee \neg q \vee s$	$p = *$	$\bar{q} = *$
$p \vee r \vee \neg s$	$p = *$	$r = *$

Initially $V = \emptyset$

A pair of literals was elected for each clause

All are undefined, all pairs are proper

$\bar{p}$ IS CHOSEN

$V = \{\bar{\mathbf{p}}\}$

All watched literals become $(0, *)$, improper

New literals are chosen to be watched

clause	λ_{c1}	λ_{c2}
$p \vee q \vee r$	$r = *$	$q = *$
$p \vee \neg q \vee s$	$s = *$	$\bar{q} = *$
$p \vee \neg r \vee \neg s$	$\bar{s} = *$	$r = *$

$\bar{r}$ IS CHOSEN

$V = \{\bar{\mathbf{p}}, \bar{\mathbf{r}}\}$
WL in clauses 1,3 become improper
No other *- or 1-literal to be chosen
Unit propagation: $q, \bar{s}$ become true

clause	λ_{c1}	λ_{c2}
$p \vee q \vee r$	$r = 0$	$q = \cancel{*}\ 1$
$p \vee \neg q \vee s$	$s = *$	$\bar{q} = *$
$p \vee \neg r \vee \neg s$	$\bar{s} = \cancel{*}\ 1$	$r = 0$

Unit propagation leads to backtracking

$V = \{\bar{\mathbf{p}}, \bar{\mathbf{r}}, q, \bar{s}\}$
WL in clause 2 becomes improper
No other *- or 1-literal to be chosen
No unit propagation is possible: clause 2 is false

clause	λ_{c1}	λ_{c2}
$p \vee q \vee r$	$r = 0$	$q = 1$
$p \vee \neg q \vee s$	$s = 0$	$\bar{q} = 0$
$p \vee r \vee \neg s$	$\bar{s} = 1$	$r = 0$

FAST BACKTRACKING

V is contracted to last choice point

$V = \cancel{\{\bar{\mathbf{p}}, \bar{\mathbf{r}}, q, \bar{s}\}}\ \{\bar{\mathbf{p}}, r\}$

clause	λ_{c1}	λ_{c2}
$p \vee q \vee r$	$r = 1$	$q = *$
$p \vee \neg q \vee s$	$s = *$	$\bar{q} = *$
$p \vee r \vee \neg s$	$\bar{s} = *$	$r = 1$

Only affected WLs had to be recomputed

No need to reestablish previous context from a stack of contexts

Very quick backtracking

Next Issue

Improvements Techniques from Chaff

- Learning new clauses
- VSDIS Heuristics
- Random restarts
- Backjumping

Learning

When do we learn?

Sages learn from their bad choices

When do we learn?

Sages learn from their bad choices

- Every time a contradiction is derived (closed branch) we can lament our choice . . .

When do we learn?

Sages learn from their bad choices

- Every time a contradiction is derived (closed branch) we can lament our choice ...
- ... or *learn* from our mistakes:

When do we learn?

Sages learn from their bad choices

- Every time a contradiction is derived (closed branch) we can lament our choice ...
- ... or *learn* from our mistakes:
    - We learn that a choice of literals lead to contradiction

When do we learn?

Sages learn from their bad choices

- Every time a contradiction is derived (closed branch) we can lament our choice . . .
- . . . or *learn* from our mistakes:
  - We learn that a choice of literals lead to contradiction
  - We learn that the clauses involved have enough information to avoid the mistake

When do we learn?

Sages learn from their bad choices

- Every time a contradiction is derived (closed branch) we can lament our choice ...
- ... or *learn* from our mistakes:
  - We learn that a choice of literals lead to contradiction
  - We learn that the clauses involved have enough information to avoid the mistake
- We can added learned information to the problem

When do we learn?

Sages learn from their bad choices

- Every time a contradiction is derived (closed branch) we can lament our choice ...
- ... or *learn* from our mistakes:
  - We learn that a choice of literals lead to contradiction
  - We learn that the clauses involved have enough information to avoid the mistake
- We can added learned information to the problem
- **Learning means adding new clauses**, without adding new variables

Learning 1: Bad Choices

Suppose we had choices

$$V = \{\bar{\mathbf{p}}, \bar{\mathbf{r}}\}$$

which after propagation led to

$$V = \{\bar{\mathbf{p}}, \bar{\mathbf{r}}, q, \bar{s}, s\}$$

That is, in that context we learned that we cannot have both $\bar{\mathbf{p}}$ and $\bar{\mathbf{r}}$.

Learning 1: Bad Choices

Suppose we had choices

$$V = \{\bar{\mathbf{p}}, \bar{\mathbf{r}}\}$$

which after propagation led to

$$V = \{\bar{\mathbf{p}}, \bar{\mathbf{r}}, q, \bar{s}, s\}$$

That is, in that context we learned that we cannot have both $\bar{\mathbf{p}}$ and $\bar{\mathbf{r}}$.
Add the new clause:

$$p \vee r$$

Learning 1: Properties

- From a closed branch with choices $\lambda_1, \ldots, \lambda_k$, learn

$$\bar{\lambda}_1 \vee \ldots \vee \bar{\lambda}_k$$

Learning 1: Properties

- From a closed branch with choices $\lambda_1, \ldots, \lambda_k$, learn

$$\bar{\lambda}_1 \vee \ldots \vee \bar{\lambda}_k$$

- In general, this form of learning does not improve efficiency, for

Learning 1: Properties

- From a closed branch with choices $\lambda_1, \ldots, \lambda_k$, learn

$$\bar{\lambda}_1 \vee \ldots \vee \bar{\lambda}_k$$

- In general, this form of learning does not improve efficiency, for
  - it will only be used if $k - 1$ literals occur in another branch

Learning 1: Properties

- From a closed branch with choices $\lambda_1, \ldots, \lambda_k$, learn

$$\bar{\lambda}_1 \vee \ldots \vee \bar{\lambda}_k$$

- In general, this form of learning does not improve efficiency, for
  - it will only be used if $k-1$ literals occur in another branch
  - not very likely

Learning 1: Properties

- From a closed branch with choices $\lambda_1, \ldots, \lambda_k$, learn

$$\bar{\lambda}_1 \vee \ldots \vee \bar{\lambda}_k$$

- In general, this form of learning does not improve efficiency, for
  - it will only be used if $k-1$ literals occur in another branch
  - not very likely
- Useful in the presence of *forgetting* (random restarts)

Learning 1: Properties

- From a closed branch with choices $\lambda_1, \ldots, \lambda_k$, learn

$$\bar{\lambda}_1 \vee \ldots \vee \bar{\lambda}_k$$

- In general, this form of learning does not improve efficiency, for
  - it will only be used if $k - 1$ literals occur in another branch
  - not very likely
- Useful in the presence of *forgetting* (random restarts)
- Other learning techniques may be applied

Learning 2: Relevant information

- Clauses involved in a contradiction present useful information

Learning 2: Relevant information

- Clauses involved in a contradiction present useful information
- In particular, some clauses leading to a contradiction can be *resolved*

Learning 2: Relevant information

- Clauses involved in a contradiction present useful information
- In particular, some clauses leading to a contradiction can be *resolved*
- We learn the resolved clause and add it

Learning 2: Relevant information

- Clauses involved in a contradiction present useful information
- In particular, some clauses leading to a contradiction can be *resolved*
- We learn the resolved clause and add it
- For that, we have to store extra information in a partial valuation:

Learning 2: Relevant information

- Clauses involved in a contradiction present useful information
- In particular, some clauses leading to a contradiction can be *resolved*
- We learn the resolved clause and add it
- For that, we have to store extra information in a partial valuation:
  - For each unit clause, its “original” clause

Learning 2: Relevant information

- Clauses involved in a contradiction present useful information
- In particular, some clauses leading to a contradiction can be *resolved*
- We learn the resolved clause and add it
- For that, we have to store extra information in a partial valuation:
  - For each unit clause, its "original" clause
  - Choice literals are associated to $\top$

Learning 2: An Example

- In the previous example, the choices are represented as:

$$V = \{(\bar{\mathbf{p}}, \top), (\bar{\mathbf{r}}, \top)\}$$

Learning 2: An Example

- In the previous example, the choices are represented as:

$$V = \{(\bar{\mathbf{p}}, \top), (\bar{\mathbf{r}}, \top)\}$$

- After linear propagation, the contradiction

$$V = \{(\bar{\mathbf{p}}, \top), (\bar{\mathbf{r}}, \top), (q, p \vee q \vee r), (\bar{s}, p \vee r \vee \neg s), (s, p \vee \neg q \vee s)\}$$

LEARNING 2: AN EXAMPLE

- In the previous example, the choices are represented as:

$$V = \{(\bar{\mathbf{p}}, \top), (\bar{\mathbf{r}}, \top)\}$$

- After linear propagation, the contradiction

$$V = \{(\bar{\mathbf{p}}, \top), (\bar{\mathbf{r}}, \top), (q, p \vee q \vee r), (\bar{s}, p \vee r \vee \neg s), (s, p \vee \neg q \vee s)\}$$

- We learn the resolution of contradição: $p \vee r \vee \neg s$ e $p \vee \neg q \vee s$

Learning 2: An Example

- In the previous example, the choices are represented as:

$$V = \{(\bar{\mathbf{p}}, \top), (\bar{\mathbf{r}}, \top)\}$$

- After linear propagation, the contradiction

$$V = \{(\bar{\mathbf{p}}, \top), (\bar{\mathbf{r}}, \top), (q, p \vee q \vee r), (\bar{s}, p \vee r \vee \neg s), (s, p \vee \neg q \vee s)\}$$

- We learn the resolution of contradição: $p \vee r \vee \neg s$ e $p \vee \neg q \vee s$
- That is, we add the clause:

$$p \vee r \vee \neg q$$

Learning 2: An Example

- In the previous example, the choices are represented as:

$$V = \{(\bar{\mathbf{p}}, \top), (\bar{\mathbf{r}}, \top)\}$$

- After linear propagation, the contradiction

$$V = \{(\bar{\mathbf{p}}, \top), (\bar{\mathbf{r}}, \top), (q, p \vee q \vee r), (\bar{s}, p \vee r \vee \neg s), (s, p \vee \neg q \vee s)\}$$

- We learn the resolution of contradição: $p \vee r \vee \neg s$ e $p \vee \neg q \vee s$
- That is, we add the clause:

$$p \vee r \vee \neg q$$

- This form of learning has better effects to proof efficiency

Learning 2: An Example

- In the previous example, the choices are represented as:

$$V = \{(\bar{\mathbf{p}}, \top), (\bar{\mathbf{r}}, \top)\}$$

- After linear propagation, the contradiction

$$V = \{(\bar{\mathbf{p}}, \top), (\bar{\mathbf{r}}, \top), (q, p \vee q \vee r), (\bar{s}, p \vee r \vee \neg s), (s, p \vee \neg q \vee s)\}$$

- We learn the resolution of contradição: $p \vee r \vee \neg s$ e $p \vee \neg q \vee s$
- That is, we add the clause:

$$p \vee r \vee \neg q$$

- This form of learning has better effects to proof efficiency
- Theorem: proofs with DPLL + Learning2 can polynomially simulate full resolution proofs

Heuristics for Choosing Literals

Choosing Literals

- A DPLL step starts when linear propagation ends without contradiction

Choosing Literals

- A DPLL step starts when linear propagation ends without contradiction
- A each step, a new literal has to be chosen to start a new cycle propagation/backtrack

Choosing Literals

- A DPLL step starts when linear propagation ends without contradiction
- A each step, a new literal has to be chosen to start a new cycle propagation/backtrack
- Several **heuristics** can be used to choose that literal

Choosing Literals

- A DPLL step starts when linear propagation ends without contradiction
- A each step, a new literal has to be chosen to start a new cycle propagation/backtrack
- Several **heuristics** can be used to choose that literal
- Heuristics are rules that help the choice of one among several possibilities

Choosing Literals

- A DPLL step starts when linear propagation ends without contradiction
- A each step, a new literal has to be chosen to start a new cycle propagation/backtrack
- Several **heuristics** can be used to choose that literal
- Heuristics are rules that help the choice of one among several possibilities
- Some heuristics are clearly more efficient than others

Heuristics without Learning

- Do not take in consideration the learning of new clauses

Heuristics without Learning

- Do not take in consideration the learning of new clauses
  - **MOM Heuristics — M**aximum number of literal **O**corrences with **M**inimum length
    Easy to implement: select the literal that is present in the largest number of clauses of minimum size. Increases the probability of backtracking

Heuristics without Learning

- Do not take in consideration the learning of new clauses
  - **MOM Heuristics — M**aximum number of literal **O**corrences with **M**inimum length
    Easy to implement: select the literal that is present in the largest number of clauses of minimum size. Increases the probability of backtracking
  - **SATO Heuristics** is a variation of MOM. Let $f(p)$ be 1 plus the number of clauses of smallest size which contain p. Choose p that maximizes $f(p) * f(\neg p)$. Choose p if $f(p) > f(\neg p)$; $\neg p$ otherwise.

Heuristics VSIDS

- *Variable State Independent Decaying Sum*

Heuristics VSIDS

- *Variable State Independent Decaying Sum*
- Takes the learning of clauses in consideration

Heuristics VSIDS

- *Variable State Independent Decaying Sum*
- Takes the learning of clauses in consideration
- One counter for each literal

Heuristics VSIDS

- *Variable State Independent Decaying Sum*
- Takes the learning of clauses in consideration
- One counter for each literal
- Initialize each with n. of occurrences of each literal

Heuristics VSIDS

- *Variable State Independent Decaying Sum*
- Takes the learning of clauses in consideration
- One counter for each literal
- Initialize each with n. of occurrences of each literal
- Increment counter when a clause containing that literal is added

Heuristics VSIDS

- *Variable State Independent Decaying Sum*
- Takes the learning of clauses in consideration
- One counter for each literal
- Initialize each with n. of occurrences of each literal
- Increment counter when a clause containing that literal is added
- Choose literal with highest count

Heuristics VSIDS

- *Variable State Independent Decaying Sum*
- Takes the learning of clauses in consideration
- One counter for each literal
- Initialize each with n. of occurrences of each literal
- Increment counter when a clause containing that literal is added
- Choose literal with highest count
- Periodically, counters are divided by a constant

HEURISTICS VSIDS

- *Variable State Independent Decaying Sum*
- Takes the learning of clauses in consideration
- One counter for each literal
- Initialize each with n. of occurrences of each literal
- Increment counter when a clause containing that literal is added
- Choose literal with highest count
- Periodically, counters are divided by a constant
- Highest priority to literals in clauses recently learned

Heuristics VSIDS

- *Variable State Independent Decaying Sum*
- Takes the learning of clauses in consideration
- One counter for each literal
- Initialize each with n. of occurrences of each literal
- Increment counter when a clause containing that literal is added
- Choose literal with highest count
- Periodically, counters are divided by a constant
- Highest priority to literals in clauses recently learned
- Low overheads: counters updated only during learning

Random Restarts

Forgetting, or Random Restarts

- Suppose a formula is SAT, but bad initial choices

Forgetting, or Random Restarts

- Suppose a formula is SAT, but bad initial choices
- A valuation will be found only after exhausting the consequences of those bad choices

Forgetting, or Random Restarts

- Suppose a formula is SAT, but bad initial choices
- A valuation will be found only after exhausting the consequences of those bad choices
- Idea: restart periodically the search for a valuation

Forgetting, or Random Restarts

- Suppose a formula is SAT, but bad initial choices
- A valuation will be found only after exhausting the consequences of those bad choices
- Idea: restart periodically the search for a valuation
- Problem: if the formula is UNSAT, this may lead to great inefficiency

Forgetting, or Random Restarts

- Suppose a formula is SAT, but bad initial choices
- A valuation will be found only after exhausting the consequences of those bad choices
- Idea: restart periodically the search for a valuation
- Problem: if the formula is UNSAT, this may lead to great inefficiency
- This problem is avoided if learned formulas are kept

Random Restart

- Consider ϵ, $0 < \epsilon \ll 1$

Random Restart

- Consider ϵ, $0 < \epsilon \ll 1$
- When a branch is closed

Random Restart

- Consider ϵ, $0 < \epsilon \ll 1$
- When a branch is closed
  - With probability $(1 - \epsilon)$, perform usual backtracking

Random Restart

- Consider ϵ, $0 < \epsilon \ll 1$
- When a branch is closed
  - With probability $(1 - \epsilon)$, perform usual backtracking
  - With probability ϵ, restart the search process with $V = \emptyset$

Random Restart

- Consider ϵ, $0 < \epsilon \ll 1$
- When a branch is closed
  - With probability $(1 - \epsilon)$, perform usual backtracking
  - With probability ϵ, restart the search process with $V = \emptyset$
- Empirically checked: this brings efficiency gains

Backjumping

Backjumping via Examples

- Suppose we have the partial valuation
$\{(\bar{\mathbf{p}}, \top), (\bar{\mathbf{r}}, \top), (\mathbf{a}, \top), (q, p \lor q \lor r), (\bar{s}, p \lor r \lor \neg s), (s, p \lor \neg q \lor s)\}$

Backjumping via Examples

- Suppose we have the partial valuation $\{(\bar{\mathbf{p}}, \top), (\bar{\mathbf{r}}, \top), (\mathbf{a}, \top), (q, p \vee q \vee r), (\bar{s}, p \vee r \vee \neg s), (s, p \vee \neg q \vee s)\}$
- The chosen literal **a** does not occur in clauses associated with subsequent unit propagations

Backjumping via Examples

- Suppose we have the partial valuation $\{(\bar{\mathbf{p}}, \top), (\bar{\mathbf{r}}, \top), (\mathbf{a}, \top), (q, p \vee q \vee r), (\bar{s}, p \vee r \vee \neg s), (s, p \vee \neg q \vee s)\}$
- The chosen literal **a** does not occur in clauses associated with subsequent unit propagations
- That is, the choice of **a** is useless

Backjumping via Examples

- Suppose we have the partial valuation $\{(\bar{\mathbf{p}}, \top), (\bar{\mathbf{r}}, \top), (\mathbf{a}, \top), (q, p \vee q \vee r), (\bar{s}, p \vee r \vee \neg s), (s, p \vee \neg q \vee s)\}$
- The chosen literal **a** does not occur in clauses associated with subsequent unit propagations
- That is, the choice of **a** is useless
- Backtracking would lead to useless repetition $\{(\bar{\mathbf{p}}, \top), (\bar{\mathbf{r}}, \top), (\bar{a}, \top), (q, p \vee q \vee r), (\bar{s}, p \vee r \vee \neg s), (s, p \vee \neg q \vee s)\}$

Backjumping via Examples

- Suppose we have the partial valuation $\{(\bar{\mathbf{p}}, \top), (\bar{\mathbf{r}}, \top), (\mathbf{a}, \top), (q, p \lor q \lor r), (\bar{s}, p \lor r \lor \neg s), (s, p \lor \neg q \lor s)\}$
- The chosen literal **a** does not occur in clauses associated with subsequent unit propagations
- That is, the choice of **a** is useless
- Backtracking would lead to useless repetition $\{(\bar{\mathbf{p}}, \top), (\bar{\mathbf{r}}, \top), (\bar{a}, \top), (q, p \lor q \lor r), (\bar{s}, p \lor r \lor \neg s), (s, p \lor \neg q \lor s)\}$
- Backtracking should "jump back" over **a** and ignore it, instead of going to $\{(\bar{\mathbf{p}}, \top), (\bar{\mathbf{r}}, \top), (\bar{\mathbf{a}}, \top)$, goes to $\{(\bar{\mathbf{p}}, \top), (\mathbf{r}, \top)\}$

Backjumping via Examples

- Suppose we have the partial valuation
  $\{(\bar{\mathbf{p}}_1, \top), (\bar{\mathbf{r}}_2, \top), (\mathbf{a}_3, \top), (q_2, p \vee q \vee r), (\bar{s}_2, p \vee r \vee \neg s), (s_2, p \vee \neg q \vee s)\}$
- The chosen literal **a** does not occur in clauses associated with subsequent unit propagations
- That is, the choice of **a** is useless
- Backtracking would lead to useless repetition
  $\{(\bar{\mathbf{p}}, \top), (\bar{\mathbf{r}}, \top), (\bar{a}, \top), (q, p \vee q \vee r), (\bar{s}, p \vee r \vee \neg s), (s, p \vee \neg q \vee s)\}$
- Backtracking should "jump back" over **a** and ignore it, instead of going to $\{(\bar{\mathbf{p}}, \top), (\bar{\mathbf{r}}, \top), (\bar{\mathbf{a}}, \top)$, goes to $\{(\bar{\mathbf{p}}, \top), (\mathbf{r}, \top)\}$
- This is backjumping

BACKJUMPING VIA EXAMPLES

- Suppose we have the partial valuation
  $\{(\bar{\mathbf{p}}_1, \top), (\bar{\mathbf{r}}_2, \top), (\mathbf{a}_3, \top), (q_2, p \vee q \vee r), (\bar{s}_2, p \vee r \vee \neg s), (s_2, p \vee \neg q \vee s)\}$
- The chosen literal **a** does not occur in clauses associated with subsequent unit propagations
- That is, the choice of **a** is useless
- Backtracking would lead to useless repetition
  $\{(\bar{\mathbf{p}}, \top), (\bar{\mathbf{r}}, \top), (\bar{a}, \top), (q, p \vee q \vee r), (\bar{s}, p \vee r \vee \neg s), (s, p \vee \neg q \vee s)\}$
- Backtracking should "jump back" over **a** and ignore it, instead of going to $\{(\bar{\mathbf{p}}, \top), (\bar{\mathbf{r}}, \top), (\bar{\mathbf{a}}, \top)$, goes to $\{(\bar{\mathbf{p}}, \top), (\mathbf{r}, \top)\}$
- This is backjumping
- Backjumping always brings efficiency gains

Next Issue

Conclusion of the Talk

- DPLL is $>$ 40 years old, but still the most used strategy for SAT solvers

Conclusion of the Talk

- DPLL is > 40 years old, but still the most used strategy for SAT solvers
- Use of smart techniques have improved DPLL's performance: $N = 15 \longrightarrow N = 100\,000$

Conclusion of the Talk

- DPLL is > 40 years old, but still the most used strategy for SAT solvers
- Use of smart techniques have improved DPLL's performance: $N = 15 \longrightarrow N = 100\,000$
- There are still very hard formulas that make DPLL exponential, e.g. PHP

Conclusion of the Talk

- DPLL is $>$ 40 years old, but still the most used strategy for SAT solvers
- Use of smart techniques have improved DPLL's performance: $N = 15 \longrightarrow N = 100\,000$
- There are still very hard formulas that make DPLL exponential, e.g. PHP
- Experiments show that these formulas do occur in practice

Conclusion of the Talk

- DPLL is >40 years old, but still the most used strategy for SAT solvers
- Use of smart techniques have improved DPLL's performance: $N=15 \longrightarrow N=100\,000$
- There are still very hard formulas that make DPLL exponential, e.g. PHP
- Experiments show that these formulas do occur in practice
- The future of SAT solvers may be in non-DPLL, non-clausal methods

Conclusion of the Talk

- DPLL is > 40 years old, but still the most used strategy for SAT solvers
- Use of smart techniques have improved DPLL's performance: $N = 15 \longrightarrow N = 100\,000$
- There are still very hard formulas that make DPLL exponential, e.g. PHP
- Experiments show that these formulas do occur in practice
- The future of SAT solvers may be in non-DPLL, non-clausal methods
- But the techniques learned from DPLL are incorporated in new techniques

Conclusion of the Talk

- DPLL is >40 years old, but still the most used strategy for SAT solvers
- Use of smart techniques have improved DPLL's performance: $N = 15 \longrightarrow N = 100\,000$
- There are still very hard formulas that make DPLL exponential, e.g. PHP
- Experiments show that these formulas do occur in practice
- The future of SAT solvers may be in non-DPLL, non-clausal methods
- But the techniques learned from DPLL are incorporated in new techniques
- SAT can also be enhanced: **SAT Modulo Theories**

SAT is not a Panacea

- Reduction to SAT may be ok for some NPc problems, but ...
  - ... some NPc problems with **no known** polynomial SAT-reduction.

    E.g. Answer Set Programming
  - ... some NPc problems with **no efficient** polynomial SAT-reduction.

    E.g. Probabilistic SAT