SAT in AI: high performance search methods with applications

Jussi Rintanen
Department of Information and Computer Science
Aalto University
Helsinki, Finland

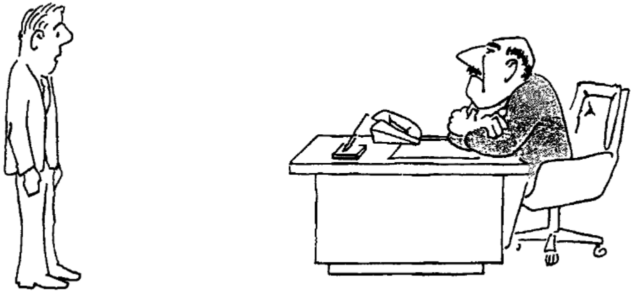
IJCAI 2013, Beijing

SAT and NP-complete problems in Artificial Intelligence

- Earlier, NP-complete problems were considered practically unsolvable, except in simplest instances.
- Breakthroughs in SAT solving from mid-1990's on.
- Leading to breakthroughs in state space search (with applications in construction of intelligent systems.)
- Starting to have impact in other areas, including probabilistic reasoning and machine learning.

Why you need*ed* to know about NP-hardness

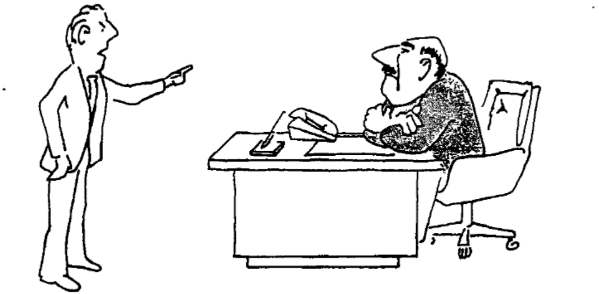
Garey & Johnson, *Computers and Intractability*, 1979



“I can’t find an efficient algorithm, I guess I’m just too dumb.”

Why you need*ed* to know about NP-hardness

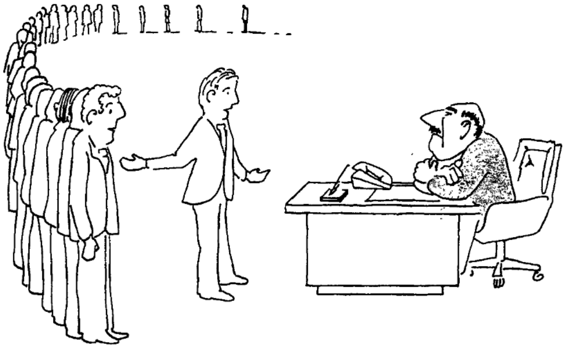
Garey & Johnson, *Computers and Intractability*, 1979



“I can’t find an efficient algorithm, because no such algorithm is possible!”

Why you needed *ed* to know about NP-hardness

Garey & Johnson, *Computers and Intractability*, 1979



"I can't find an efficient algorithm, but neither can all these famous people."

NP-completeness has changed

- Earlier: “It is NP-complete, don’t bother trying to solve it.”
- Now: “It is NP-complete, you might well solve it.”
- SAT now has several industrial applications, and more are emerging.
- Extensions of SAT are a topic of intense research in automated reasoning and AI.
- Many important problems in AI and CS are NP-complete:
  - Combinatorics of the real world (too many options to do things).
  - How to do something optimally?

Applications of SAT in Computer Science

- reachability problems
  - model-checking in Computer Aided Verification [BCCZ99] of sequential circuits and software
  - planning in Artificial Intelligence [KS92, KS96]
  - discrete event systems diagnosis [GARK07]
- integrated circuits
  - automatic test pattern generation (ATPG) [Lar92]
  - equivalence checking [KPKG02, CGL^{+}10, WGMD09]
  - logic synthesis [KKY04]
  - fault diagnosis [SVFAV05]
- biology and language
  - haplotype inference [LMS06]
  - computing evolutionary tree measures [BSJ09]
  - construction of phylogenetic trees [BEE^{+}07]

Classification of Problems by Complexity

problem		class	search space
SAT	find a solution	NP	trees
SMT	find a solution	NP	
MAX-SAT	find best solution	FP^{NP}	
#SAT	how many solutions?	#P, PP	
SSAT	$\exists - \forall - R$ alternation	PSPACE	and-or trees
QBF	$\exists - \forall$ alternation	PSPACE	

Differences in NP-hardness

Most scalable methods are for satisfiable instances of SAT (NP). These can be solved because of good heuristics: solvers are successfully guessing their way through an exponentially large search space.

Currently, the same does not (as often) hold for

- unsatisfiable instances: determining that no solutions exist
- optimization: finding best solutions
- problems involving counting models, e.g. probabilistic questions
- problems involving alternation $\sim$ and-or trees

Progress in these questions is made, but it has been slower.
NP substantially easier than co-NP, #P, FP^{NP}, ...

Propositional logic
Syntax

Let X be a set of atomic propositions.

1. $\bot$ and $\top$ are formulae.
2. x is a formula for all $x \in X$.
3. $\neg\phi$ is a formula if ϕ is.
4. $\phi \vee \phi'$ and $\phi \wedge \phi'$ are formulae if ϕ and ϕ' are.

$\phi \rightarrow \phi'$ is an abbreviation for $\neg\phi \vee \phi'$.
$\phi \leftrightarrow \phi'$ is an abbreviation for $(\phi \rightarrow \phi') \wedge (\phi' \rightarrow \phi)$.

For literals $l \in X \cup \{\neg x | x \in X\}$, complement $\bar{l}$ is defined by $\overline{x} = \neg x$ and $\overline{\neg x} = x$.

A clause is a disjunction of literals $l_1 \vee \cdots \vee l_n$.

Propositional logic
Valuations and truth

Define truth with respect to a valuation $v : X \to \{0, 1\}$:

1. $v \models \top$
2. $v \not\models \bot$
3. $v \models x$ if and only if $v(x) = 1$, for all $x \in X$.
4. $v \models \neg\phi$ if and only if $v \not\models \phi$.
5. $v \models \phi \vee \phi'$ if and only if $v \models \phi$ or $v \models \phi'$.
6. $v \models \phi \wedge \phi'$ if and only if $v \models \phi$ and $v \models \phi'$.

Define for sets C of formulas, $v \models C$ iff $v \models \phi$ for all $\phi \in C$.

The SAT decision problem

SAT

Let X be a set of propositional variables. Let $\mathcal{F}$ be a set of clauses over X.
$\mathcal{F} \in$ SAT iff there is $v : X \to \{0, 1\}$ such that $v \models \mathcal{F}$.

UNSAT

Let X be a set of propositional variables. Let $\mathcal{F}$ be a set of clauses over X.
$\mathcal{F} \in$ UNSAT iff $v \not\models \mathcal{F}$ for all $v : X \to \{0, 1\}$.

Complexity class NP

- NP = decision problems solvable by nondeterministic Turing Machines with a polynomial bound on the number of computation steps.
- This is roughly: search problems with a search tree (OR tree) of polynomial depth.
- SAT is in NP because
  1. a valuation v of X can be guessed in $|X|$ steps, and
  2. testing $v \models \mathcal{F}$ is polynomial time in the size of $\mathcal{F}$.

NP-hardness of SAT
(Cook, *The Complexity of Theorem Proving Procedures*, 1971)

- Cook showed that the halting problem of any nondeterministic Turing machine with a polynomial time bound can be reduced to SAT [Coo71]. Idea:
  - TM configuration $\sim$ a valuation of propositional variables
  - sequence of configurations $\sim$ sequence of valuations
  - relations between consecutive configurations $\sim$ propositional formula
  - initial and accepting configurations $\sim$ propositional formula
  - accepting computation $\sim$ valuation that makes the formula true
- The proof is similar to the reduction from AI planning to SAT! We will discuss the topic in detail later.

Significance of NP-completeness

- No NP-complete problem is known to have a polynomial time algorithm.
- Best algorithms have a worst-case exponential runtime.

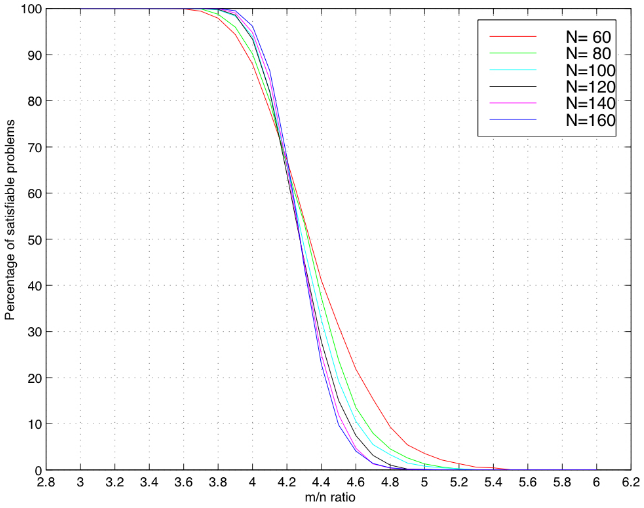
  $2^{0.30897m}$, $2^{0.10299L}$ [Hir00]

  $(2 - \frac{2}{k+1})^n$ [DGH+02]

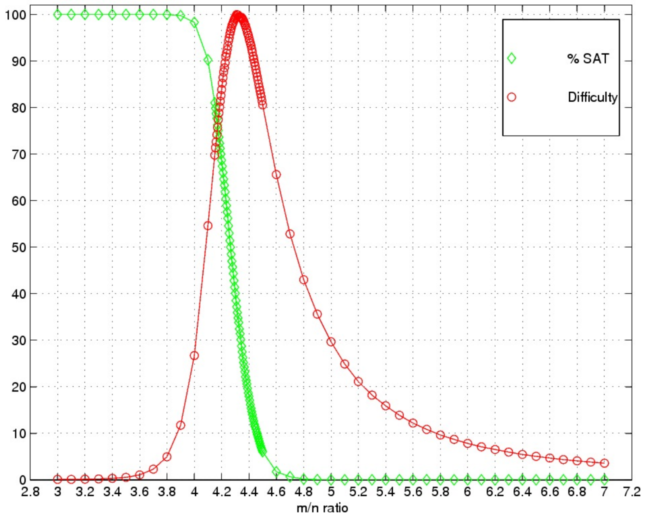
  $2^{n(1-\frac{1}{\ln \frac{m}{n} + O(\ln \ln m)})}$ [DHW05]

  (m clauses of length $\leq k$, n variables, size L).
- However, worst-case doesn't always show up!
- Current SAT algorithms can solve problem instances with millions of clauses and hundreds of thousands of variables in seconds.

Phase transitions
phase transition from SAT to UNSAT in 3-SAT

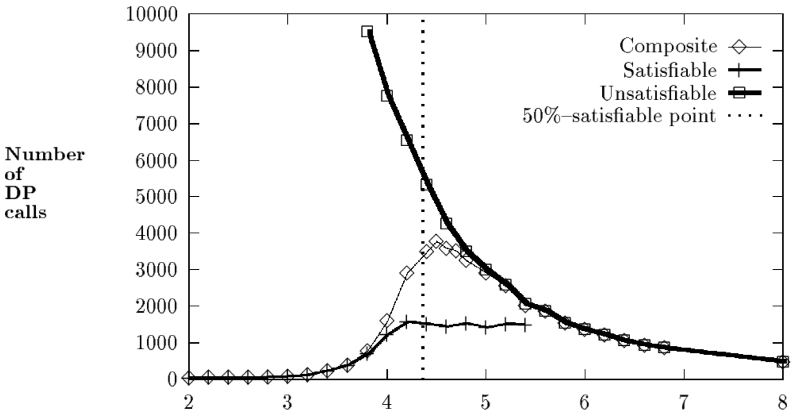


Phase transitions
Problem difficulty in the phase transition area



Phase transitions

Problem difficulty separately for SAT and UNSAT



Meaning of phase transitions

Even though all known complete algorithms have an exponential runtime in the worst case, their scalability on under-constrained and over-constrained problem instances is often much much better.

Other hard problems have similar phase transitions: keep problem size constant, and vary one of the parameters.

- scheduling: few..many tasks, a lot of..little time
- diagnosis: few..many observations
- planning, model-checking: many..few transitions

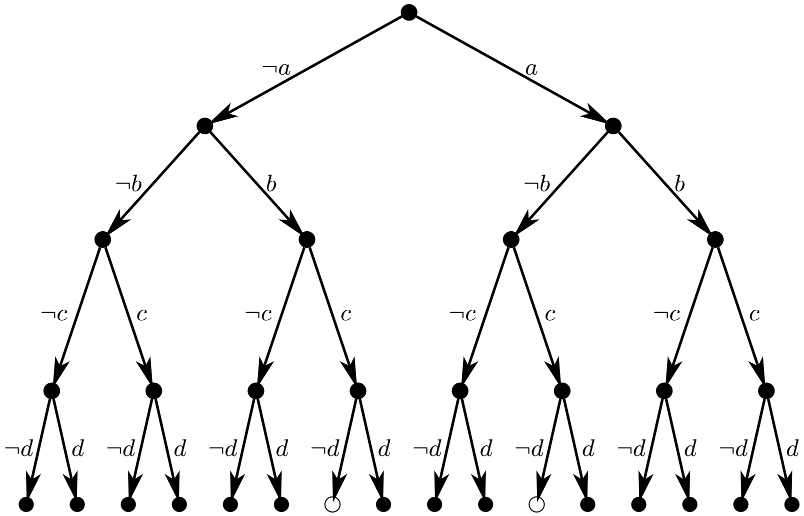
Truth-tables

Truth table for
$\phi = (a \leftrightarrow b) \vee (c \rightarrow d)$:

v				$v(\phi)$
a	b	c	d	
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Truth-tables vs binary search trees

Binary search tree for $\phi = (a \leftrightarrow b) \vee (c \rightarrow d)$:



The Resolution Rule

Resolution

$$\frac{l \vee \phi \qquad \bar{l} \vee \phi'}{\phi \vee \phi'}$$

One of l and $\bar{l}$ is false.
Hence at least one of ϕ and ϕ' is true.

Unit Resolution

- Unrestricted application of the resolution rule is too expensive.
- Unit resolution restricts one of the clauses to be a unit clause consisting of only one literal.
- Performing all possible unit resolution steps on a clause set can be done in linear time [DG84], and there are very efficient implementations [MMZ^{+}01].

Unit Propagation

Unit Resolution

$$\frac{l \qquad \bar{l} \vee \phi}{\phi}$$

Unit Propagation algorithm UNIT($\mathcal{F}$) for clause sets $\mathcal{F}$

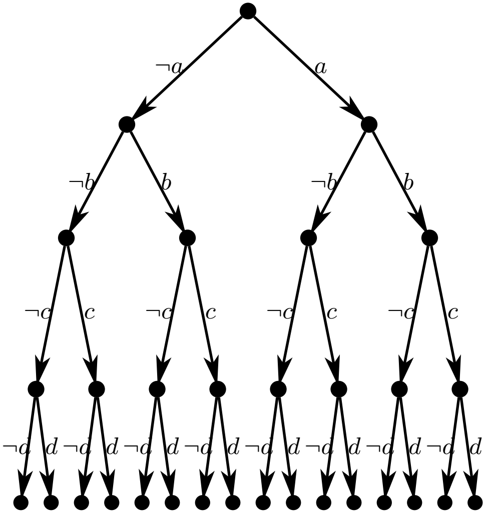
1. If there is a unit clause $l \in \mathcal{F}$, then replace every $\bar{l} \vee \phi \in \mathcal{F}$ by ϕ and remove all clauses containing l from $\mathcal{F}$.
   As a special case the empty clause $\perp$ may be obtained.
2. If $\mathcal{F}$ still contains a unit clause, repeat step 1.
3. Return $\mathcal{F}$.

We sometimes write $\mathcal{F} \vdash_{UP} l$ if $l \in UP(\mathcal{F})$.

Binary search with unit resolution

The Davis-Putnam-Logemann-Loveland procedure DPLL [DLL62]

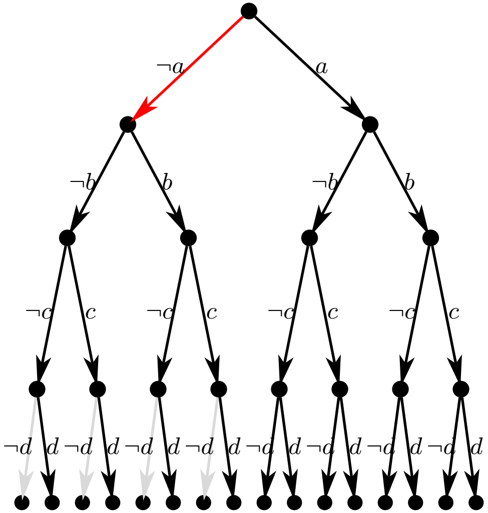
$a \vee d$
$b \vee c$



Binary search with unit resolution

The Davis-Putnam-Logemann-Loveland procedure DPLL [DLL62]

$a \vee d$
$b \vee c$

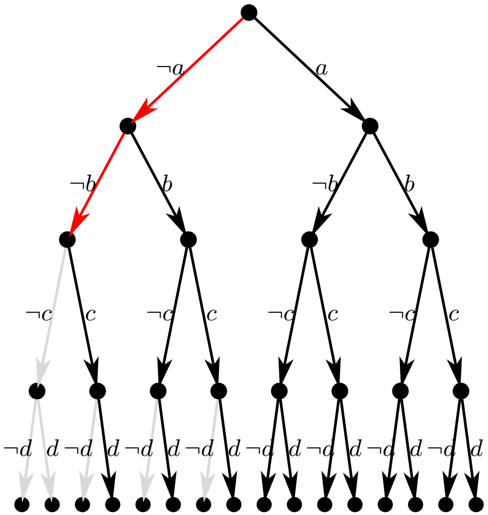


Binary search with unit resolution

The Davis-Putnam-Logemann-Loveland procedure DPLL [DLL62]

$a \vee d$

$b \vee c$



$\neg a$
a
$\neg b$
b
$\neg b$
b
$\neg c$
c
$\neg c$
c
$\neg c$
c
$\neg c$
c
$\neg d$ d $\neg d$ d $\neg d$ d $\neg d$ d $\neg d$ d $\neg d$ d $\neg d$ d $\neg d$ d

Davis-Putnam-Logemann-Loveland procedure
[DLL62]

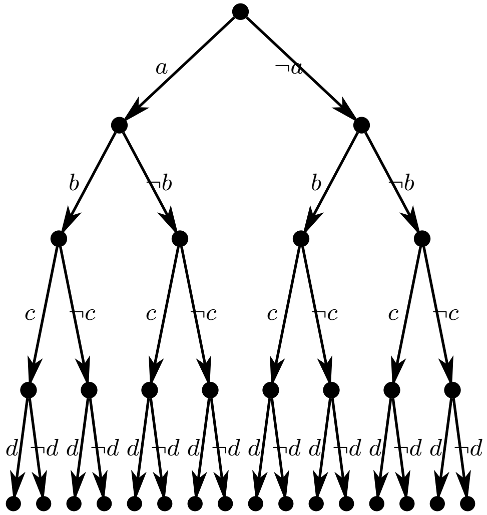
```
1: PROCEDURE DPLL($C$)
2: $C$ := UNIT$(C)$;
3: IF $\{x, \neg x\} \subseteq C$ for some $x \in X$ THEN RETURN false;
4: $x$ := any variable such that $\{x, \neg x\} \cap C = \emptyset$;
5: IF no such variable exists THEN RETURN true;
6: IF DPLL($C \cup \{x\}$) = true THEN RETURN true;
7: RETURN DPLL($C \cup \{\neg x\}$);
```

DPLL with backjumping

- The DPLL backtracking procedure often discovers the same conflicts repeatedly.
- In a branch $l_1, l_2, \dots, l_{n-1}, l_n$, after l_n and $\overline{l_n}$ have led to conflicts (derivation of $\bot$), $\overline{l_{n-1}}$ is always tried next, even when it is irrelevant to the conflicts with l_n and $\overline{l_n}$.
- Backjumping [Gas77] can be adapted to DPLL to backtrack from l_n to l_i when $l_{i+1}, \dots, l_{n-1}$ are all irrelevant.

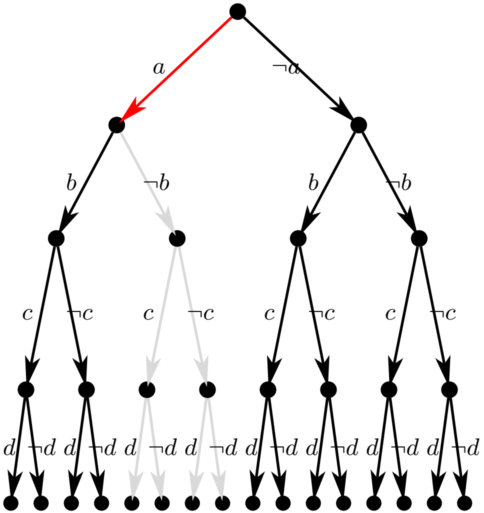
DPLL with backjumping

$\neg a \vee b$
$\neg b \vee \neg d \vee e$
$\neg d \vee \neg e$
$\neg b \vee d \vee e$
$d \vee \neg e$
$c \vee f$


a
$\neg a$
b
$\neg b$
b
$\neg b$
c $\neg c$ c $\neg c$ c $\neg c$ c $\neg c$
d $\neg d$ d $\neg d$ d $\neg d$ d $\neg d$ d $\neg d$ d $\neg d$ d $\neg d$ d $\neg d$

DPLL with backjumping

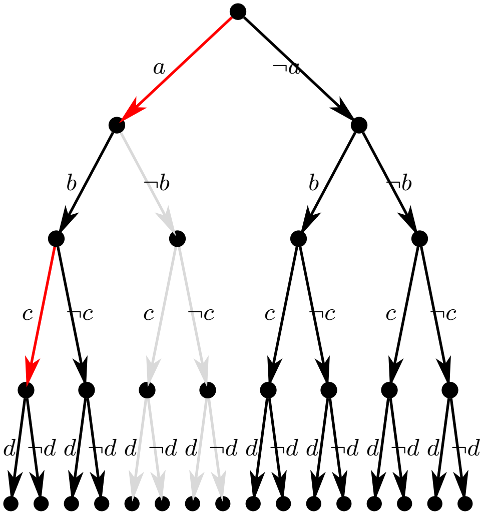
$\neg a \vee b$
$\neg b \vee \neg d \vee e$
$\neg d \vee \neg e$
$\neg b \vee d \vee e$
$d \vee \neg e$
$c \vee f$



a
$\neg a$
b
$\neg b$
b
$\neg b$
c $\neg c$ c $\neg c$ c $\neg c$ c $\neg c$
d $\neg d$ d $\neg d$ d $\neg d$ d $\neg d$ d $\neg d$ d $\neg d$ d $\neg d$ d $\neg d$

DPLL with backjumping

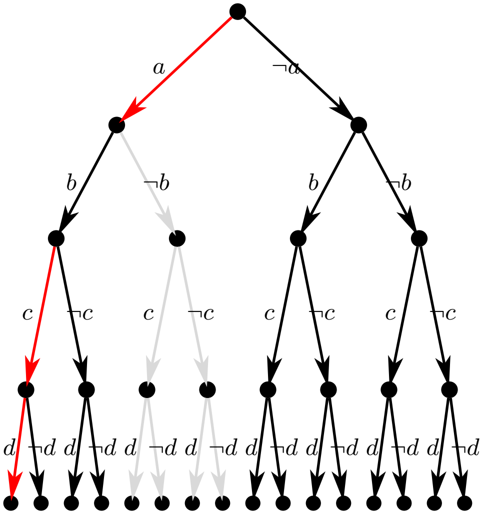
$\neg a \lor b$
$\neg b \lor \neg d \lor e$
$\neg d \lor \neg e$
$\neg b \lor d \lor e$
$d \lor \neg e$
$c \lor f$


a
$\neg a$
b
$\neg b$
b
$\neg b$
c
$\neg c$
c
$\neg c$
c
$\neg c$
c
$\neg c$
d $\neg d$ d $\neg d$ d $\neg d$ d $\neg d$ d $\neg d$ d $\neg d$ d $\neg d$ d $\neg d$

DPLL with backjumping

$\neg a \vee b$
$\neg b \vee \neg d \vee e$
$\neg d \vee \neg e$
$\neg b \vee d \vee e$
$d \vee \neg e$
$c \vee f$

Conflict set with d: $\{a, d\}$

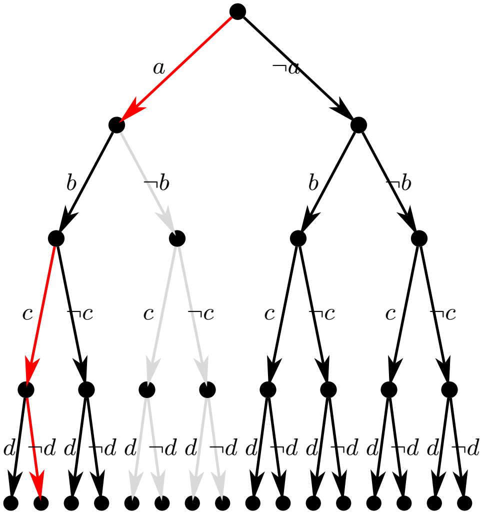


a
$\neg a$
b
$\neg b$
b
$\neg b$
c
$\neg c$
c
$\neg c$
c
$\neg c$
c
$\neg c$
d $\neg d$ d $\neg d$ d $\neg d$ d $\neg d$ d $\neg d$ d $\neg d$ d $\neg d$ d $\neg d$

DPLL with backjumping

$\neg a \vee b$
$\neg b \vee \neg d \vee e$
$\neg d \vee \neg e$
$\neg b \vee d \vee e$
$d \vee \neg e$
$c \vee f$

Conflict set with d: $\{a, d\}$
Conflict set with $\neg d$: $\{a, \neg d\}$

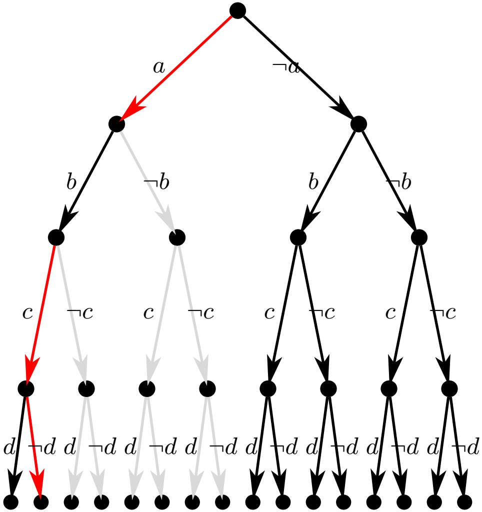


DPLL with backjumping

$$\neg a \vee b$$
$$\neg b \vee \neg d \vee e$$
$$\neg d \vee \neg e$$
$$\neg b \vee d \vee e$$
$$d \vee \neg e$$
$$c \vee f$$

Conflict set with d: $\{a, d\}$
Conflict set with $\neg d$: $\{a, \neg d\}$

No use trying $\neg c$.
Directly go to $\neg a$.


a
$\neg a$
b
$\neg b$
b
$\neg b$
c
$\neg c$
c
$\neg c$
c
$\neg c$
c
$\neg c$
d $\neg d$ d $\neg d$ d $\neg d$ d $\neg d$ d $\neg d$ d $\neg d$ d $\neg d$ d $\neg d$

Conflict-Driven Clause Learning (CDCL)
[MSS96]

- The Resolution rule is more powerful than DPLL: UNSAT proofs by DPLL may be exponentially bigger than the smallest resolution proofs.
- An extension to DPLL, based on learned clauses, is similarly exponentially more powerful than DPLL [BKS04].
- It has been shown that CDCL with restarts is equally powerful to resolution [PD09a].
- In many applications, SAT solvers with CDCL are the best.

Conflict-Driven Clause Learning (CDCL)

- Assume a partial valuation (a path in the DPLL search tree from the root to a leaf node) corresponding to literals $l_1, \dots, l_n$ leads to a contradiction (with unit resolution)

  $$\mathcal{F} \cup \{l_1, \dots, l_n\} \vdash_{UP} \bot$$

  From this follows

  $$\mathcal{F} \models \overline{l_1} \vee \cdots \vee \overline{l_n}.$$

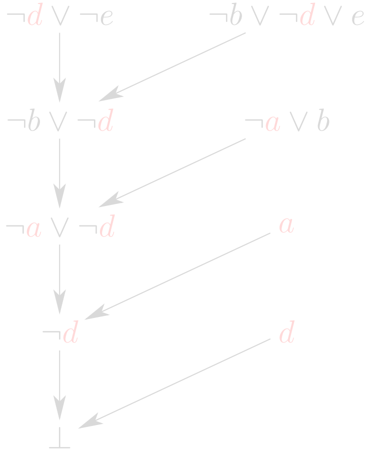
- Often not all of the literals $l_1, \dots, l_n$ are needed for deriving the empty clause $\bot$, and a shorter clause can be derived.

Conflict-Driven Clause Learning (CDCL)
Example

$\neg a \vee b$

$\neg b \vee \neg d \vee e$

$\neg d \vee \neg e$ falsified



$$\begin{tikzcd}
\neg d \vee \neg e \arrow[d] & \neg b \vee \neg d \vee e \arrow[dl] \\
\neg b \vee \neg d \arrow[d] & \neg a \vee b \arrow[dl] \\
\neg a \vee \neg d \arrow[d] & a \arrow[dl] \\
\neg d \arrow[d] & d \arrow[dl] \\
\perp &
\end{tikzcd}$$

Conflict-Driven Clause Learning (CDCL)

Example

$\neg a \vee b$

$\neg b \vee \neg d \vee e$

$\neg d \vee \neg e$ falsified

$$\begin{tikzcd}
\neg d \vee \neg e \arrow[d] & \neg b \vee \neg d \vee e \arrow[dl] \\
\neg b \vee \neg d \arrow[d] & \neg a \vee b \arrow[dl] \\
\neg a \vee \neg d \arrow[d] & a \arrow[dl] \\
\neg d \arrow[d] & d \arrow[dl] \\
\perp &
\end{tikzcd}$$

a, b

Conflict-Driven Clause Learning (CDCL)

Example

$$\neg a \vee b$$

$$\neg b \vee \neg d \vee e$$

$$\neg d \vee \neg e$$ falsified

$$a, b, c$$

$$\begin{tikzcd}
\neg d \vee \neg e \arrow[d] & \neg b \vee \neg d \vee e \arrow[dl] \\
\neg b \vee \neg d \arrow[d] & \neg a \vee b \arrow[dl] \\
\neg a \vee \neg d \arrow[d] & a \arrow[dl] \\
\neg d \arrow[d] & d \arrow[dl] \\
\perp &
\end{tikzcd}$$

Conflict-Driven Clause Learning (CDCL)

Example

$\neg a \vee b$

$\neg b \vee \neg d \vee e$

$\neg d \vee \neg e$ falsified

a, b, c, d, e

$$\begin{tikzcd}
\neg d \vee \neg e \arrow[d] & \neg b \vee \neg d \vee e \arrow[dl] \\
\neg b \vee \neg d \arrow[d] & \neg a \vee b \arrow[dl] \\
\neg a \vee \neg d \arrow[d] & a \arrow[dl] \\
\neg d \arrow[d] & d \arrow[dl] \\
\bot &
\end{tikzcd}$$

Conflict-Driven Clause Learning (CDCL)

Example

$\neg a \vee b$

$\neg b \vee \neg d \vee e$

$\neg d \vee \neg e$ falsified

a, b, c, d, e

$$\begin{tikzcd}
\neg d \vee \neg e \arrow[d] & \neg b \vee \neg d \vee e \arrow[dl] \\
\neg b \vee \neg d \arrow[d] & \neg a \vee b \arrow[dl] \\
\neg a \vee \neg d \arrow[d] & a \arrow[dl] \\
\neg d \arrow[d] & d \arrow[dl] \\
\bot &
\end{tikzcd}$$

Conflict-Driven Clause Learning (CDCL)

Example

$$\neg a \lor b$$

$$\neg b \lor \neg d \lor e$$

$$\neg d \lor \neg e$$ falsified

$$a, b, c, d, e$$

$$\begin{tikzcd}
\neg d \lor \neg e \arrow[d] & \neg b \lor \neg d \lor e \arrow[dl] \\
\neg b \lor \neg d \arrow[d] & \neg a \lor b \arrow[dl] \\
\neg a \lor \neg d \arrow[d] & a \arrow[dl] \\
\neg d \arrow[d] & d \arrow[dl] \\
\bot &
\end{tikzcd}$$

Conflict-Driven Clause Learning (CDCL)
Example

$\neg a \lor b$

$\neg b \lor \neg d \lor e$

$\neg d \lor \neg e$ falsified

a, b, c, d, e

$$\begin{tikzcd}
\neg d \lor \neg e \arrow[d] & \neg b \lor \neg d \lor e \arrow[dl] \\
\neg b \lor \neg d \arrow[d] & \neg a \lor b \arrow[dl] \\
\neg a \lor \neg d \arrow[d] & a \arrow[dl] \\
\neg d \arrow[d] & d \arrow[dl] \\
\bot &
\end{tikzcd}$$

Conflict-Driven Clause Learning (CDCL)

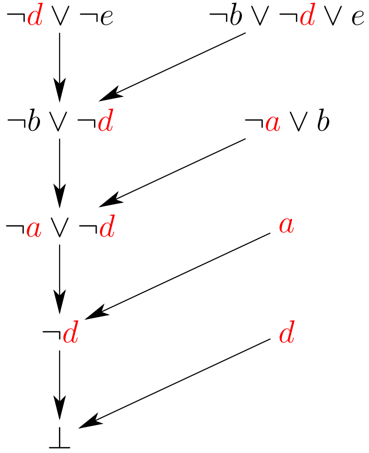
Example

$\neg a \vee b$

$\neg b \vee \neg d \vee e$

$\neg d \vee \neg e$ falsified

a, b, c, d, e



$$\begin{tikzcd}
\neg d \vee \neg e \arrow[d] & \neg b \vee \neg d \vee e \arrow[dl] \\
\neg b \vee \neg d \arrow[d] & \neg a \vee b \arrow[dl] \\
\neg a \vee \neg d \arrow[d] & a \arrow[dl] \\
\neg d \arrow[d] & d \arrow[dl] \\
\bot &
\end{tikzcd}$$

Conflict-Driven Clause Learning (CDCL)
Procedure

The Reason of a Literal

For each non-decision literal l a reason is recorded: it is the clause $l \vee \phi$ from which it was derived with $\neg\phi$.

A Basic Clause Learning Procedure

- Start with the clause $C = l_1 \vee \cdots \vee l_n$ that was falsified.
- Resolve it with the reasons $\bar{l} \vee \phi$ of non-decision literals $\bar{l}$ until only decision variables are left.

Conflict-Driven Clause Learning (CDCL)
Different variants of the procedure

decision scheme	Stop when only decision variables left.
First UIP	(Unique Implication Point) Stop when only one literal of current decision level left.
Last UIP	Stop when at the current decision level only the decision literal is left.

First UIP is usually considered to be the most useful.
Some solvers learn multiple clauses.

Conflict-Driven Clause Learning (CDCL)

Forgetting/deleting clauses

- In contrast to DPLL, a main problem with CDCL is the high number of learned clauses.
- To avoid memory filling up, large numbers of learned clauses are deleted at regular intervals, typically based on clause length, last use, and other criteria.
- One interesting strategy is to rank the clauses according to the number of decision levels appearing in the clause [AS09].

Heuristics for CDCL: VSIDS
Variable State Independent Decaying Sum [MMZ+01]

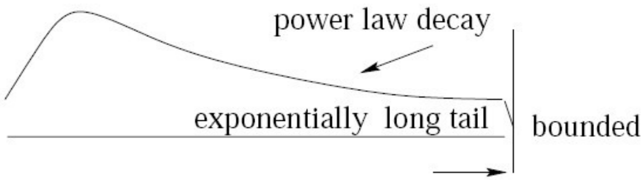
- Initially the score $s(l)$ of literal l is its number of occurrences in $\mathcal{F}$.
- When clause with l is learned, increase $r(l)$.
- Periodically decay the scores:

$$s(l) := r(l) + 0.5s(l); \qquad r(l) := 0;$$

- Always choose unassigned literal l with maximum $s(l)$.

Variations and extensions of VSIDS most popular in current solvers.

Heavy-tailed runtime distributions



(Diagram from [CGS01])

On many NP-complete problems, heavy-tailed distributions characterize

- runtimes of a randomized algorithm on a single instance and
- runtimes of a deterministic algorithm on a class of instances.

Heavy-tailed runtime distributions
Estimating the mean is problematic

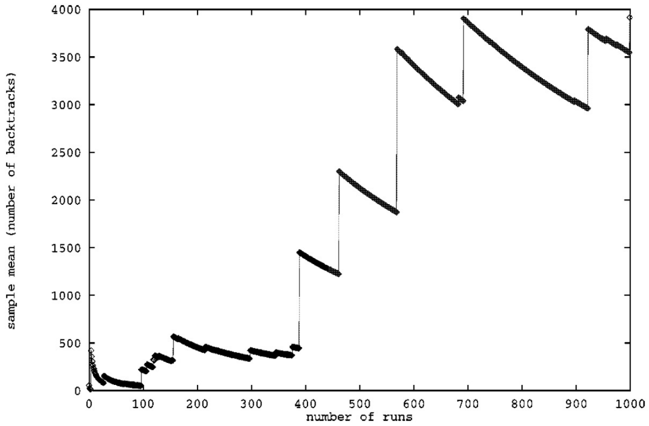


Diagram from [GSCK00]

Heavy-tailed runtime distributions
Cause

- A small number of wrong decisions lead to a part of the search tree not containing any solutions.
- Backtrack-style search needs a long time to traverse the search tree.
    - Many short paths from the root node to a success leaf node.
    - High probability of reaching a huge subtree with no solutions.

These properties mean that

- average runtime is high,
- restarting the procedure after t seconds reduces the mean substantially, if t is close to the mean of the original distribution.

Restarts in SAT algorithms
Answer to heavy-tailedness

Restarts had been used in stochastic local search algorithms:

- Necessary for escaping local minima!

Gomes et al. demonstrated the utility of restarts for systematic SAT solvers:

- Small amount of randomness in branching variable selection.
- Restart the algorithm after a given number of seconds.

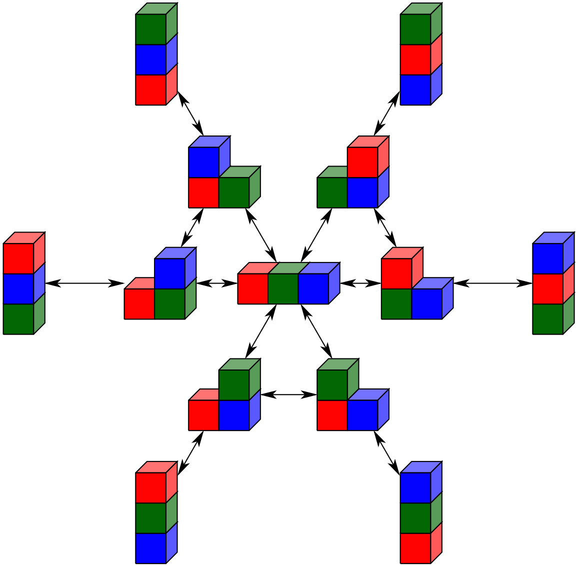
Restarts with CDCL

- Learned clauses are retained when doing the restart.
- Problem: Optimal restart policy depends on the runtime distribution, which is generally not known.
- Problem: Deletion of learned clauses and too early restarts may lead to non-termination for unsatisfiable formulas. This is avoided by gradually increasing restart interval.
- One effective restart strategy is based on the Luby series $n = 1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, ...$, learning e.g. $30n$ clauses between consecutive restarts [Hua07].

Application: Reachability

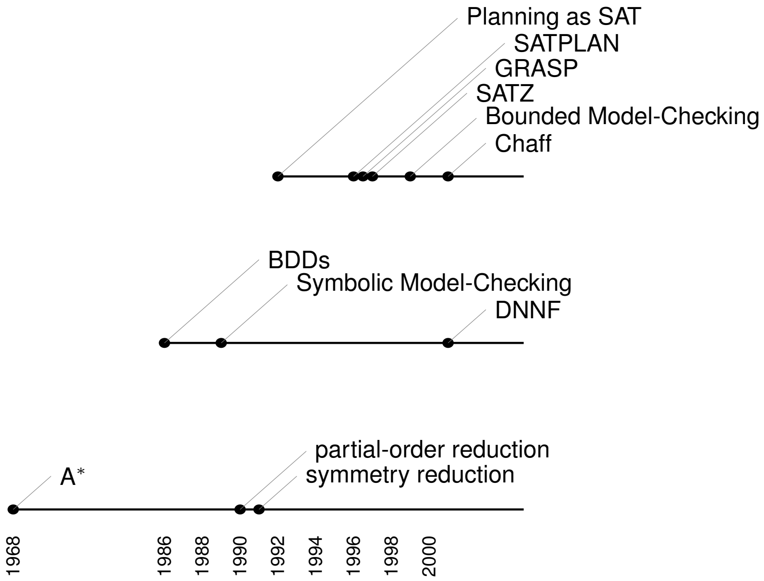
- finding a path from a state from in I to a state in set G in a succinctly/compactly represented graph
- PSPACE-complete [GW83, Loz88, LB90, Byl94]
- in NP when restricted to paths of polynomial length
- Basis of efficient solutions to
  - planning problem in AI [KS92, KS96]
  - LTL model-checking problem [BCCZ99]
  - DES diagnosis problem [GARK07]
- Often replacing traditional state-space search methods
- One of the first and most prominent applications of SAT
- Extensions to timed systems with SAT modulo Theories (SMT)

State-space transition graphs
Blocks world with three blocks



State-space search and satisfiability

Explicit state-space search; symbolic search with BDDs, SAT



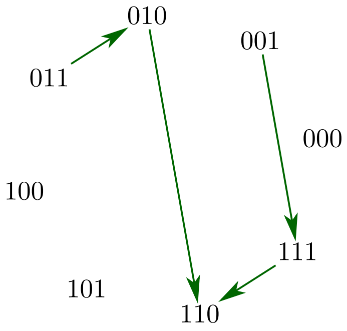
Transition relations in propositional logic

State variables are
$X = \{a, b, c\}$.

$$(\neg a \wedge b \wedge c \wedge \neg a' \wedge b' \wedge \neg c') \vee$$
$$(\neg a \wedge b \wedge \neg c \wedge a' \wedge b' \wedge \neg c') \vee$$
$$(\neg a \wedge \neg b \wedge c \wedge a' \wedge b' \wedge c') \vee$$
$$(a \wedge b \wedge c \wedge a' \wedge b' \wedge \neg c')$$

The corresponding matrix is

	000	001	010	011	100	101	110	111
000	0	0	0	0	0	0	0	0
001	0	0	0	0	0	0	0	1
010	0	0	0	0	0	0	1	0
011	0	0	1	0	0	0	0	0
100	0	0	0	0	0	0	0	0
101	0	0	0	0	0	0	0	0
110	0	0	0	0	0	0	0	0
111	0	0	0	0	0	0	1	0



Transition relations in propositional logic

Let $X = \{x_1, \dots, x_n\}$ be the state variables.

- Any deterministic action/event corresponds to a partial function.
  Partial functions correspond to conjunctions of a precondition formula $\Pi(x_1, \dots, x_n)$ and equivalences

  $$x'_i \leftrightarrow F(x_1, \dots, x_n)$$

  for every $x_i \in X$.
- Choice between actions/events $\alpha_1, \dots, \alpha_k$ corresponds to

  $$\Phi = \alpha_1 \vee \dots \vee \alpha_k.$$

Reachability as SAT

Let $\Phi(n, m)$ denote the formula obtained from Φ by replacing each $x \in X$ by $x@n$ and each x' by $x@m$.
Then satisfying valuations of

$$\Phi(0,1) \wedge \Phi(1,2) \wedge \cdots \Phi(n-1,n)$$

are in 1-to-1 correspondence to paths of length n in the transition graph.

Testing whether a state satisfying G can be reached from a state satisfying I in n steps reduces to testing the satisfiability of

$$I(0) \wedge \Phi(0,1) \wedge \Phi(1,2) \wedge \cdots \Phi(n-1,n) \wedge G(n).$$

Applications

Interpretations of SAT tests

$$I(0) \wedge \Phi(0,1) \wedge \Phi(1,2) \wedge \cdots \Phi(n-1,n) \wedge G(n).$$

Planning Can goals G be reached from the initial state I [KS96]?

Model-checking Can the safety property $\neg G$ be violated on executions that start from I? (Extensions for LTL model-checking in [BCCZ99].)

DES Diagnosis Consider

$$\Phi(0,1) \wedge \Phi(1,2) \wedge \cdots \Phi(n-1,n) \wedge (o_1@t_1 \wedge \cdots \wedge o_m@t_m) \wedge F.$$

Are observations $o_1, \ldots, o_m$ respectively at $t_1, \ldots, t_m$ compatible with fault assumptions F [GARK07]? F encodes e.g. “there are n faults between time points 0 and n.

Applications

The most basic encodings given above can often be improved.

- optimal (linear-size) encodings [LBHJ04, RHN06]
- multiple actions in parallel [RHN06]
- scheduling the SAT tests for different path lengths [Rin04, Zar04] in parallel
- search heuristics replacing VSIDS [Gan11, Rin10, Rin12b]
- reachability-specific implementation technology [Rin12a]

MAXSAT
Motivation

- Many AI problems involve optimization:
  - Learn an explanation with the best match to data [Cus08].
  - Find a least-cost plan [RGPS10].
  - Select best drugs for cancer therapy [LK12].
- SAT insufficient: answers a binary yes–no question
- MAXSAT extends SAT with a basic form of optimization.
- Other frameworks: Mixed Integer-Linear Programming (MILP/ILP/MIP), constraint programming and optimization [DRGN10], SMT + optimization [ST12]
- advantage over MILP: efficient Boolean reasoning

Introduction: (Weighted) (Partial) MAXSAT

plain MAXSAT	Maximize the number of satisfied clauses
partial MAXSAT	Maximize the number of satisfied soft clauses Hard clauses must be satisfied
weighted MAXSAT	Maximize the sum of weights of satisfied clauses

Decision problem “is there an valuation with weight $\geq n$” NP-complete.

The FP^{NP} optimization problem solvable by a polynomial number of SAT calls.

Algorithms for MAXSAT

- reduction to a sequence of SAT problems [FM06, ABL13, DB11]
- branch and bound [HLO08, LMMP10]
- Mixed Integer Linear Programming [DB13] (CPLEX)

Some MAXSAT solvers

dfs + bounding	MaxSatz, MiniMaxSat
SAT	sat4j, wbo, wpm, pwbo, maxhs

MAXSAT by a sequence of SAT queries

1. From a weighted partial MAXSAT instance, construct a SAT instance [FM06, ABL13]:
   - Hard clauses are taken as is.
   - For each soft clause $l_1 \vee \cdots \vee l_n$, have $b \vee l_1 \vee \cdots \vee l_n$, where b is a new auxiliary variable.
2. If the SAT instance is unsatisfiable, the best valuation so far is the globally best (And if this was the first time here, the hard clauses are unsatisfiable.)
3. Otherwise, each true b variable corresponds to a (possibly) false soft clause.
4. Calculate the sum F of the weights of true soft clauses.
5. Construct a new SAT instance, with cardinality constraints [BB03, Sin05] requiring that weights of true soft clauses $> F$.
6. One can also add a clause requiring at least one previously false soft clause to be true. (unsatisfiable cores [ABL13])
7. Continue from step 2.

Other query strategies

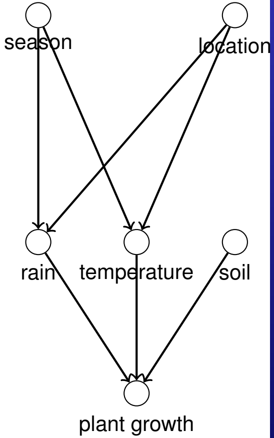
Given SAT instances saying “at most k soft clauses are false”, alternative query strategies are possible.

- unsatisfiability based: try $k = 0$, then $k = 1$, and so on.
- satisfiability based: try $k = k_{max} - 1$, then $k = k_{max} - 2$, and so on.
- binary search: try half-way between 0 and k_{max}, and after tightening either lower or upper bound, then again half-way.

Same question of SAT queries with different parameter values k arises also in other SAT and constraints applications, including planning and scheduling, with other algorithms proposed [Rin04, SS07]. (Usefulness of these algorithms to MAXSAT is not clear.)

Bayesian networks

- Compact representation of probability distributions [Pea89]
- Makes probabilistic dependence and independence explicit.
- lots of applications e.g. in intelligent robotics, especially for dynamic Bayesian networks
- Other graphical models: Markov networks [Pea89]



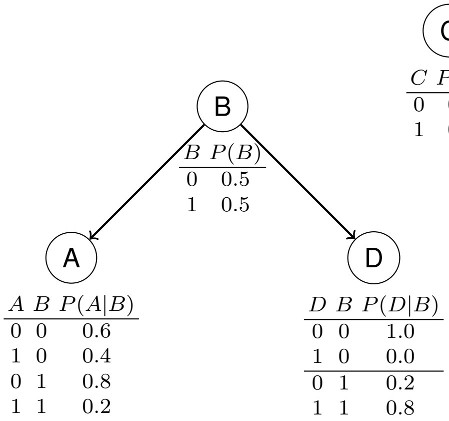
Bayesian networks

- Probabilistic Inference (PI): calculate marginal probability of a variable given evidence
- Most Probable Explanation (MPE): find a valuation for the variables with the highest probability
- Maximum A Posteriori hypothesis (MAP) [PD04]: find hypotheses that explain the observations best
- Structure Learning (SL): find Bayesian network that best matches given data

problem	complexity	SAT variant
PI	#P	#SAT
MPE	FP^{NP}	MAXSAT
MAP	NP^{PP}	E-MAJSAT (SSAT)
SL	FP^{NP}	MAXSAT

MPE: Most Probable Explanation

- Of all valuations of the variables, find one with the highest probability.
- Has the flavor of diagnosis problems (but see the MAP problem later!)
- Solution e.g. by reduction to MAXSAT [KD99, Par02]



C	$P(C)$
0	0.9
1	0.1

B	$P(B)$
0	0.5
1	0.5

A	B	$P(A\|B)$
0	0	0.6
1	0	0.4
0	1	0.8
1	1	0.2

D	B	$P(D\|B)$
0	0	1.0
1	0	0.0
0	1	0.2
1	1	0.8

Reduction of MPE to MAXSAT

A	B	$P(A\|B)$
0	0	0.6
1	0	0.4
0	1	0.8
1	1	0.2

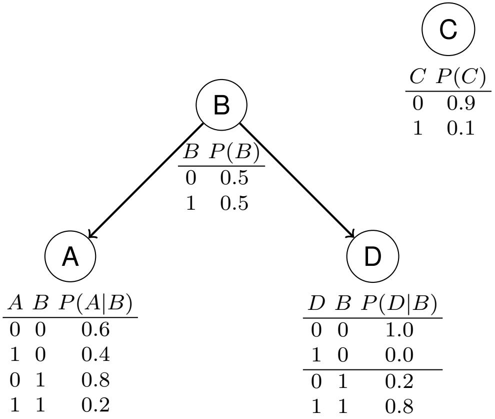
translates to

$\neg A \wedge \neg B$	probability 0.6
$\neg A \wedge B$	probability 0.4
$A \wedge \neg B$	probability 0.8
$A \wedge B$	probability 0.2

- Problem 1: Probabilities must be multiplied to get the overall probability.
- Solution: Sum the logarithms of the probabilities.
- Problem 2: Probabilities 0 correspond to $\log 0 = \infty$.
- Solution: Use hard clauses.
- Negate the conjunctions to get clauses.
  Negate $\log p$ (with $p \leq 1$) to get positive weights.

Structure Learning for Bayesian network

ABCD
0000
0101
1000
0101
0000
1101
0010
1000
0101
0100



B	$P(B)$
0	0.5
1	0.5

C	$P(C)$
0	0.9
1	0.1

A	B	$P(A\|B)$
0	0	0.6
1	0	0.4
0	1	0.8
1	1	0.2

D	B	$P(D\|B)$
0	0	1.0
1	0	0.0
0	1	0.2
1	1	0.8

Structure Learning for Bayesian networks
Mapping to Constraint Satisfaction, including MAXSAT

- The score of a network is the sum of all per-node scores.
- The score of each node is determined by its parents: each alternative parent set has a score.
- Constraint satisfaction formulation:
    - Choose a parent set for each node. (E.g. max. 3 parents)
    - The resulting graph must be acyclic.
    - Objective: maximize the sum of the parent set scores.
- main challenge in encoding: acyclicity constraint
    - transitive ancestor relation [Cus08]
    - total ordering of nodes [Cus08]
    - recursively define distance from leaf 0, 1, 2, ...

Structure Learning for Bayesian networks

- Finding optimal nets translatable into MAXSAT, MILP etc.
- Optimal solutions found for nets of up to some dozens of nodes.
- On many standard benchmarks, MAXSAT and MILP solvers comparable.
- Best methods enhance MILP with specialized heuristics [Cus11].

Methods used for approximate solutions are different!

Model-Counting (#SAT)

- How many satisfying valuations does a propositional formula have?
- The problem is #P-complete [Val79].
- Interestingly, model-counting is #P-complete also when SAT is easy (in P): DNF-SAT, 2-SAT, Horn-SAT, ... [Val79].
- #P harder than NP: $\phi \in$SAT if and only if model-count ≥ 1

Weighted Model-Counting

- Weighted Model-Counting assigns a weight to each literal.
- Compute the sum of the weights of satisfying valuations.
- Weight of a valuation is the product of weights of true literals.
- This generalization is useful e.g. for probabilistic reasoning.
- Coincides with unweighted MC when all weights are 1.

Algorithms for Model Counting

- exact algorithms: extensions of DPLL and CDCL [BDP03, BDP09, SBB$^+$04, SBK05a, GSS09]
- approximate counting (upper bound)
- approximate counting (no guaranteed lower or upper bound) [KSS11]

Algorithms
extensions of DPLL and CDCL

- basic algorithm: DPLL-style tree search
- connected components [BP00]
- component caching [BDP03]
- combining clause-learning with component caching [SBB$^+$04]
- heuristics [SBK05a]

Basic model-counting DPLL algorithm

Consider a model-counting run of DPLL for a formula with propositional variables X.

- Two branches $\{x\} \cup C$ and $\{\neg x\} \cup C$ disjoint $\Longrightarrow$ take the sum the respective model counts.
- When DPLL detects that all clauses are satisfied with n variables assigned, the count for the branch is

$$2^{|X|-n}$$

Component analysis and component caching

Enhancements to the basic model-counting DPLL (e.g. in Cachet [SBB$^+$04]):

- Component analysis: if C can be partitioned to $(C_1, \ldots, C_n)$ so that partitions don't share variables, then count each C_i separately and take the product of the counts [BP00]
- Component caching [BDP03]: record model-counts and recall them when encountering a clause set again.

Efficient model-counts for normal forms

- Model-counting for CNF (#SAT) is #P-complete [Val79].
- Some normal forms have polynomial time model-counting.
  - Binary Decision Diagrams (BDD) [Bry92]
  - deterministic Decomposable Negation Normal Form (d-DNNF) [Dar02]
- Reaching these normal forms can take exponential time, space.
- Some of the best translators for these normal forms [HD07] are similar to the model-counting variants of the Davis-Putnam procedure, for example in utilizing component analysis.

MC Applications: Bayesian inference

- optimal distinguishing tests [HS09]
- Bayesian inference [BDP09, SBK05b, CD08], calculating marginal probabilities of some variables given values of other variables of a Bayesian network.
  (There are interesting connections between specialized Bayesian inference algorithms and model-counting algorithms. E.g., many can be viewed as instances of algorithms for the SumProd problem [BDP09].)

Probabilistic Inference by Model-Counting

Marginal probability of given evidence

$$\begin{tikzcd}
C & & B \arrow[dl] \arrow[dr] & \\
& A & & D
\end{tikzcd}$$

C	$P(C)$
0	0.9
1	0.1

B	$P(B)$
0	0.5
1	0.5

A	B	$P(A\|B)$
0	0	0.6
1	0	0.4
0	1	0.8
1	1	0.2

D	B	$P(D\|B)$
0	0	1.0
1	0	0.0
0	1	0.2
1	1	0.8

- Variable for each node A, B, C, D.
- Parentless nodes have the obvious weights $w(B) = w(\neg B) = 0.5$, $w(C) = 0.1, w(\neg C) = 0.9$.
- Chance variables $c_{A|B}$ and $c_{A|\neg B}$ for nodes with parents.

$$\begin{array}{cc}
w(c_{A|B}) = 0.2 & w(\neg c_{A|B}) = 0.8 \\
w(c_{A|\neg B}) = 0.4 & w(\neg c_{A|\neg B}) = 0.6 \\
w(A) = 1 & w(A) = 1
\end{array}$$

- $B \wedge c_{A|B} \rightarrow A$
  $B \wedge \neg c_{A|B} \rightarrow \neg A$
  $\neg B \wedge c_{A|\neg B} \rightarrow A$
  $\neg B \wedge \neg c_{A|\neg B} \rightarrow \neg A$
- Conditioning with evidence $B, \neg C$ by adding in the clause set.

Stochastic Satisfiability SSAT

- Stochastic satisfiability [Pap85] extends propositional logic with stochastic AND-OR quantification. (An extension of Quantified Boolean formulas (QBF) [Sto76]).
- Prefix consisting of variables quantified by $\exists$, $\forall$ and $\reflectbox{R}^r$, followed by a propositional formula.

In SSAT, the probability $P(\phi)$ associated with a formula ϕ is defined recursively as follows.

- Base case: variable free (quantifier free) formulas containing only atomic formulas $\perp$ and $\top$ and Boolean connectives.
  $P(\top) = 1.0$
  $P(\perp) = 0.0$
- $P(\exists x \phi) = \max(P(\phi[\top/x]), P(\phi[\perp/x]))$
- $P(\reflectbox{R}^r x \phi) = r \times P(\phi[\top/x]) + (1 - r) \times P(\phi[\perp/x])$
- $P(\forall x \phi) = \min(P(\phi[\top/x]), P(\phi[\perp/x]))$

Question: Is $P(\phi) \geq R$ for some $R \in [0, 1[$?

Stochastic Satisfiability SSAT
Special cases

SSAT can be viewed as a generalization of

- SAT: quanfiers $\exists$ only
- TAUT: quanfiers $\forall$ only
- quantified Boolean formulas (QBF): quantifiers $\exists$, $\forall$ only [Sto76]
- E-MAJSAT: prefix $\exists\exists\cdots\exists\text{ꓤ}^{r_1}\text{ꓤ}^{r_2}\cdots\text{ꓤ}^{r_n}$ [PD09b]

Algorithms for E-MAJSAT and SSAT

- Basic approach [Lit99, LMP01]:
  - DPLL-style tree search
  - variables selected in quantification order
  - prune subtrees if irrelevant for establishing the lb R (thresholding [ML03])
  - component caching (as in model-counting #SAT)
- Implementations reported by Majercik, Littman, Boots [ML03, MB05].
- resolution rule [TF10] (following QBF resolution [KBKF95])
- SMT-style extension to cover the orthogonal problem of combining SAT with linear arithmetics (SSMT [TEF11])

Applications

- Maximum A Posteriori Hypothesis (MAP) is NP^{PP}-complete [PD04], corresponding to E-MAJSAT $(\exists \cdots \rotatebox[origin=c]{180}{R}^{r} \cdots)$
- MAP application: diagnosis
- Probabilistic verification of safety critical systems: what is the probability that event x will take place? [TF11]
- probabilistic planning [ML03]

MAP: Maximum A Posteriori Hypothesis

- MPE finds a single most probable valuation of variables.
- The probability of this valuation is typically low, and it is often not representative of the most likely fault e.g. in diagnosis.
- The Maximum A Posteriori Hypothesis (MAP) problem [PD04]:
  Find a valuation to a subset of hypothesis variables H that maximizes the probability of the given observations.
- Decision version of MAP is NP^{PP}-complete: guess a valuation of H; then verify that the probability of the observations is $\geq r$ for a given bound r.

MAP: Maximum A Posteriori Hypothesis
Encoding as E-MAJSAT

- How to choose hypotheses $h_1, \ldots, h_n$ to maximize the probability expressed by $\exists x_1 \exists x_2 \cdots \exists x_n \rotatebox[origin=c]{180}{R}^{0.5} y_1 \cdots \rotatebox[origin=c]{180}{R}^{0.5} y_m \phi$
- Encoding like Probabilistic Inference with Model-Checking. Difference is quantification:

$$\exists h_1 \exists h_2 \cdots \exists h_n \rotatebox[origin=c]{180}{R}^{w_1} x_1 \cdots \rotatebox[origin=c]{180}{R}^{w_m} x_m \Phi$$

where $x_1, ..., x_m$ are all the non-hypothesis variables with the same weights $w_1, \ldots, w_m$ as in the Probabilistic Inference problem.

Probabilistic planning by SSAT
[ML98]

$$\exists P \text{Я}^q C \exists E \left(I^0 \rightarrow \left(\bigwedge_{i=0}^{t-1} \mathcal{T}(i, i+1) \wedge G^t \right) \right) \qquad (1)$$

1. 1st block: $\exists$-quantification over all action sequences
2. 2nd block: Я-quantification over all contingencies
3. 3rd block: $\exists$-quantification over all executions of the plan

SMT: Satisfiability Modulo Theories

- numbers needed in representing
  - time
  - space (distance, size, ...)
  - resources (money, materials, ...)
- SAT has no numbers: reduction to SAT is feasible only for small integers
- SAT modulo Theories = SAT + specialized solvers for specific theories, such as
  - linear integer/rational/real arithmetic
  - bitvectors
  - graphs
- Similar to constraint programming frameworks.

Basic ideas of SMT

- Not everything is compactly expressible and efficiently solvable if only Boolean variables are used, for example real and rational arithmetics.
- SAT can be extended with non-Boolean theories. A clause has the form

$$l_1 \vee \cdots \vee l_n \vee E$$

  where E is a set of quantifier-free inequations over some set V of real/rational/other variables.
- The theories can be e.g.
  - linear inequalities,
  - mixed integer integer linear programs, or
  - something completely different.
- Compare: mixed integer linear programming MILP

SMT: Algorithms
Implementation

Extension of DPLL to theories

1. Run DPLL ignoring the inequations in the clauses.
2. After all Boolean variables have been set (at a leaf of the DPLL search tree), take the inequations $E_1, \ldots, E_m$ from all clauses that have no true literal.
3. Test with a specialized solver if $E_1 \cup \cdots \cup E_m$ is solvable. If it is, terminate.
4. Otherwise backtrack with the DPLL algorithm.

- The general idea is easy to implement for different theories, e.g. linear arithmetic.
- Early pruning of the DPLL search tree can be achieved by running the arithmetic solver before all Boolean variables are set.

SMT applications

- reachability with numeric state variables: planning with resources [WW99]
- reachability for timed transition systems: model-checking of timed systems [ACKS02], planning [SD05])

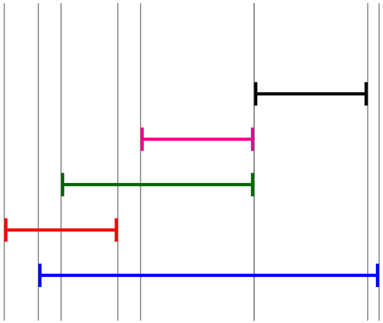
Timed systems reachability

- The most basic reachability problem (e.g. classical planning) is about instantaneous/asynchronous changes of (discrete) state variables.
- In timed systems, change may have a duration or a delay.
- Multiple simultaneous overlapping changes
- Change of continuous state variables may be continuous.
- Lots of applications: model-checking/verification of timed systems, temporal planning, temporal diagnosis, ...

SMT formalization of Timed Systems

Represent system state at time points where something non-continuous happens.

- Action is taken.
- Delayed effect of action takes place.
- A continuously changing variable reaches a critical value.



SMT formalization of Timed Systems

Actions and counters

Variable $\Delta@t$ indicates duration between time points $t-1$ and t.

Following is for actions a, state variables x, and counters C.

precondition of action	$a@t \rightarrow \phi@t$
counter initialization	$a@t \rightarrow (C@t = c)$
counter update	$\neg a@t \rightarrow (C@t = C@(t-1) - \Delta_t)$
discrete change	$(C@t = 0) \rightarrow x@t$
discrete change	$(C@t = 0) \rightarrow \neg x@t$
frame axiom	$(x@(t-1) \wedge \neg x@t) \rightarrow (C_1@t > 0 \vee \cdots$

Additionally, we need formulas to prevent overlap of actions using same resources.

SMT formalization of Timed Systems

Progress of time

Progress of time $\Delta@t$ between points $t-1$ and t.	
progress always positive	$\Delta@t > 0$
don't pass a scheduled change	$\Delta@t \leq C_k@(t-1)$

SMT applications

- Timed and hybrid systems analysis and verification [ABCS05]
- Planning in timed and hybrid systems [SD05]
- Timed and hybrid systems diagnosis:
  - Representation of observations: absolute time points
  - Representation of observations: temporal uncertainty

Other approaches to Timed Systems Reachability

- Explicit state-space search in the space of timed states (e.g. the UPPAAL model-checker [BLL$^+$96])
- Generate untimed transition sequences with SAT, then test whether possible to schedule [RGS13].
- Each method has strengths in different types of problems.

Conclusion
Algorithms

- NP-complete problems have become more solvable since mid-1990ies
- strength of algorithms such as CDCL over a wide range of SAT problems and applications
- convergence of search methods in different areas:
  - Probabilistic Inference for Bayesian networks vs. Model-Counting (#SAT)
  - reachability in AI planning and Computer Aided Verification
- increasing connections to combinatorial optimization methods, e.g. Mixed Integer Linear Programming

Conclusion
Problems

mappings complexity class - SAT variant - AI problem for reachability, planning, games:

NP	SAT	succinct reachability (poly-length paths)
NP	SMT	timed systems reachability (poly-length paths)
NP^{PP}	SSAT	succinct stochastic reachability (poly-length paths)
PSPACE	QBF	(succinct) 2-player games winning strategies
PSPACE	SSAT	stochastic 2-player games optimal strategies

probabilistic reasoning:

FP^{NP}	MAXSAT	Bayesian network MPE, SL
#P	#SAT	Bayesian network PI
NP^{PP}	E-MAJSAT	Bayesian network MAP

References I

Gilles Audemard, Marco Bozzano, Alessandro Cimatti, and Roberto Sebastiani.
Verifying industrial hybrid systems with MathSAT.
Electronic Notes in Theoretical Computer Science, 119(2):17–32, 2005.

Ansótegui, Maria Luisa Bonet, and Jordi Levy.
SAT-based MaxSAT algorithms.
Artificial Intelligence, 196:77–105, 2013.

Gilles Audemard, Alessandro Cimatti, Artur Korniłowicz, and Roberto Sebastiani.
Bounded model checking for timed systems.
In *Formal Techniques for Networked and Distributed Systems - FORTE 2002*, number 2529 in Lecture Notes in Computer Science, pages 243–259. Springer-Verlag, 2002.

Gilles Audemard and Laurent Simon.
Predicting learnt clauses quality in modern SAT solvers.
In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, pages 399–404. Morgan Kaufmann Publishers, 2009.

Olivier Bailleux and Yacine Boufkhad.
Efficient CNF encoding of Boolean cardinality constraints.
In Francesca Rossi, editor, *Principles and Practice of Constraint Programming – CP 2003: 9th International Conference, Kinsale, Ireland, September 29 – October 3, 2003, Proceedings*, volume 2833 of *Lecture Notes in Computer Science*, pages 108–122. Springer-Verlag, 2003.

References II

Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu.
Symbolic model checking without BDDs.
In W. R. Cleaveland, editor, *Tools and Algorithms for the Construction and Analysis of Systems, Proceedings of 5th International Conference, TACAS'99*, volume 1579 of *Lecture Notes in Computer Science*, pages 193–207. Springer-Verlag, 1999.

Fahiem Bacchus, Shannon Dalmao, and Toniann Pitassi.
Algorithms and complexity results for #SAT and Bayesian inference.
In *Foundations of Computer Science, 2003. Proceedings. 44th Annual IEEE Symposium on*, pages 340–351. IEEE, 2003.

Fahiem Bacchus, Shannon Dalmao, and Toniann Pitassi.
Solving #SAT and Bayesian inference with backtracking search.
Journal of Artificial Intelligence Research, 34(2):391–442, 2009.

Daniel R. Brooks, Esra Erdem, Selim T. Erdoğan, James W. Minett, and Don Ringe.
Inferring phylogenetic trees using answer set programming.
Journal of Automated Reasoning, 39(4):471–511, 2007.

Paul Beame, Henry Kautz, and Ashish Sabharwal.
Towards understanding and harnessing the potential of clause learning.
Journal of Artificial Intelligence Research, 22:319–351, 2004.

References III

Johan Bengtsson, Kim Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi.
UPPAAL - a tool suite for automatic verification of real-time systems.
In *Hybrid Systems III*, volume 1066 of *Lecture Notes in Computer Science*, pages 232–243. Springer-Verlag, 1996.

Robert J Bayardo and Joseph Daniel Pehoushek.
Counting models using connected components.
In *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI-2000) and the 12th Conference on Innovative Applications of Artificial Intelligence (IAAI-2000)*, pages 157–162. AAAI Press / The MIT Press, 2000.

R. E. Bryant.
Symbolic Boolean manipulation with ordered binary decision diagrams.
ACM Computing Surveys, 24(3):293–318, September 1992.

Maria Luisa Bonet and Katherine St. John.
Efficiently calculating evolutionary tree measures using SAT.
In *Proceedings of the 11th International Conference on Theory and Applications of Satisfiability Testing (SAT-2009)*, pages 4–17. Springer-Verlag, 2009.

Tom Bylander.
The computational complexity of propositional STRIPS planning.
Artificial Intelligence, 69(1-2):165–204, 1994.

References IV

Mark Chavira and Adnan Darwiche.
On probabilistic inference by weighted model counting.
Artificial Intelligence, 172(6-7):772–799, 2008.

Orly Cohen, Moran Gordon, Michael Lifshits, Alexander Nadel, and Vadim Ryvchin.
Designers work less with quality formal equivalence checking.
In *Design and Verification Conference (DVCon)*, 2010.

Hubie Chen, Carla Gomes, and Bart Selman.
Formal models of heavy-tailed behavior in combinatorial search.
In Toby Walsh, editor, *Proceedings of the 7th International Conference on Principles and Practice of Constraint Programming*, number 2239 in Lecture Notes in Computer Science, pages 408–421. Springer-Verlag, 2001.

S. A. Cook.
The complexity of theorem proving procedures.
In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pages 151–158, 1971.

James Cussens.
Bayesian network learning by compiling to weighted MAX-SAT.
In *UAI'08, Proceedings of the 24th Conference in Uncertainty in Artificial Intelligence*, pages 105–112, 2008.

References V

James Cussens.
Bayesian network learning with cutting planes.
In *Proceedings of the Twenty-Seventh Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-11)*, pages 153–160. AUAI Press, 2011.

Adnan Darwiche.
A compiler for deterministic, decomposable negation normal form.
In *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI-2002) and the 14th Conference on Innovative Applications of Artificial Intelligence (IAAI-2002)*, pages 627–634, 2002.

Jessica Davies and Fahiem Bacchus.
Solving MAXSAT by solving a sequence of simpler SAT instances.
In *Principles and Practice of Constraint Programming - CP 2011, 17th International Conference*, volume 6876 of *Lecture Notes in Computer Science*, pages 225–239. Springer, 2011.

Jessica Davies and Fahiem Bacchus.
Exploiting the power of MIPs solvers in MaxSAT.
In *Proceedings of the 14th International Conference on Theory and Applications of Satisfiability Testing (SAT-2013)*, Lecture Notes in Computer Science. Springer-Verlag, 2013.

W. F. Dowling and J. H. Gallier.
Linear-time algorithms for testing the satisfiability of propositional Horn formulae.
Journal of Logic Programming, 1(3):267–284, 1984.

References VI

Evgeny Dantsin, Andreas Goerdt, Edward A. Hirsch, Ravi Kannan, Jon M. Kleinberg, Christos H. Papadimitriou, Prabhakar Raghavan, and Uwe Schöning.
A deterministic $(2 - 2/(k+1))^n$ algorithm for k-SAT based on local search.
Theoretical Computer Science, 289(1):69–83, 2002.

Evgeny Dantsin, Edward A. Hirsch, and Alexander Wolpert.
Clause shortening combined with pruning yields a new upper bound for deterministic SAT algorithms.
Electronic Colloquium on Computational Complexity (ECCC), 102, 2005.

M. Davis, G. Logemann, and D. Loveland.
A machine program for theorem proving.
Communications of the ACM, 5:394–397, 1962.

Luc De Raedt, Tias Guns, and Siegfried Nijssen.
Constraint programming for data mining and machine learning.
In *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI-10)*, pages 1513–1518. AAAI Press, 2010.

Zhaohui Fu and Sharad Malik.
On solving the partial MAX-SAT problem.
In Armin Biere and CarlaP. Gomes, editors, *Theory and Applications of Satisfiability Testing - SAT 2006*, volume 4121 of *Lecture Notes in Computer Science*, pages 252–265. Springer-Verlag, 2006.

References VII

Malay K. Ganai.
DPLL-based SAT solver using with application-aware branching, July 2011.
patent US 2011/0184705 A1; filed August 31, 2010; provisional application January 26, 2010.

Alban Grastien, Anbulagan, Jussi Rintanen, and Elena Kelareva.
Diagnosis of discrete-event systems using satisfiability algorithms.
In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI-07)*, pages 305–310. AAAI Press, 2007.

J. Gaschnig.
A general backtrack algorithm that eliminates most redundant tests.
In *Proceedings of the 5th International Joint Conference on Artificial Intelligence*, pages 457–457, 1977.

C. P. Gomes, B. Selman, N. Crato, and H. Kautz.
Heavy-tailed phenomena in satisfiability and constraint satisfaction problems.
Journal of Automated Reasoning, 24(1–2):67–100, 2000.

Carla P. Gomes, Ashish Sabharwal, and Bart Selman.
Model counting.
In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*. IOS Press, 2009.

References VIII

Hana Galperin and Avi Wigderson.
Succinct representations of graphs.
Information and Control, 56:183–198, 1983.
See [Loz88] for a correction.

Jinbo Huang and Adnan Darwiche.
The language of search.
Journal of Artificial Intelligence Research, 29:191–219, 2007.

Edward A. Hirsch.
New worst-case upper bounds for SAT.
Journal of Automated Reasoning, 24(4):397–420, 2000.

Federico Heras, Javier Larrosa, and Albert Oliveras.
MiniMaxSAT: An efficient weighted Max-SAT solver.
Journal of Artificial Intelligence Research, 31(1):1–32, 2008.

Stefan Heinz and Martin Sachenbacher.
Using model counting to find optimal distinguishing tests.
In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 117–131. Springer-Verlag, 2009.

Jinbo Huang.
The effect of restarts on the efficiency of clause learning.
In Manuela Veloso, editor, *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 2318–2323. AAAI Press, 2007.

References IX

Hans Kleine Büning, Marek Karpinski, and Andreas Flögel.
Resolution for quantified Boolean formulas.
Information and Computation, 117:12–18, 1995.

Kalev Kask and Rina Dechter.
Stochastic local search for Bayesian networks.
In *Seventh International Workshop on Artificial Intelligence and Statistics*. Morgan Kaufmann Publishers, 1999.

Victor Khomenko, Maciej Koutny, and Alex Yakovlev.
Logic synthesis for asynchronous circuits based on petri net unfoldings and incremental sat.
In *Application of Concurrency to System Design, 2004. ACSD 2004. Proceedings. Fourth International Conference on*, pages 16–25. IEEE, 2004.

Andreas Kuehlmann, Viresh Paruthi, Florian Krohm, and Malay K. Ganai.
Robust boolean reasoning for equivalence checking and functional property verification.
Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 21(12):1377–1394, 2002.

Henry Kautz and Bart Selman.
Planning as satisfiability.
In Bernd Neumann, editor, *Proceedings of the 10th European Conference on Artificial Intelligence*, pages 359–363. John Wiley & Sons, 1992.

References X

Henry Kautz and Bart Selman.
Pushing the envelope: planning, propositional logic, and stochastic search.
In *Proceedings of the 13th National Conference on Artificial Intelligence and the 8th Innovative Applications of Artificial Intelligence Conference*, pages 1194–1201. AAAI Press, 1996.

Lukas Kroc, Ashish Sabharwal, and Bart Selman.
Leveraging belief propagation, backtrack search, and statistics for model counting.
Annals of Operations Research, 184(1):209–231, 2011.

Tracy Larrabee.
Test pattern generation using Boolean satisfiability.
Transactions on Computer-Aided Design of Integrated Circuits and Systems, 11(1):4–15, 1992.

Antonio Lozano and José L. Balcázar.
The complexity of graph problems for succinctly represented graphs.
In Manfred Nagl, editor, *Graph-Theoretic Concepts in Computer Science, 15th International Workshop, WG'89*, number 411 in Lecture Notes in Computer Science, pages 277–286. Springer-Verlag, 1990.

T. Latvala, A. Biere, K. Heljanko, and T. Junttila.
Simple bounded LTL model-checking.
In *Proceedings Intl. Conf. on Formal Methods in Computer-Aided Design, FMCAD'04*, pages 186–200, 2004.

References XI

Michael L. Littman.
Initial experiments in stochastic satisfiability.
In *Proceedings of the 16th National Conference on Artificial Intelligence (AAAI-99) and the 11th Conference on Innovative Applications of Artificial Intelligence (IAAI-99)*, pages 667–672. AAAI Press / The MIT Press, 1999.

Pey-Chang Lin and Sunil Khatri.
Application of Max-SAT-based ATPG to optimal cancer therapy design.
BMC Genomics, 13(Suppl 6):S5, 2012.

Chu Min Li, Felip Manyà, Nouredine Ould Mohamedou, and Jordi Planes.
Resolution-based lower bounds in MaxSAT.
Constraints, 15(4):456–484, 2010.

Michael L. Littman, Stephen M. Majercik, and Toniann Pitassi.
Stochastic Boolean satisfiability.
Journal of Automated Reasoning, 27(3):251–296, 2001.

Inês Lynce and João Marques-Silva.
Efficient haplotype inference with boolean satisfiability.
In *Proceedings of the 21th National Conference on Artificial Intelligence (AAAI-2006)*, pages 104–109. AAAI Press, 2006.

References XII

Antonio Lozano.
NP-hardness of succinct representations of graphs.
Bulletin of the European Association for Theoretical Computer Science, 35:158–163, June 1988.

Stephen M. Majercik and Byron Boots.
DC-SSAt: a divide-and-conquer approach to solving stochastic satisfiability problems efficiently.
In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI-2005)*, pages 416–422. AAAI Press / The MIT Press, 2005.

Stephen M. Majercik and Michael L. Littman.
MAXPLAN: A new approach to probabilistic planning.
In Reid Simmons, Manuela Veloso, and Stephen Smith, editors, *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems*, pages 86–93, Pittsburgh, Pennsylvania, 1998.

Stephen M. Majercik and Michael L. Littman.
Contingent planning under uncertainty via stochastic satisfiability.
Artificial Intelligence, 147(1-2):119–162, 2003.

Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik.
Chaff: engineering an efficient SAT solver.
In *Proceedings of the 38th ACM/IEEE Design Automation Conference (DAC'01)*, pages 530–535. ACM Press, 2001.

References XIII

João P. Marques-Silva and Karem A. Sakallah.
Conflict analysis in search algorithms for propositional satisfiability.
In *Proceedings of the IEEE International Conference on Tools with Artificial Intelligence*, pages 467–469, 1996.

Christos H. Papadimitriou.
Games against nature.
Journal for Computer and System Sciences, 31:288–301, 1985.

James D. Park.
Using weighted max-sat engines to solve mpe.
In *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI-2002) and the 14th Conference on Innovative Applications of Artificial Intelligence (IAAI-2002)*, pages 682–687. AAAI Press / The MIT Press, 2002.

James .D. Park and Adnan Darwiche.
Complexity results and approximation strategies for map explanations.
Journal of Artificial Intelligence Research, 21(1):101–133, 2004.

K. Pipatsrisawat and A. Darwiche.
On the power of clause-learning SAT solvers with restarts.
In I. P. Gent, editor, *Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming, CP 2009*, number 5732 in Lecture Notes in Computer Science, pages 654–668. Springer-Verlag, 2009.

References XIV

Knot Pipatsrisawat and Adnan Darwiche.
A new d-DNNF-based bound computation algorithm for functional E-MAJSAT.
In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, pages 590–595. Morgan Kaufmann Publishers, 2009.

J. Pearl.
Probabilistic semantics for nonmonotonic reasoning: a survey.
In R. J. Brachman, H. J. Levesque, and R. Reiter, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the First International Conference (KR '89)*, pages 505–516. Morgan Kaufmann Publishers, 1989.
Reprinted in *Readings in Uncertain Reasoning*, G. Shafer and J. Pearl (eds.), Morgan Kaufmann Publishers, San Francisco, California, 1990, pp. 699–710.

Nathan Robinson, Charles Gretton, Duc-Nghia Pham, and Abdul Sattar.
Partial weighted MaxSAT for optimal planning.
In Byoung-Tak Zhang and Mehmet A. Orgun, editors, *PRICAI 2010: Trends in Artificial Intelligence*, volume 6230 of *Lecture Notes in Computer Science*, pages 231–243. Springer-Verlag, 2010.

Masood Feyzbakhsh Rankooh and Gholamreza Ghassem-Sani.
New encoding methods for sat-based temporal planning.
In *ICAPS 2013. Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling*, 2013.

References XV

Jussi Rintanen, Keijo Heljanko, and Ilkka Niemelä.
Planning as satisfiability: parallel plans and algorithms for plan search.
Artificial Intelligence, 170(12-13):1031–1080, 2006.

Jussi Rintanen.
Evaluation strategies for planning as satisfiability.
In Ramon López de Mántaras and Lorenza Saitta, editors, *ECAI 2004. Proceedings of the 16th European Conference on Artificial Intelligence*, pages 682–687. IOS Press, 2004.

Jussi Rintanen.
Heuristics for planning with SAT.
In David Cohen, editor, *Principles and Practice of Constraint Programming - CP 2010, 16th International Conference, CP 2010, St. Andrews, Scotland, September 2010, Proceedings.*, number 6308 in Lecture Notes in Computer Science, pages 414–428. Springer-Verlag, 2010.

Jussi Rintanen.
Engineering efficient planners with SAT.
In *ECAI 2012. Proceedings of the 20th European Conference on Artificial Intelligence*, pages 684–689. IOS Press, 2012.

Jussi Rintanen.
Planning as satisfiability: heuristics.
Artificial Intelligence, 193:45–86, 2012.

References XVI

Tian Sang, Fahiem Bacchus, Paul Beame, Henry Kautz, and Toniann Pitassi.
Combining component caching and clause learning for effective model counting.
In *Theory and Applications of Satisfiability Testing, 7th International Conference, SAT 2004, Vancouver, BC, Canada, May 10-13, 2004, Revised Selected Papers*, 2004.

Tian Sang, Paul Beame, and Henry Kautz.
Heuristics for fast exact model counting.
In *Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing (SAT-2005)*, pages 226–240. Springer-Verlag, 2005.

Tian Sang, Paul Beame, and Henry Kautz.
Performing Bayesian inference by weighted model counting.
In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI-2005)*, pages 475–482. AAAI Press / The MIT Press, 2005.

Ji-Ae Shin and Ernest Davis.
Processes and continuous change in a SAT-based planner.
Artificial Intelligence, 166(1):194–253, 2005.

Carsten Sinz.
Towards an optimal cnf encoding of boolean cardinality constraints.
In *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming, Sitges, Spain, October 2005.*, pages 827–831. Springer-Verlag, 2005.

References XVII

Matthew Streeter and Stephen F. Smith.
Using decision procedures efficiently for optimization.
In *ICAPS 2007. Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling*, pages 312–319. AAAI Press, 2007.

Roberto Sebastiani and Silvia Tomasi.
Optimization in SMT with LA($\mathcal{Q}$) cost functions.
In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, *Automated Reasoning*, volume 7364 of *Lecture Notes in Computer Science*, pages 484–498. Springer-Verlag, 2012.

L. J. Stockmeyer.
The polynomial-time hierarchy.
Theoretical Computer Science, 3(1):1–22, 1976.

Alexander Smith, Andreas Veneris, Moayad Fahim Ali, and Anastasios Viglas.
Fault diagnosis and logic debugging using Boolean satisfiability.
Transactions on Computer-Aided Design of Integrated Circuits and Systems, 24(10), 2005.

Tino Teige, Andreas Eggers, and Martin Fränzle.
Constraint-based analysis of concurrent probabilistic hybrid systems: An application to networked automation systems.
Nonlinear Analysis: Hybrid Systems, 5(2):343–366, 2011.

References XVIII

Tino Teige and Martin Fränzle.
Resolution for stochastic boolean satisfiability.
In *Logic for Programming, Artificial Intelligence, and Reasoning*, pages 625–639. Springer-Verlag, 2010.

Tino Teige and Martin Fränzle.
Generalized Craig interpolation for stochastic Boolean satisfiability problems.
In *Tools and Algorithms for the Construction and Analysis of Systems*, number 6605 in Lecture Notes in Computer Science, pages 158–172. Springer-Verlag, 2011.

Leslie G. Valiant.
The complexity of enumeration and reliability problems.
SIAM Journal on Computing, 8(3):410–421, 1979.

Robert Wille, Daniel Große, D. Michael Miller, and Rolf Drechsler.
Equivalence checking of reversible circuits.
In *Multiple-Valued Logic, 2009. ISMVL'09. 39th International Symposium on*, pages 324–330. IEEE, 2009.

Steven A. Wolfman and Daniel S. Weld.
The LPSAT engine & its application to resource planning.
In Thomas Dean, editor, *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pages 310–315. Morgan Kaufmann Publishers, 1999.

References XIX

Emmanuel Zarpas.
Simple yet efficient improvements of SAT based bounded model checking.
In Alan J. Hu and Andrew K. Martin, editors, *Formal Methods in Computer-Aided Design: 5th International Conference, FMCAD 2004, Austin, Texas, USA, November 15-17, 2004. Proceedings*, number 3312 in Lecture Notes in Computer Science, pages 174–185. Springer-Verlag, 2004.