Advanced Satisfiability

Mausam

(Based on slides of Carla Gomes, Henry Kautz, Subbarao Kambhampati, Cristopher Moore, Ashish Sabharwal, Bart Selman, Toby Walsh)
Why study Satisfiability?

- Canonical NP complete problem.
  - several hard problems modeled as SAT
- Tonne of applications
- State-of-the-art solvers superfast
Real-World Reasoning
Tackling inherent computational complexity

Example domains cast in propositional reasoning system (variables, rules).

- High-Performance Reasoning
- Temporal/uncertainty reasoning
- Strategic reasoning/Multi-player

Technology Targets

No. of atoms on earth $10^{47}$
Seconds until heat death of sun
Protein folding calculation (petaflop-year)

Variables

- Car repair diagnosis
  - 100
  - 200

- Deep space mission control
  - 10K
  - 50K

- Chess
  - 50K
  - 200K

- Deep space mission control
  - 200K
  - 600K

- Military Logistics
  - 200K
  - 50K

- Hardware/Software Verification
  - 0.5M
  - 1M

- Multi-Agent Systems
  - 1M
  - 5M

DARPA Research Program
Application: Diagnosis

• Problem: diagnosis a malfunctioning device
  – Car
  – Computer system
  – Spacecraft

• where
  – Design of the device is known
  – We can observe the state of only certain parts of the device – much is hidden
Model-Based, Consistency-Based Diagnosis

• Idea: create a logical formula that describes how the device **should** work
  – Associated with each “breakable” component C is a proposition that states “C is okay”
  – Sub-formulas about component C are all conditioned on C being okay

• A **diagnosis** is a smallest of “not okay” assumptions that are consistent with what is actually observed
Consistency-Based Diagnosis

1. Make some Observations O.
2. Initialize the Assumption Set A to assert that all components are working properly.
3. Check if the KB, A, O together are inconsistent (can deduce false).
4. If so, delete propositions from A until consistency is restored (cannot deduce false). The deleted propositions are a diagnosis.

*There may be many possible diagnoses*
Example: Automobile Diagnosis

- **Observable Propositions:**
  - EngineRuns,
  - GasInTank,
  - ClockRuns

- **Assumable Propositions:**
  - FuelLineOK,
  - BatteryOK,
  - CablesOK,
  - ClockOK

- **Hidden (non-Assumable) Propositions:**
  - GasInEngine,
  - PowerToPlugs

- **Device Description F:**
  - $(\text{GasInTank} \land \text{FuelLineOK}) \rightarrow \text{GasInEngine}$
  - $(\text{GasInEngine} \land \text{PowerToPlugs}) \rightarrow \text{EngineRuns}$
  - $(\text{BatteryOK} \land \text{CablesOK}) \rightarrow \text{PowerToPlugs}$
  - $(\text{BatteryOK} \land \text{ClockOK}) \rightarrow \text{ClockRuns}$

- **Observations:**
  - $\neg \text{EngineRuns, GasInTank, ClockRuns}$
Example

- Is $F \cup \text{Observations} \cup \text{Assumptions}$ consistent?
Example

• Is $F \cup \text{Observations} \cup \text{Assumptions}$ consistent?

• $F \cup \{ \neg \text{EngineRuns}, \text{GasInTank}, \text{ClockRuns} \}
\cup \{ \text{FuelLineOK, BatteryOK, CablesOK, ClockOK} \} \rightarrow false$
  
  – Must restore consistency!
Example

• Is $F \cup \text{Observations} \cup \text{Assumptions}$ consistent?

• $F \cup \{\neg\text{EngineRuns, GasInTank, ClockRuns}\}$
  $\cup \{\text{FuelLineOK, BatteryOK, CablesOK, ClockOK}\} \rightarrow false$
  – Must restore consistency!

• $F \cup \{\neg\text{EngineRuns, GasInTank, ClockRuns}\}$
  $\cup \{\text{BatteryOK, CablesOK, ClockOK}\} \rightarrow false$
  – $\neg\text{FuelLineOK is a diagnosis}$
Example

• *Is* \( F \cup \text{Observations} \cup \text{Assumptions} \) *consistent*?

• \( F \cup \{ \neg \text{EngineRuns, GasInTank, ClockRuns} \} \)
\( \cup \{ \text{FuelLineOK, BatteryOK, CablesOK, ClockOK} \} \) \( \rightarrow false \)
  – *Must restore consistency!*

• \( F \cup \{ \neg \text{EngineRuns, GasInTank, ClockRuns} \} \)
\( \cup \{ \text{BatteryOK, CablesOK, ClockOK} \} \) \( \rightarrow false \)
  – \( \neg \text{FuelLineOK is a diagnosis} \)

• \( F \cup \{ \neg \text{EngineRuns, GasInTank, ClockRuns} \} \)
\( \cup \{ \text{FuelLineOK, CablesOK, ClockOK} \} \) \( \rightarrow false \)
  – \( \neg \text{BatteryOK is not a diagnosis} \)
Complexity of Diagnosis

• If F is Horn, then each consistency test takes linear time – unit propagation is complete for Horn clauses.

• Complexity = ways to delete propositions from Assumption Set that are considered.
  – Single fault diagnosis – $O(n^2)$
  – Double fault diagnosis – $O(n^3)$
  – Triple fault diagnosis – $O(n^4)$
  ...

Deep Space One

- Autonomous diagnosis & repair “Remote Agent”
- Compiled systems schematic to 7,000 var SAT problem

Started: January 1996
Launch: October 15th, 1998
Experiment: May 17-21
Deep Space One

• a failed electronics unit
  – Remote Agent fixed by reactivating the unit.

• a failed sensor providing false information
  – Remote Agent recognized as unreliable and therefore correctly ignored.

• an altitude control thruster (a small engine for controlling the spacecraft's orientation) stuck in the "off" position
  – Remote Agent detected and compensated for by switching to a mode that did not rely on that thruster.
Testing Circuit Equivalence

- Do two circuits compute the same function?
- Circuit optimization
- Is there input for which the two circuits compute different values?
Testing Circuit Equivalence

\[ C \equiv (A \lor B) \]
\[ C' \equiv (D \land E) \]
\[ D \equiv \neg A \]
\[ E \equiv \neg B \]
\[ \neg(C \equiv C') \]
SAT Translation of N-Queens
SAT Translation of N-Queens

- At least one queen each column:
  \[(Q_{11} \lor Q_{12} \lor Q_{13} \lor \ldots \lor Q_{18})\]
  \[(Q_{21} \lor Q_{22} \lor Q_{23} \lor \ldots \lor Q_{28})\]
  ...

SAT Translation of N-Queens

• At least one queen each column:
  \[(Q_{11} \lor Q_{12} \lor Q_{13} \lor \ldots \lor Q_{18})\]
  \[(Q_{21} \lor Q_{22} \lor Q_{23} \lor \ldots \lor Q_{28})\]
  \[
  \ldots
  \]

• No attacks:
  \[(\neg Q_{11} \lor \neg Q_{12})\]
  \[(\neg Q_{11} \lor \neg Q_{22})\]
  \[(\neg Q_{11} \lor \neg Q_{21})\]
  \[
  \ldots
  \]
CSP $\rightarrow$ SAT

- A new SAT Variable for var-val pair $X_{WA-r}, X_{WA-g}, X_{WA-b}, X_{NT-r}$...
- Each var has at least 1 value
  - $X_{WA-r} \lor X_{WA-g} \lor X_{WA-b}$
- No var has two values
  - $\lnot X_{WA-r} \lor \lnot X_{WA-g}$
  - $\lnot X_{WA-r} \lor \lnot X_{WA-b}$
- Constraints
  - $\lnot X_{WA-r} \lor \lnot X_{NT-r}$
Symbolic Model Checking

• Any finite state machine is characterized by a transition function
  – CPU
  – Networking protocol
• Wish to prove some invariant holds for any possible inputs
• Bounded model checking: formula is sat iff invariant fails $k$ steps in the future

$$\overline{S}_t = \text{vector of Booleans representing state of machine at time } t$$

$$\rho : \text{State} \times \text{Input} \rightarrow \text{State}$$

$$\gamma : \text{State} \rightarrow \{0, 1\}$$

$$\left( \bigwedge_{i=0}^{k-1} \left( \overline{S}_{i+1} \equiv \rho(\overline{S}_i, \overline{I}_i) \right) \right) \land S_o \land \neg \gamma(\overline{S}_k)$$
A “real world” example

From “SATLIB”:

http://www.satlib.org/benchm.html

SAT-encoded bounded model checking instances
contributed by Ofer Shtrichman

In Bounded Model Checking (BMC) [BCCZ99], a rather newly introduced problem in formal methods, the task is to check whether a given model $M$ (typically a hardware design) satisfies a temporal property $P$ in all paths with length less or equal to some bound $k$. The BMC problem can be efficiently reduced to a propositional satisfiability problem, and in fact if the property is in the form of an invariant (invariants are the most common type of properties, and many other temporal properties can be reduced to their form. It has the form of 'it is always true that ... '), it has a structure which is similar to many AI planning problems.
Bounded Model Checking instance

The instance bmc-ibm-6.cnf, IBM LSU 1997:

\[ p \text{ cnf } 51639 \text{ 368352} \]
\[ -1 \text{ 7 0} \]
\[ -1 \text{ 6 0} \]
\[ -1 \text{ 5 0} \]
\[ -1 \text{ -4 0} \]
\[ -1 \text{ 3 0} \]
\[ -1 \text{ 2 0} \]
\[ -1 \text{ -8 0} \]
\[ -9 \text{ 15 0} \]
\[ -9 \text{ 14 0} \]
\[ -9 \text{ 13 0} \]
\[ -9 \text{ -12 0} \]
\[ -9 \text{ 11 0} \]
\[ -9 \text{ 10 0} \]
\[ -9 \text{ -16 0} \]
\[ -17 \text{ 23 0} \]
\[ -17 \text{ 22 0} \]

\text{i.e. } ((\text{not } x_1) \text{ or } x_7) \text{ and } ((\text{not } x_1) \text{ or } x_6) \text{ and } \ldots \text{ etc.}
10 pages later:

\[ x_{177} \text{ or } x_{169} \text{ or } x_{161} \text{ or } x_{153} \ldots \text{ or } x_{17} \text{ or } x_9 \text{ or } x_1 \text{ or (not } x_{185}) \]

clauses / constraints are getting more interesting...
4000 pages later:

!!!

a 59-cnf clause...

```
10236  -10050  0
10236  -10051  0
10236  -10235  0
10008  10009  10010  10011  10012  10013  10014
10015  10016  10017  10018  10019  10020  10021
10022  10023  10024  10025  10026  10027  10028
10029  10030  10031  10032  10033  10034  10035
10036  10037  10038  10039  10040  10041  10042
10043  10044  10045  10046  10047  10048  10049
10050  10051  10235  -10236  0
10237  -10008  0
10237  -10009  0
10237  -10010  0
...```

...
Finally, 15,000 pages later:

\[-7 \ 260 \ 0\]
\[7 \ -260 \ 0\]
\[1072 \ 1070 \ 0\]
\[-15 \ -14 \ -13 \ -12 \ -11 \ -10 \ 0\]
\[-15 \ -14 \ -13 \ -12 \ -11 \ 10 \ 0\]
\[-15 \ -14 \ -13 \ -12 \ 11 \ -10 \ 0\]
\[-15 \ -14 \ -13 \ -12 \ 11 \ 10 \ 0\]
\[-7 \ -6 \ -5 \ -4 \ -3 \ -2 \ 0\]
\[-7 \ -6 \ -5 \ -4 \ -3 \ 2 \ 0\]
\[-7 \ -6 \ -5 \ -4 \ 3 \ -2 \ 0\]
\[-7 \ -6 \ -5 \ -4 \ 3 \ 2 \ 0\]
\[185 \ 0\]

Note that: \[2^{50000} \approx 3.160699437 \cdot 10^{15051}\] ... !!!
Finally, 15,000 pages later:

\[-7 \; 260 \; 0 \\
7 \; -260 \; 0 \\
1072 \; 1070 \; 0 \\
-15 \; -14 \; -13 \; -12 \; -11 \; -10 \; 0 \\
-15 \; -14 \; -13 \; -12 \; -11 \; 10 \; 0 \\
-15 \; -14 \; -13 \; -12 \; 11 \; -10 \; 0 \\
-15 \; -14 \; -13 \; -12 \; 11 \; 10 \; 0 \\
-7 \; -6 \; -5 \; -4 \; -3 \; -2 \; 0 \\
-7 \; -6 \; -5 \; -4 \; -3 \; 2 \; 0 \\
-7 \; -6 \; -5 \; -4 \; 3 \; -2 \; 0 \\
-7 \; -6 \; -5 \; -4 \; 3 \; 2 \; 0 \\
185 \; 0 \]

Note that: \( 2^{50000} \approx 3.160699437 \cdot 10^{15051} \) ... !!!

The Chaff SAT solver (Princeton) solves this instance in less than one minute.
Finally, 15,000 pages later:

\[-7 \ 260 \ 0
7 \ -260 \ 0
1072 \ 1070 \ 0
-15 \ -14 \ -13 \ -12 \ -11 \ -10 \ 0
-15 \ -14 \ -13 \ -12 \ -11 \ 10 \ 0
-15 \ -14 \ -13 \ -12 \ 11 \ -10 \ 0
-15 \ -14 \ -13 \ -12 \ 11 \ 10 \ 0
-7 \ -6 \ -5 \ -4 \ -3 \ -2 \ 0
-7 \ -6 \ -5 \ -4 \ -3 \ 2 \ 0
-7 \ -6 \ -5 \ -4 \ 3 \ -2 \ 0
-7 \ -6 \ -5 \ -4 \ 3 \ 2 \ 0
185 \ 0\]

What makes this possible?

Note that: \(2^{50000} \approx 3.160699437 \cdot 10^{15051}\) ... !!!

The Chaff SAT solver (Princeton) solves this instance in less than one minute.
Progress in Last 20 years

• *Significant progress since the 1990’s.* How much?
• Problem size: We went from 100 variables, 200 constraints (early 90’s) to 1,000,000+ variables and 5,000,000+ constraints in 20 years

• Search space: from $10^{30}$ to $10^{300,000}$.
  [Aside: “one can encode quite a bit in 1M variables.”]

• Is this just Moore’s Law? It helped, but not much...
• – 2x faster computers does *not* mean can solve 2x larger instances
• – search difficulty does not scale linearly with problem size!
• **Tools**: 50+ competitive SAT solvers available
Forces Driving Faster, Better SAT Solvers

• From academically interesting to practically relevant “Real” benchmarks, with real interest in solving them

• Regular SAT Solver Competitions (Germany-89, Dimacs-93, China-96, SAT-02, SAT-03, ..., SAT-07, SAT-09, SAT-2011)
  – A tremendous resource! E.g., SAT Competition 2006 (Seattle):
    • 35+ solvers submitted, downloadable, mostly open source
    • 500+ industrial benchmarks, 1000+ other benchmarks
    • 50,000+ benchmark instances available on the Internet

• This constant improvement in SAT solvers is the key to the success of, e.g., SAT-based planning and verification
Assignment 2: Graph Subset Mapping

• Given two directed graphs G and G’
  – Check if G is a subset mapping to G’

• I.e. construct a one-one mapping (M) from all nodes of G to some nodes of G’ s.t.
  – (n1,n2) in G \(\rightarrow\) (M(n1), M(n2)) in G’
  – (n1,n2) not in G \(\rightarrow\) (M(n1), M(n2)) not in G’
Graph $G$

Graph $G'$

A → B → C

P → Q → R

S

P → Q → R → S

A → B → C

P → Q → R
No, because the directionality of edges doesn’t match.
**Graph G**  

No, because the directionality of edges doesn’t match.

**Graph G’**  

No, because there is no edge between A and C in G whereas there is one between P and R in G’.
Graph $G$  

Graph $G'$

No, because the directionality of edges doesn’t match.

No, because there is no edge between A and C in $G$ whereas there is one between P and R in $G'$.

Yes. A mapping is: $M(A) = S$, $M(B) = Q$, $M(C) = R$

The edges from P to other nodes don’t matter since no node in $G$ got mapped to P.
SAT Model for Graph Subset Mapping

• If a mapping exists then SAT formula is satisfiable
• Else unsatisfiable

• The satisfying assignment suggests the mapping $M$
GSAT

- **Local** search (Hill Climbing + Random Walk) over space of *complete* truth assignments
  - With prob $p$: flip any variable in any unsatisfied clause
  - With prob $(1-p)$: flip *best* variable in any unsat clause
    - best = one which minimizes #unsatisfied clauses

- SAT encodings of N-Queens, scheduling
- Best algorithm for random K-SAT
  - Best DPLL: 700 variables
  - Walksat: 100,000 variables
Refining Greedy Random Walk

• Each flip
  – makes some false clauses become true
  – breaks some true clauses, that become false

• Suppose \( s_1 \rightarrow s_2 \) by flipping \( x \). Then:
  \[
  \text{#unsat}(s_2) = \text{#unsat}(s_1) - \text{make}(s_1,x) + \text{break}(s_1,x)
  \]

• Idea 1: if a choice breaks nothing, it is very likely to be a good move

• Idea 2: near the solution, only the break count matters
  – the make count is usually 1
Walksat

state = random truth assignment;
while ! GoalTest(state) do
    clause := random member { C | C is false in state };
    for each x in clause do compute break[x];
    if exists x with break[x]=0 then var := x;
    else
        with probability p do
            var := random member { x | x is in clause };
        else
            var := arg x min { break[x] | x is in clause };
        endif
    state[var] := 1 – state[var];
end
return state;

Put everything inside of a restart loop.
Parameters: p, max_flips, max_runs
Hardness of 3-sat as a function of 
#clauses/#variables

- Probability that there is a satisfying assignment
- Cost of solving (either by finding a solution or showing there ain't one)
Hardness of 3-sat as a function of $\text{#clauses/#variables}$

- Probability that there is a satisfying assignment
- Cost of solving (either by finding a solution or showing there ain't one)
Hardness of 3-sat as a function of 
#clauses/#variables

- Probability that there is a satisfying assignment
- Cost of solving (either by finding a solution or showing there ain't one)
Hardness of 3-sat as a function of \#clauses/\#variables

- Probability that there is a satisfying assignment
- Cost of solving (either by finding a solution or showing there ain't one)
Hardness of 3-sat as a function of 
#clauses/#variables

- Probability that there is a satisfying assignment
- Cost of solving (either by finding a solution or showing there ain't one)
Hardness of 3-sat as a function of \#clauses/\#variables

- Probability that there is a satisfying assignment
- Cost of solving (either by finding a solution or showing there ain't one)
Hardness of 3-sat as a function of \#clauses/\#variables

- Probability that there is a satisfying assignment
- Cost of solving (either by finding a solution or showing there ain't one)
Hardness of 3-sat as a function of \#clauses/\#variables

- Probability that there is a satisfying assignment
- Cost of solving (either by finding a solution or showing there ain't one)

\( p=0.5 \)
Hardness of 3-sat as a function of \#clauses/\#variables

You would expect this

- **Cost of solving** (either by finding a solution or showing there ain't one)
- **Probability that there is a satisfying assignment**

\( p=0.5 \)
Hardness of 3-sat as a function of \#clauses/\#variables

You would expect this

\[ p=0.5 \]

- Green line: Probability that there is a satisfying assignment
- Red line: Cost of solving (either by finding a solution or showing there ain't one)
Hardness of 3-sat as a function of #clauses/#variables

You would expect this

- Probability that there is a satisfying assignment
- Cost of solving (either by finding a solution or showing there ain't one)

#clauses/#variables
Hardness of 3-sat as a function of 
#clauses/#variables

This is what happens!

- Probability that there is a satisfying assignment
- Cost of solving (either by finding a solution or showing there ain't one)

You would expect this

p=0.5
Hardness of 3-sat as a function of #clauses/#variables

This is what happens!

Probability that there is a satisfying assignment

Cost of solving (either by finding a solution or showing there ain't one)

You would expect this

p=0.5

~4.3

#clauses/#variables
Random 3-SAT

- Random 3-SAT
  - sample uniformly from space of all possible 3-clauses
  - $n$ variables, $l$ clauses

- Which are the hard instances?
  - around $l/n = 4.3$
Random 3-SAT

- Varying problem size, $n$

- Complexity peak appears to be largely invariant of algorithm
Random 3-SAT

- Complexity peak coincides with solubility transition
  - $l/n < 4.3$ problems under-constrained and SAT
  - $l/n > 4.3$ problems over-constrained and UNSAT
  - $l/n=4.3$, problems on “knife-edge” between SAT and UNSAT
Random 3-SAT as of 2005

Mitchell, Selman, and Levesque ’92
Random 3-SAT as of 2005

Phase transition

Mitchell, Selman, and Levesque ’92
Random 3-SAT as of 2005

Mitchell, Selman, and Levesque ’92
Random 3-SAT as of 2005

Random Walk

DP Calls

Ratio of Clauses-to-Variables

50 var
40 var
20 var

Mitchell, Selman, and Levesque ’92
Random 3-SAT as of 2005

Mitchell, Selman, and Levesque ’92
Random 3-SAT as of 2005

Mitchell, Selman, and Levesque ’92
Random 3-SAT as of 2005

Mitchell, Selman, and Levesque '92
Random 3-SAT as of 2005

Random Walk

DP

DP'

GSAT

Walksat

Mitchell, Selman, and Levesque ’92
Random Walk

Random 3-SAT as of 2005

Mitchell, Selman, and Levesque ’92
Results: Random 3-SAT
Results: Random 3-SAT

- Random walk up to ratio 1.36 (Alekhnovich and Ben Sasson 03). empirically up to 2.5
Results: Random 3-SAT

- **Random walk** up to ratio 1.36 (Alekhnovich and Ben Sasson 03).
  empirically up to 2.5
- **Davis Putnam (DP)** up to 3.42 (Kaporis et al. ’02) ‘
  empirically up to 3.6
  *approx. 400 vars at phase transition*
Results: Random 3-SAT

- **Random walk** up to ratio 1.36 (Alekhnovich and Ben Sasson 03).
  empirically up to 2.5
- **Davis Putnam (DP)** up to 3.42 (Kaporis et al. ’02) ‘
  empirically up to 3.6
  *approx. 400 vars at phase transition*
- **GSAT** up till ratio 3.92 (Selman et al. ’92, Zecchina et al. ‘02)
  *approx. 1,000 vars at phase transition*
Results: Random 3-SAT

- **Random walk** up to ratio 1.36 (Alekhnovich and Ben Sasson 03).
  empirically up to 2.5
- **Davis Putnam (DP)** up to 3.42 (Kaporis et al. ’02) ‘
  empirically up to 3.6
  *approx. 400 vars at phase transition*
- **GSAT** up till ratio 3.92 (Selman et al. ’92, Zecchina et al. ‘02)
  *approx. 1,000 vars at phase transition*
- **Walksat** up till ratio 4.1 (empirical, Selman et al. ’93)
  *approx. 100,000 vars at phase transition*
Results: Random 3-SAT

- **Random walk** up to ratio 1.36 (Alekhnovich and Ben Sasson 03).
  empirically up to 2.5
- **Davis Putnam (DP)** up to 3.42 (Kaporis et al. ’02) ‘
  empirically up to 3.6
  *approx. 400 vars at phase transition*
- **GSAT** up till ratio 3.92 (Selman et al. ’92, Zecchina et al. ‘02)
  *approx. 1,000 vars at phase transition*
- **Walksat** up till ratio 4.1 (empirical, Selman et al. ’93)
  *approx. 100,000 vars at phase transition*
- **Survey propagation (SP)** up till 4.2
  (empirical, Mezard, Parisi, Zecchina ’02)
  *approx. 1,000,000 vars near phase transition*
3SAT phase transition

• Upper bounds (easier)
  – Typically by estimating count of solutions
3SAT phase transition

• Upper bounds (easier)
  – Typically by estimating count of solutions
  – E.g. Markov (or 1st moment) method

For any statistic $X$

$$\text{prob}(X \geq 1) \leq E[X]$$

$$E[X] = 0 \cdot p(X=0) + 1 \cdot p(X=1) + 2 \cdot p(X=2) + 3 \cdot p(X=3) + \ldots$$

$$\geq 1 \cdot p(X=1) + 1 \cdot p(X=2) + 1 \cdot p(X=3) + \ldots$$

$$\geq p(X \geq 1)$$
3SAT phase transition

• Upper bounds (easier)
  – Typically by estimating count of solutions
  – E.g. Markov (or 1st moment) method

For any statistic $X$

$$\Pr(X \geq 1) \leq \mathbb{E}[X]$$

*No assumptions about the distribution of $X$ except non-negative!*
3SAT phase transition

• Upper bounds (easier)
  – Typically by estimating count of solutions
  – E.g. Markov (or 1st moment) method
    For any statistic $X$
    \[
    \text{prob}(X \geq 1) \leq E[X]
    \]
    Let $X$ be the number of satisfying assignments for a 3SAT problem
3SAT phase transition

• Upper bounds (easier)
  – Typically by estimating count of solutions
  – E.g. Markov (or 1st moment) method

For any statistic $X$

$$\text{prob}(X \geq 1) \leq E[X]$$

Let $X$ be the number of satisfying assignments for a 3SAT problem

*The expected value of $X$ can be easily calculated*
3SAT phase transition

- Upper bounds (easier)
  - Typically by estimating count of solutions
  - E.g. Markov (or 1st moment) method

For any statistic $X$

$$\text{prob}(X \geq 1) \leq E[X]$$

Let $X$ be the number of satisfying assignments for a 3SAT problem

$$E[X] = 2^n \times (7/8)^l$$
3SAT phase transition

• Upper bounds (easier)
  – Typically by estimating count of solutions
  – E.g. Markov (or 1st moment) method

For any statistic $X$

$$\text{prob}(X \geq 1) \leq E[X]$$

Let $X$ be the number of satisfying assignments for a 3SAT problem

$$E[X] = 2^n \times (7/8)^l$$

If $E[X] < 1$, then $\text{prob}(X \geq 1) = \text{prob}(\text{SAT}) < 1$
3SAT phase transition

• Upper bounds (easier)
  – Typically by estimating count of solutions
  – E.g. Markov (or 1st moment) method

For any statistic $X$

$$\text{prob}(X \geq 1) \leq E[X]$$

Let $X$ be the number of satisfying assignments for a 3SAT problem

$$E[X] = 2^n \times (7/8)^l$$

If $E[X] < 1$, then $2^n \times (7/8)^l < 1$
3SAT phase transition

• Upper bounds (easier)
  – Typically by estimating count of solutions
  – E.g. Markov (or 1st moment) method

For any statistic $X$

$$\text{prob}(X \geq 1) \leq E[X]$$

Let $X$ be the number of satisfying assignments for a 3SAT problem

$$E[X] = 2^n \times (7/8)^l$$

If $E[X] < 1$, then $2^n \times (7/8)^l < 1$

$$n + l \log_2(7/8) < 0$$
3SAT phase transition

• Upper bounds (easier)
  – Typically by estimating count of solutions
  – E.g. Markov (or 1st moment) method

For any statistic $X$

$$\text{prob}(X \geq 1) \leq E[X]$$

Let $X$ be the number of satisfying assignments for a 3SAT problem

$$E[X] = 2^n \times (7/8)^l$$

If $E[X] < 1$, then

$$2^n \times (7/8)^l < 1$$

$$n + l \log_2(7/8) < 0$$

$$l/n > 1/\log_2(8/7) = 5.19...$$
A Heavy Tail

- But the transition is much lower at l/n ~ 4.27. What going on?
A Heavy Tail

• But the transition is much lower at $l/n \sim 4.27$. What going on?

• In the range $4.27 < l/n < 5.19$,
  – the average no. of solutions is exponentially large.
• Occasionally, there are exponentially many...
  – ...but most of the time there are none!
• Large average doesn’t prove satisfiability!
Random 3-SAT as of 2004

- Random Walk
- DP
- DP'
- GSAT
- Walksat
- SP

Ratio of Clauses-to-Variables

DP Calls

50 var
40 var
20 var
Random 3-SAT as of 2004

Upper bounds by combinatorial arguments (’92 – ’05)
Random 3-SAT as of 2004

Upper bounds by combinatorial arguments (‘92 – ’05)
Random 3-SAT as of 2004

Upper bounds by combinatorial arguments (‘92 – ’05)

Random Walk

DP

DP'

GSAT

Walksat

SP

5.19
Random 3-SAT as of 2004

Upper bounds by combinatorial arguments ('92 – '05)
Random 3-SAT as of 2004

Upper bounds by combinatorial arguments (‘92 – ’05)
Random 3-SAT as of 2004

Upper bounds by combinatorial arguments ('92 – '05)
Random 3-SAT as of 2004

Upper bounds by combinatorial arguments ('92 – ’05)
Random 3-SAT as of 2004

Upper bounds by combinatorial arguments (‘92 – ’05)

Random Walk

DP

DP’

GSAT

Walksat

SP

5.19

5.081

4.762

4.643

4.601

4.596

Ratio of Clauses-to-Variables

DP Calls

50 var

40 var

20 var

4000

3000

2000

1000

0

2 3 4 5 6 7 8

4.76
Random 3-SAT as of 2004

Upper bounds by combinatorial arguments (‘92 – ’05)

Ratio of Clauses-to-Variables

DP Calls

5.19
5.081
4.762
4.643
4.601
4.596
4.506

Random Walk
DP
DP’
GSAT
Walksat
SP

Upper bounds by combinatorial arguments (‘92 – ’05)
Real versus Random

• Real graphs tend to be sparse
  – dense random graphs contain lots of (rare?) structure

• Real graphs tend to have short path lengths
  – as do random graphs

• Real graphs tend to be clustered
  – unlike sparse random graphs
Small world graphs

• Sparse, clustered, short path lengths

• Six degrees of separation
  – Stanley Milgram’s famous 1967 postal experiment
  – recently revived by Watts & Strogatz
  – shown applies to:
    • actors database
    • US electricity grid
    • neural net of a worm
    • ...
An example

• 1994 exam timetable at Edinburgh University
  – 59 nodes, 594 edges so relatively sparse
  – but contains 10-clique
• less than $10^{-10}$ chance in a random graph
  – assuming same size and density
• clique totally dominated cost to solve problem
Observation: Complete backtrack style search SAT solvers (e.g. DPLL) display a remarkably wide range of run times. Even when repeatedly solving the same problem instance; variable branching is choice randomized.

Run time distributions are often “heavy-tailed”.

Orders of magnitude difference in run time on different runs.

(Gomes et al. 1998; 2000)
Heavy Tails on Structured Problems

![Graph showing unsolved fraction vs. number of backtracks](image-url)
Heavy Tails on Structured Problems

50% runs:
1 backtrack

![Graph showing unsolved fraction versus number of backtracks for different constrained scenarios.](image)
Heavy Tails on Structured Problems

50% runs: 1 backtrack

10% runs: > 100,000 backtracks
Randomized Restarts

Solution: randomize the backtrack strategy
   Add noise to the heuristic branching (variable choice) function
   Cutoff and restart search after a fixed number of backtracks

Provably Eliminates heavy tails

In practice: rapid restarts with low cutoff can dramatically improve performance (Gomes et al. 1998, 1999)

Exploited in many current SAT solvers combined with clause learning and non-chronological backtracking. (e.g., Chaff etc.)
Restarts

1 - F(x)

Unsolved fraction

Number backtracks (log)

no restarts

restart every 4 backtracks
Restarts

- 70% unsolved
- 1 - F(x)
- Unsolvable fraction
- Number backtracks (log)
- No restarts
- Restart every 4 backtracks

70% unsolved
Restarts

- 70% unsolved
- 0.001% unsolved
- No restarts
- Restart every 4 backtracks
Restarts

- 70% unsolved
- Restart every 4 backtracks
- No restarts
- Unsolved fraction
- 0.001% unsolved
- 250 (62 restarts)
Example of Rapid Restart Speedup

![Graph showing the relationship between cutoff (log) and number of backtracks (log). The graph indicates a rapid speedup as the cutoff increases.]
Example of Rapid Restart Speedup

![Graph showing the relationship between Cutoff (log) and Number backtracks (log)]

- Number backtracks (log) on the y-axis.
- Cutoff (log) on the x-axis.
- The plot shows a sharp decrease followed by a gradual increase in number backtracks as the cutoff increases.
- A point labeled with 100000 and marked with a label ~10 restarts indicates a significant speedup.

Logarithmic scales are used for both axes to represent the data.
Example of Rapid Restart Speedup

![Graph showing the relationship between cutoff and the number of backtracks, with log scales on both axes. The graph illustrates a significant speedup with rapid restarts.]
Several ways to use restarts

- **Restart with increasing cutoff** - cutoff increases linearly
  - Geometric restarts – (Walsh 99) cutoff increased geometrically;

- **Randomized backtracking** – (Lynce et al 2001)
  - randomizes the target decision points when backtracking (several variants)

- **Random jumping** (Zhang 2002)
  - solver randomly jumps to unexplored portions of search space;
  - jumping decisions are based on analyzing the ratio between the space searched vs. the remaining search space;
  - solved several open problems in combinatorics;

- **Learning restart strategies** – (Kautz et al 2001 and Ruan et. al 2002) –
  - results on optimal policies for restarts under particular scenarios. Huge area for further research.
Intuitively: Exponential penalties hidden in backtrack search, consisting of large inconsistent subtrees in the search space.

But, for restarts to be effective, you also need short runs.
Intuitively: Exponential penalties hidden in backtrack search, consisting of large inconsistent subtrees in the search space.

But, for restarts to be effective, you also need short runs.

Where do short runs come from?
Real World Problems are characterized by Hidden Tractable Substructure

**BACKDOORS**
Subset of “critical” variables such that once assigned a value the instance simplifies to a tractable class.

Explain how a solver can get “lucky” and solve very large instances
Backdoors

Informally:

A backdoor to a given problem is a subset of the variables such that once they are assigned values, the polynomial propagation mechanism of the SAT solver solves the remaining formula.

Formal definition includes the notion of a “subsolver”: a polynomial simplification procedure with certain general characteristics found in current DPLL SAT solvers.

**Backdoors correspond to “clever reasoning shortcuts” in the search space.**
Given a combinatorial problem C

Backdoors (for satisfiable instances) (wrt subsolver A):

**Definition** [backdoor] A nonempty subset $S$ of the variables is a backdoor in $C$ for $A$ if for some $a_S : S \rightarrow D$, $A$ returns a satisfying assignment of $C[a_S]$.

**Strong backdoors (apply to satisfiable or inconsistent instances):**

**Definition** [strong backdoor] A nonempty subset $S$ of the variables is a strong backdoor in $C$ for $A$ if for all $a_S : S \rightarrow D$, $A$ returns a satisfying assignment or concludes unsatisfiability of $C[a_S]$. 
Reminder: Cycle-cutset

• Given an undirected graph, a cycle cutset is a subset of nodes in the graph whose removal results in a graph without cycles

• Once the cycle-cutset variables are instantiated, the remaining problem is a tree → solvable in polynomial time using arc consistency;

• A constraint graph whose graph has a cycle-cutset of size $c$ can be solved in time of $O((n-c) \cdot k^{c+2})$

• Important: verifying that a set of nodes is a cutset can be done in polynomial time (in number of nodes).

(Dechter 93)
Backdoors vs. Cutsets

• Can be viewed as a generalization of cutsets;

• Backdoors use a general notion of tractability based on a polytime sub-solver --- backdoors do not require a syntactic characterization of tractability.

• Backdoors factor in the semantics of the constraints wrt sub-solver and values of the variables;

• Backdoors apply to different representations, including different semantics for graphs, e.g., network flows --- CSP, SAT, MIP, etc;

Note: Cutsets and W-cutsets – tractability based solely on the structure of the constraint graph, independently of the semantics of the constraints;
Backdoors can be surprisingly small

<table>
<thead>
<tr>
<th>instance</th>
<th># vars</th>
<th># clauses</th>
<th>backdoor</th>
<th>fract.</th>
</tr>
</thead>
<tbody>
<tr>
<td>logistics.d</td>
<td>6783</td>
<td>437431</td>
<td>12</td>
<td>0.0018</td>
</tr>
<tr>
<td>3bitadd_32</td>
<td>8704</td>
<td>32316</td>
<td>53</td>
<td>0.0061</td>
</tr>
<tr>
<td>pipe_01</td>
<td>7736</td>
<td>26087</td>
<td>23</td>
<td>0.0030</td>
</tr>
<tr>
<td>qg_30_1</td>
<td>1235</td>
<td>8523</td>
<td>14</td>
<td>0.0113</td>
</tr>
<tr>
<td>qg_35_1</td>
<td>1597</td>
<td>10658</td>
<td>15</td>
<td>0.0094</td>
</tr>
</tbody>
</table>

Most recent: Other combinatorial domains. E.g. graphplan planning, near constant size backdoors (2 or 3 variables) and log(n) size in certain domains. (Hoffmann, Gomes, Selman ’04)

Backdoors capture critical problem resources (bottlenecks).
Backdoors --- “seeing is believing”

Constraint graph of reasoning problem. One node per variable: edge between two variables if they share a constraint.

Logistics_b.cnf planning formula. 843 vars, 7,301 clauses, approx min backdoor 16 (backdoor set = reasoning shortcut)
Logistics.b.cnf after setting 5 backdoor vars.
After setting just 12 (out of 800+) backdoor vars – problem almost solved.
MAP-6-7.cnf infeasible planning instances. Strong backdoor of size 3. 392 vars, 2,578 clauses.
After setting 2 (out of 392) backdoor vars. --- reducing problem complexity in just a few steps!
Inductive inference problem --- ii16a1.cnf. 1650 vars, 19,368 clauses. Backdoor size 40.
After setting 6 backdoor vars.
Some other intermediate stages:

After setting 38 (out of 1600+) backdoor vars:
Some other intermediate stages:

After setting 38 (out of 1600+) backdoor vars:

So: Real-world structure hidden in the network. Can be exploited by automated reasoning engines.
### Size

**backdoor**

<table>
<thead>
<tr>
<th>$B(n)$</th>
<th>deterministic</th>
<th>randomized</th>
<th>heuristic</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n/k$</td>
<td>small $\exp(n)$</td>
<td>smaller $\exp(n)$</td>
<td>tiny $\exp(n)$</td>
</tr>
<tr>
<td>$O(\log n)$</td>
<td>$\left( \frac{n}{\sqrt{\log n}} \right)^{O(\log n)}$</td>
<td>$\left( \frac{n}{\log n} \right)^{O(\log n)}$</td>
<td>$\text{poly}(n)$</td>
</tr>
<tr>
<td>$O(1)$</td>
<td>$\text{poly}(n)$</td>
<td>$\text{poly}(n)$</td>
<td>$\text{poly}(n)$</td>
</tr>
</tbody>
</table>

$n = \text{num. vars.}$  
$k$ is a constant

(Williams, Gomes, and Selman ’03)
Other Techniques: Nogood Learning

• Learn from mistakes *during* search
  – Nogood Learning: when DPLL backtracks,

• Learn a concise reason: what went wrong
  – avoid similar ‘mistakes’ in the future!
  – Extremely powerful in practice
Other Techniques: Machine Learning

• Machine learning to build algorithm portfolios
  – Observation: no single SAT solver is good on every family of instances
  – Features of a given instance can be used to predict, with reasonable accuracy, which solver will work well on it!
  – Solution: design a portfolio solver using ML techniques
    • Based on runtime prediction models
    • Recent work – avoid complex models, use k-NN or clustering

• Automatic parameter tuning (generic and instance-specific)
  – SAT solvers are designed with many ‘hardwired’ parameters
  – Millions of parameter combinations – impossible to explore all by hand!
  – Solution: use automatic parameter tuning tools based on local search, genetic algorithms, etc.
Where is SAT Research headed?

Direction A: getting more out of SAT solvers
- Minimal/minimum *unsatisfiable cores*: very useful in practice!
- MAXSAT, weighted MAXSAT
- Circuit representations (rather than CNF)

Direction B: tacking problems *harder* than SAT
- Near-uniform *sampling* from the solution space
- *Solution counting* (with relations to probabilistic inference)
  -- #P-hard: challenging even to approximate with good confidence bounds

Direction C: expanding the applicability of SAT technology
- *Pseudo-Boolean* SAT (i.e., linear inequalities over Boolean vars)
- *SMT*: Satisfiability Modulo Theories (e.g., linear arithmetic, bit-vector operations, uninterpreted functions)