

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

Tonne of Applications

MY HOBBY:
EMBEDDING NP-COMPLETE PROBLEMS IN RESTAURANT ORDERS

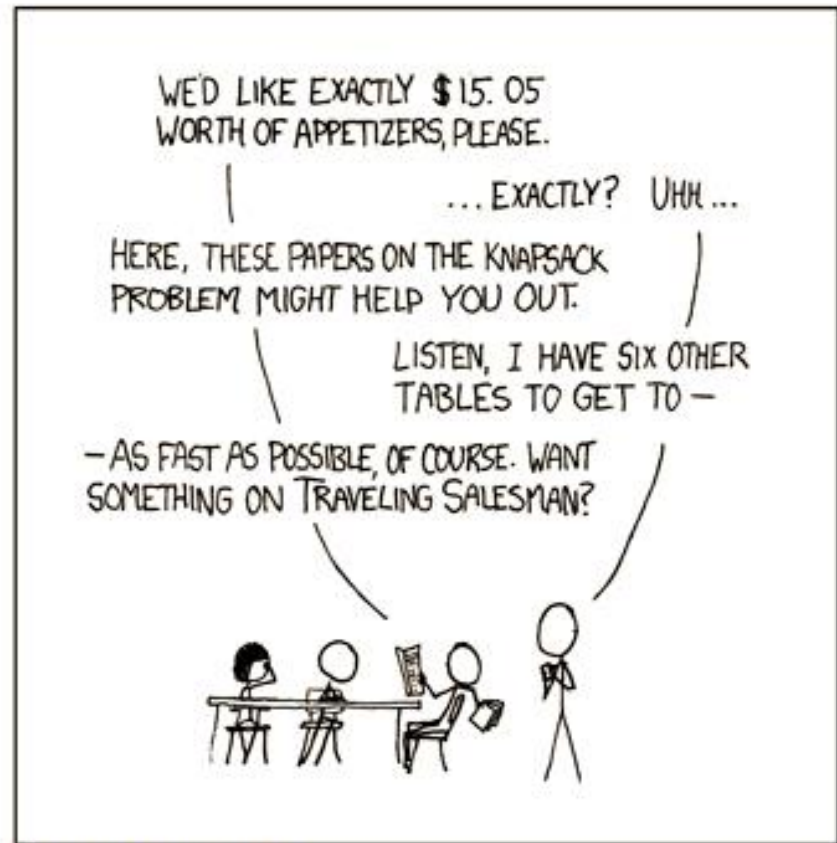
CHOTCHKIES RESTAURANT

~ APPETIZERS ~

MIXED FRUIT	2.15
FRENCH FRIES	2.75
SIDE SALAD	3.35
HOT WINGS	3.55
MOZZARELLA STICKS	4.20
SAMPLER PLATE	5.80

~ SANDWICHES ~

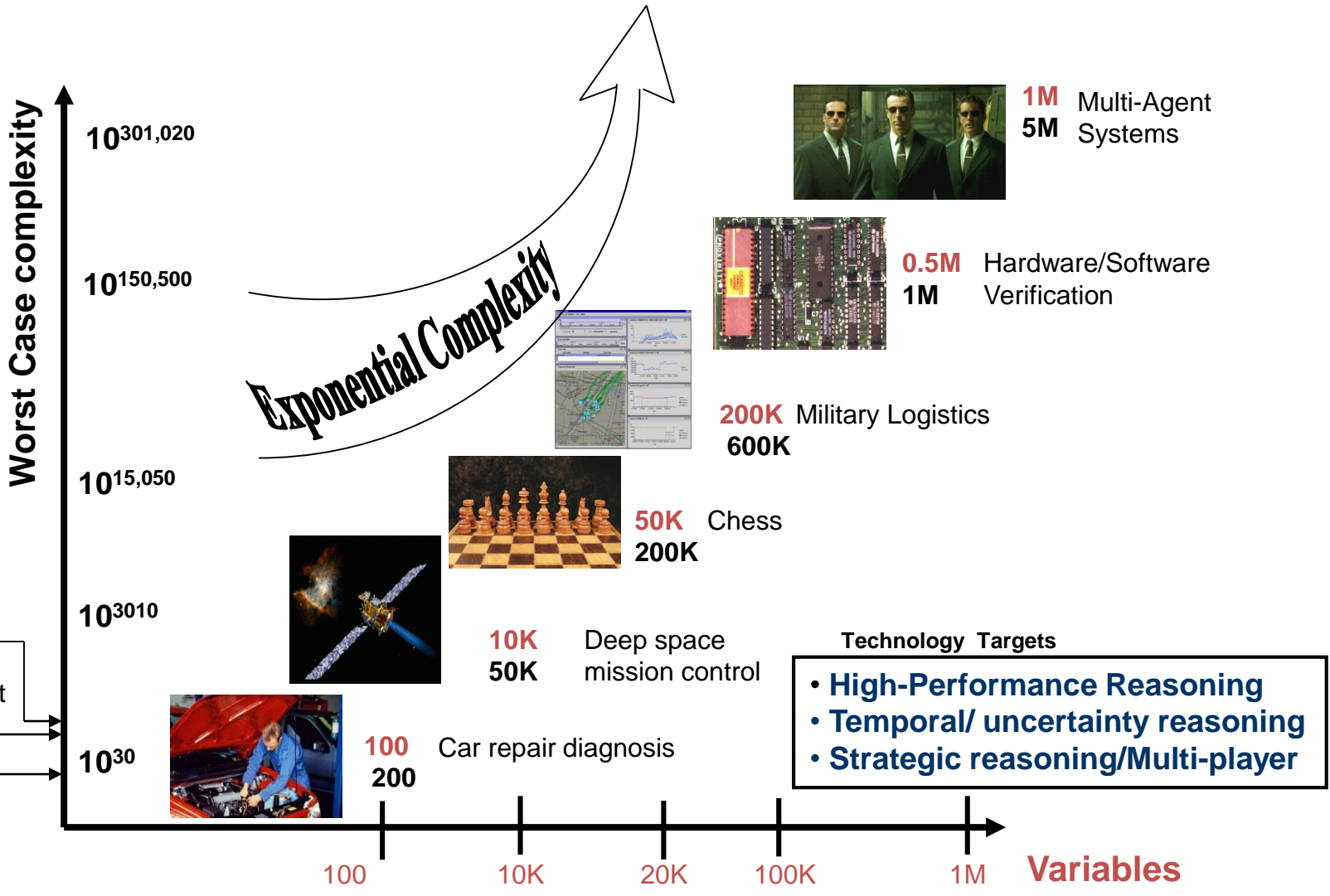
BARBECUE	6.55
----------	------



Title text: General solutions get you a 50% tip.

Real-World Reasoning

Tackling inherent computational complexity



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

Rules (Constraints)⁴

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:*
 $(\text{GasInTank} \wedge \text{FuelLineOK}) \rightarrow \text{GasInEngine}$
 $(\text{GasInEngine} \wedge \text{PowerToPlugs}) \rightarrow \text{EngineRuns}$
 $(\text{BatteryOK} \wedge \text{CablesOK}) \rightarrow \text{PowerToPlugs}$
 $(\text{BatteryOK} \wedge \text{ClockOK}) \rightarrow \text{ClockRuns}$
- *Observations:*
 $\neg \text{EngineRuns}, \text{GasInTank}, \text{ClockRuns}$

Example

- Is $F \cup \text{Observations} \cup \text{Assumptions}$ consistent?
- $F \cup \{\neg\text{EngineRuns}, \text{GasInTank}, \text{ClockRuns}\}$
 $\cup \{\text{FuelLineOK}, \text{BatteryOK}, \text{CablesOK}, \text{ClockOK}\} \rightarrow \text{false}$
 - *Must restore consistency!*
- $F \cup \{\neg\text{EngineRuns}, \text{GasInTank}, \text{ClockRuns}\}$
 $\cup \{\text{BatteryOK}, \text{CablesOK}, \text{ClockOK}\} \rightarrow \text{false}$
 - $\neg \text{FuelLineOK}$ is a diagnosis
- $F \cup \{\neg\text{EngineRuns}, \text{GasInTank}, \text{ClockRuns}\}$
 $\cup \{\text{FuelLineOK}, \text{CablesOK}, \text{ClockOK}\} \rightarrow \text{false}$
 - $\neg \text{BatteryOK}$ is not a diagnosis

Complexity of Diagnosis

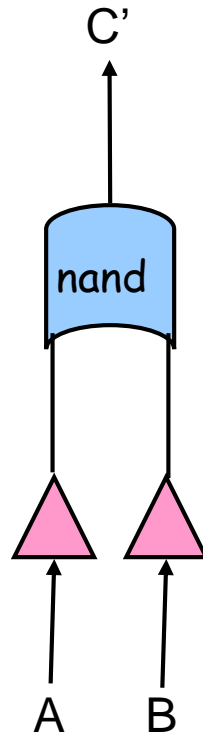
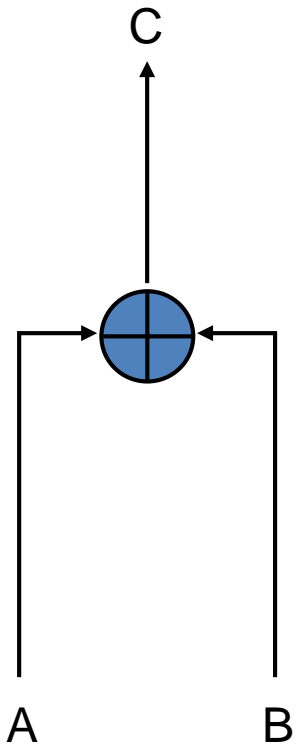
- If F is **Horn**, then each consistency test takes linear time – unit propagation is complete for Horn clauses.
- Complexity = ways to delete propositions from Assumption Set that are considered.
 - Single fault diagnosis – $O(n^2)$
 - Double fault diagnosis – $O(n^3)$
 - Triple fault diagnosis – $O(n^4)$

...

Deep Space One

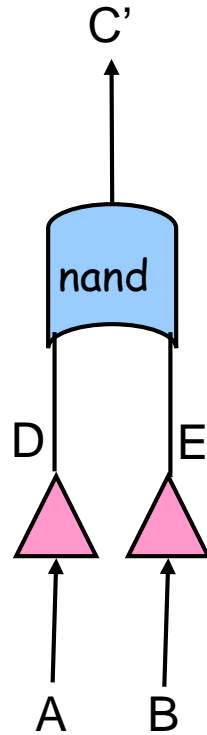
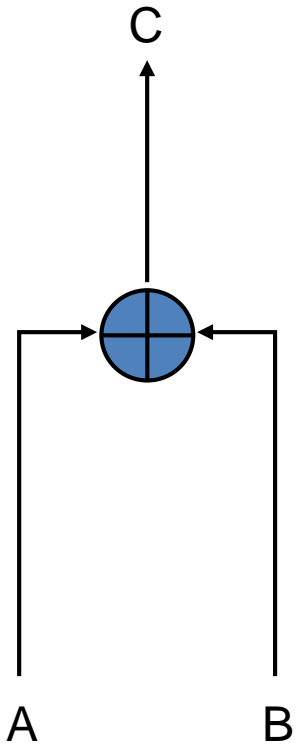
- a failed electronics unit
 - Remote Agent fixed by reactivating the unit.
- a failed sensor providing false information
 - Remote Agent recognized as unreliable and therefore correctly ignored.
- an attitude control thruster (a small engine for controlling the spacecraft's orientation) stuck in the "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



$$C \equiv (A \vee B)$$

$$C' \equiv \neg(D \wedge E)$$

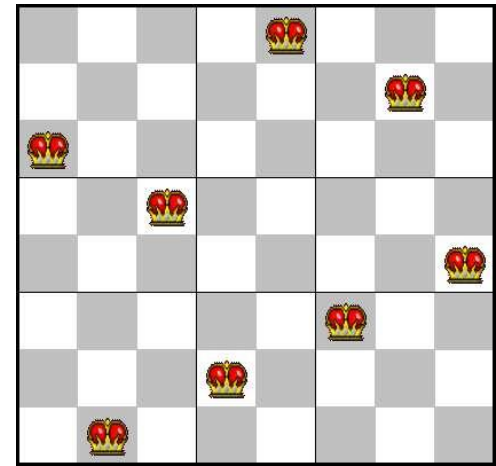
$$D \equiv \neg A$$

$$E \equiv \neg B$$

$$\neg(C \equiv C')$$

SAT Translation of N-Queens

- At least one queen each column:
($Q_{11} \vee Q_{12} \vee Q_{13} \vee \dots \vee Q_{18}$)
($Q_{21} \vee Q_{22} \vee Q_{23} \vee \dots \vee Q_{28}$)
...
- No attacks:
($\sim Q_{11} \vee \sim Q_{12}$)
($\sim Q_{11} \vee \sim Q_{22}$)
($\sim Q_{11} \vee \sim Q_{21}$)
...



Graph Coloring

- A new SAT Variable for var-val pair

$$X_{WA-r}, X_{WA-g}, X_{WA-b}, X_{NT-r} \dots$$

- Each var has at least 1 value

$$- X_{WA-r} \vee X_{WA-g} \vee X_{WA-b}$$

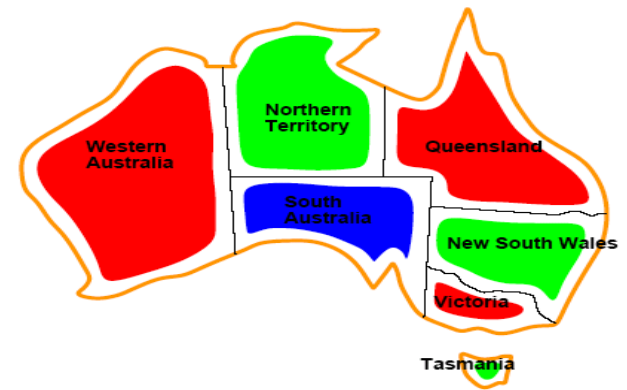
- No var has two values

$$- \sim X_{WA-r} \vee \sim X_{WA-g}$$

$$- \sim X_{WA-r} \vee \sim X_{WA-b}$$

- Constraints

$$- \sim X_{WA-r} \vee \sim X_{NT-r}$$



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 = vector of Booleans representing
state of machine at time t

$\rho : State \times Input \rightarrow State$

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

$$\left(\bigwedge_{i=0}^{k-1} \left(\overline{S}_{i+1} \equiv \rho(\overline{S}_i, \overline{I}_i) \right) \right) \wedge S_o \wedge \neg \gamma(S_k)$$

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 cnf 51639 368352`

`-1 7 0`

`-1 6 0`

`-1 5 0`

`-1 -4 0`

`-1 3 0`

`-1 2 0`

`-1 -8 0`

`-9 15 0`

`-9 14 0`

`-9 13 0`

`-9 -12 0`

`-9 11 0`

`-9 10 0`

`-9 -16 0`

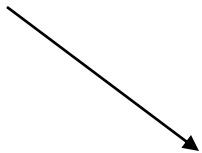
`-17 23 0`

`-17 22 0`

*i.e. ((not x_1) or x_7)
and ((not x_1) or x_6)
and ... etc.*

10 pages later:

185 -9 0
185 -1 0
177 169 161 153 145 137 129 121 113 105 97
89 81 73 65 57 49 41
33 25 17 9 1 -185 0
186 -187 0
186 -188 0
...



(x_{177} or x_{169} or x_{161} or x_{153} ...
or x_{17} or x_9 or x_1 or (not x_{185}))

clauses / constraints are getting more interesting...

4000 pages later:

```
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 10086 10087 10088 10089 10090
 10098 10099 10100 10101 10102 10103 10104
 10105 10106 10107 10108 -55 -54 53 -52 -51 50
 10047 10048 10049 10050 10051 10235 -10236 0
10237 -10008 0
10237 -10009 0
10237 -10010 0
```

!!!

***a 59-cnf
clause...***

...

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: $2^{50000} \approx 3.160699437 \cdot 10^{15051} \dots !!!$

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)
 - “Industrial-instances-only” **SAT Races** (2008, 2010)
 - A tremendous resource! E.g., SAT Competition 2014:
 - 137 solvers submitted, downloadable, mostly open source
 - 79 teams, 14 countries
 - 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*

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)

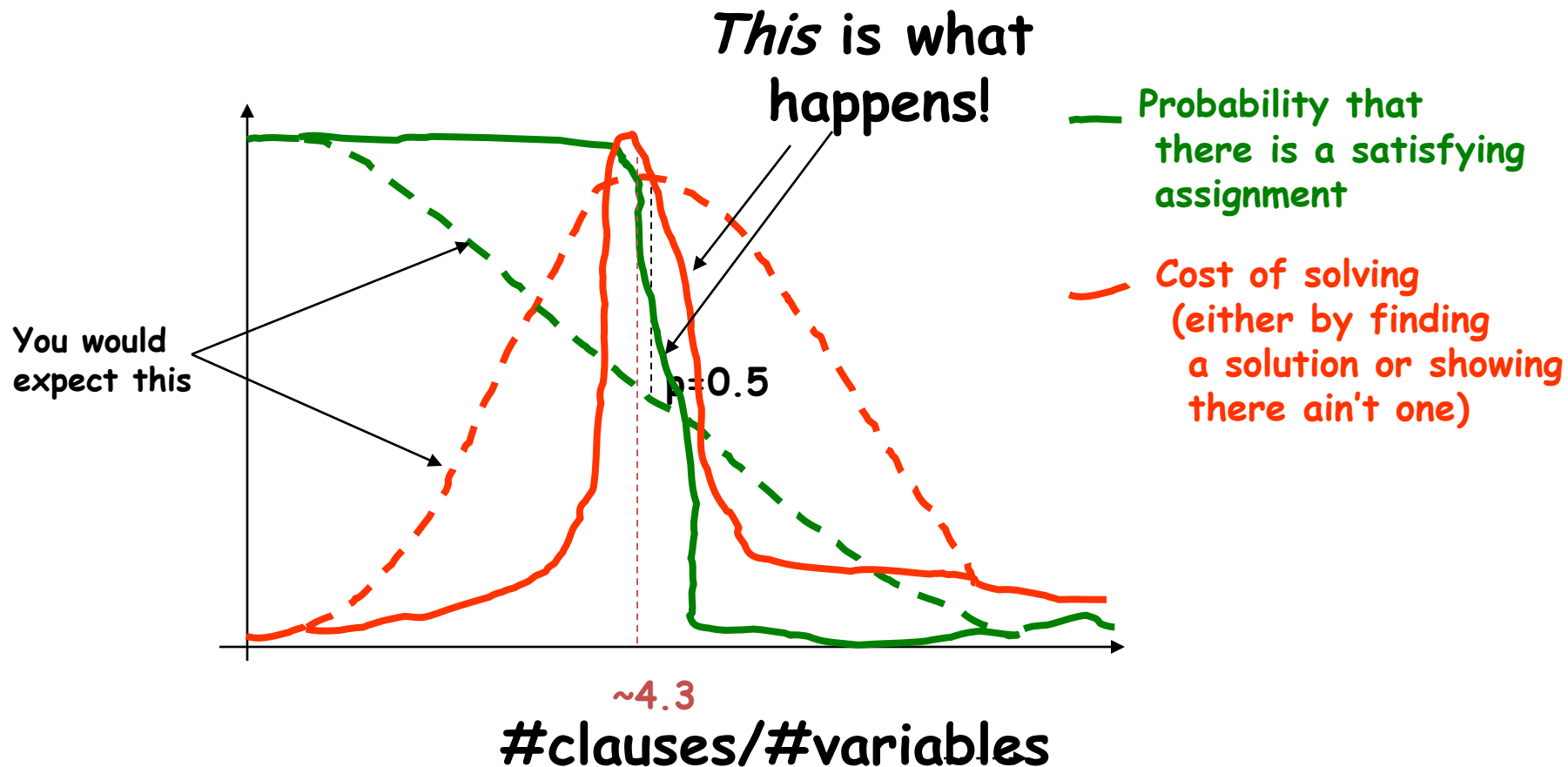
Direction B: tackling problems *harder* than SAT

- 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

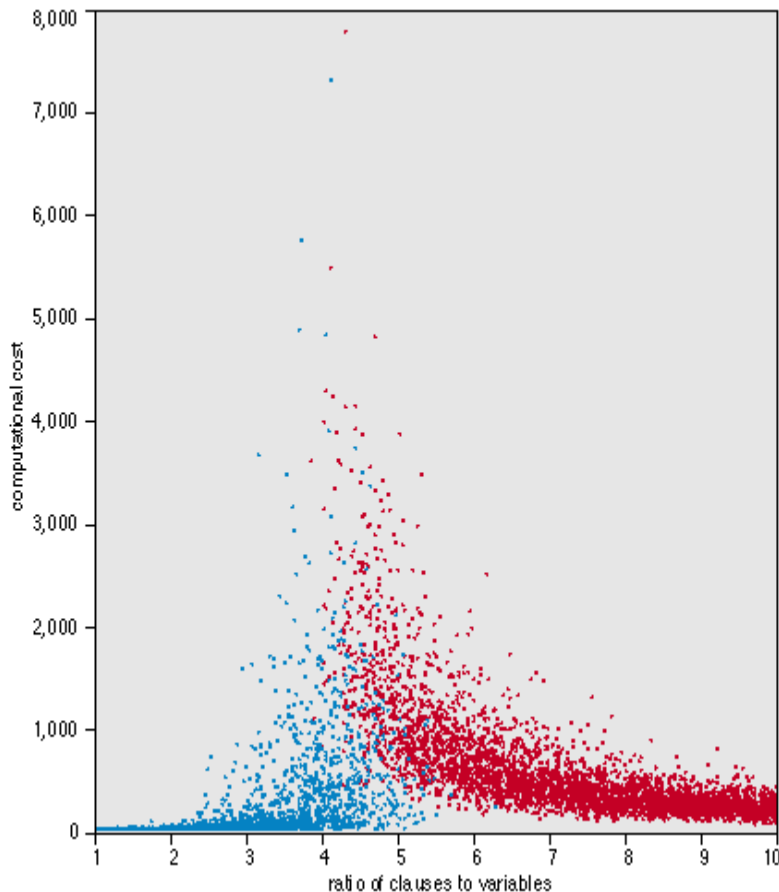
- **Pseudo-Boolean** SAT (i.e., linear inequalities over Boolean vars)
- **SMT**: Satisfiability Modulo Theories (e.g., linear arithmetic, bit-vector operations, uninterpreted functions)

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



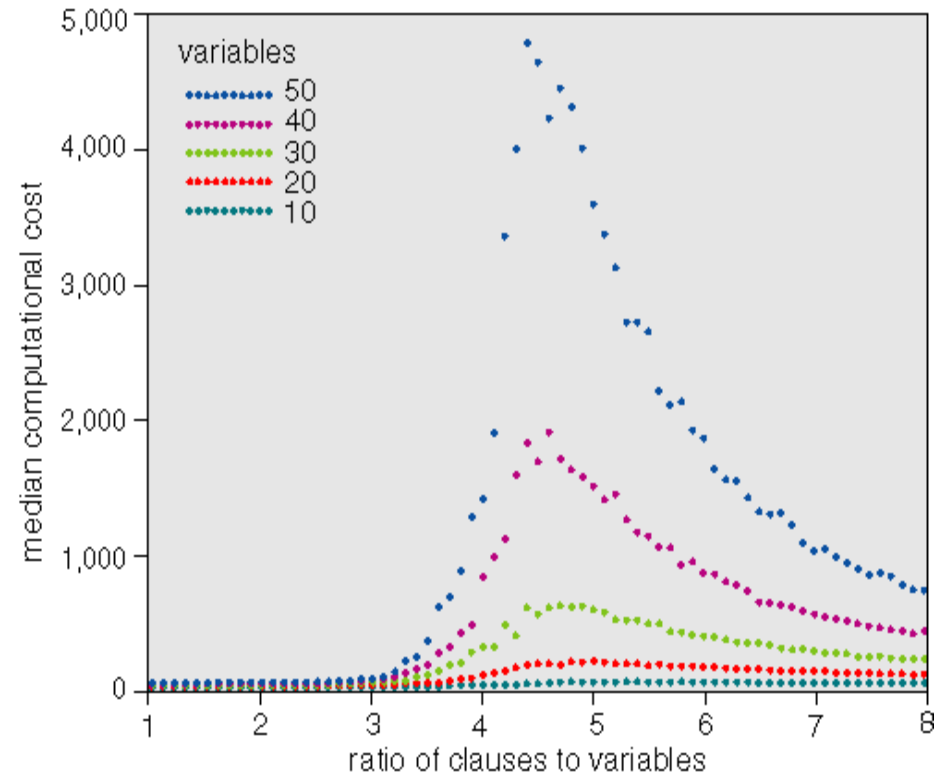
Random 3-SAT

- Random 3-SAT
 - sample uniformly from space of all possible 3-clauses
 - n variables, l clauses
- Which are the hard instances?
 - around $l/n = 4.3$



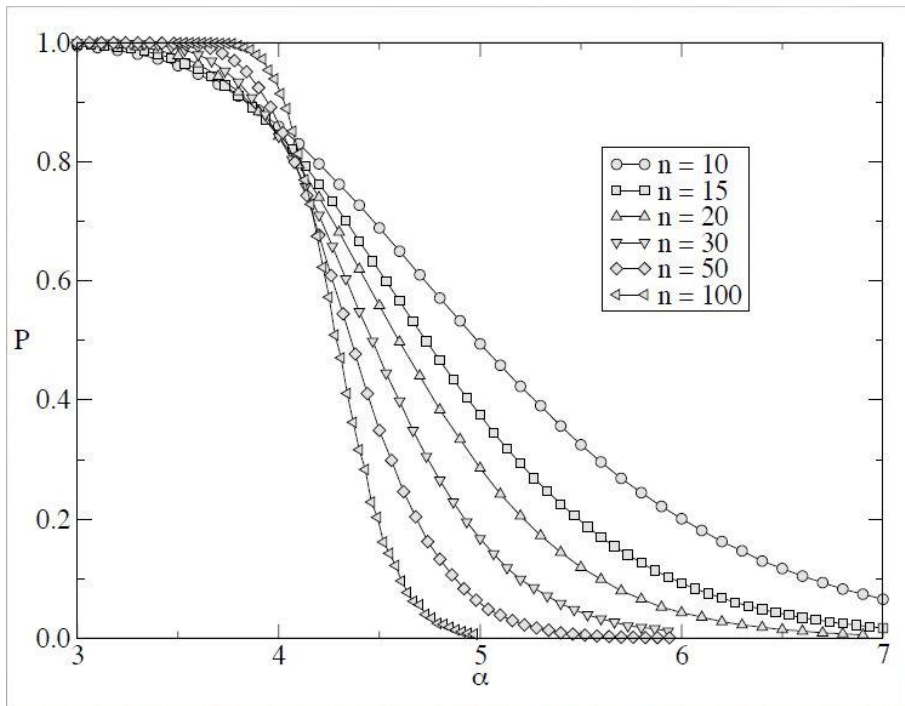
Random 3-SAT

- Varying problem size, n
- Complexity peak appears to be largely invariant of algorithm

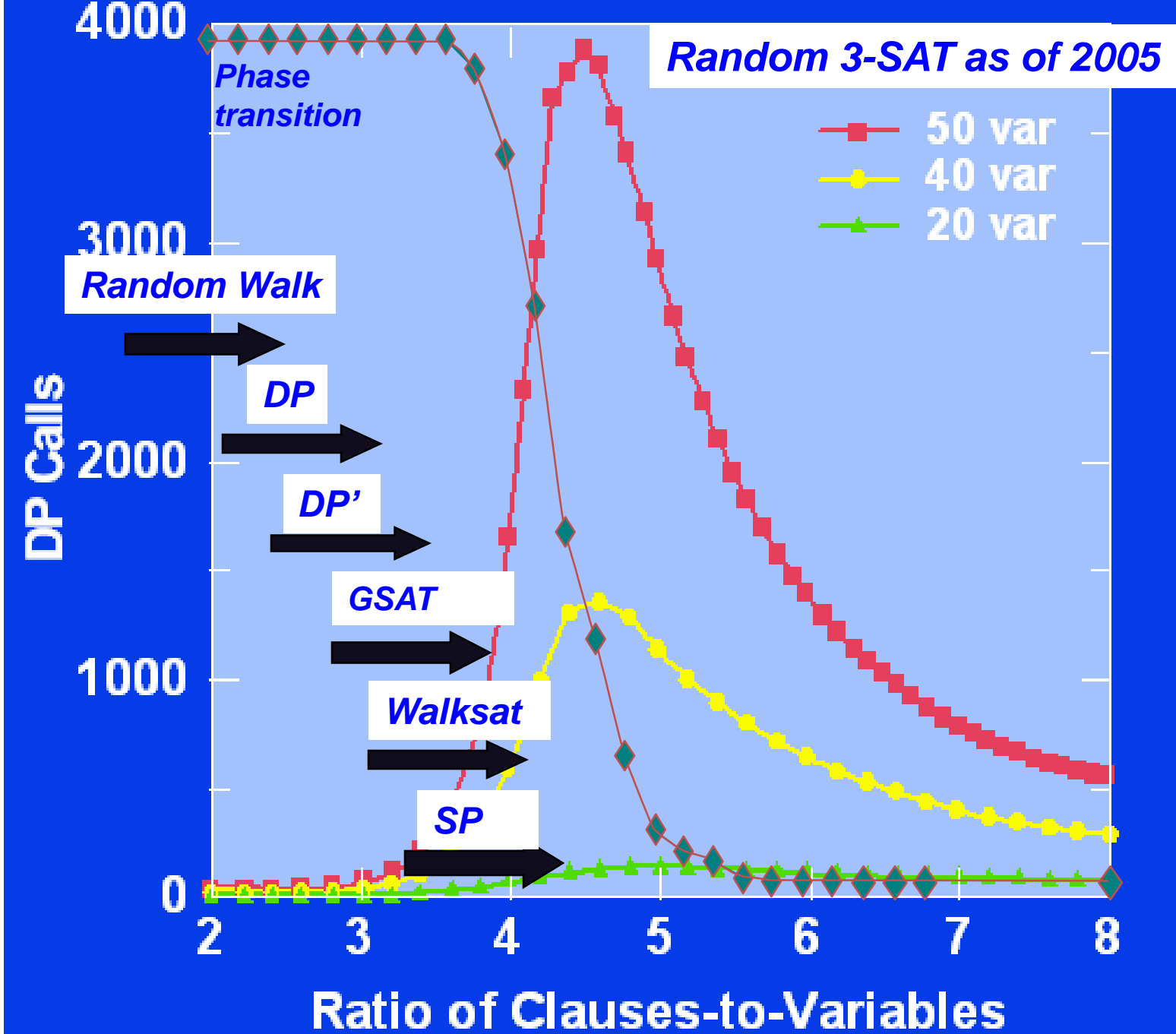


Random 3-SAT

- 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



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

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

3SAT phase transition

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

For any statistic X

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

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

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

$$\geq p(X \geq 1)$$

3SAT phase transition

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

For any statistic X

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

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

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

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

$$E[X] = 2^n * (7/8)^n$$

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

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

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

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

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

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

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

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

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

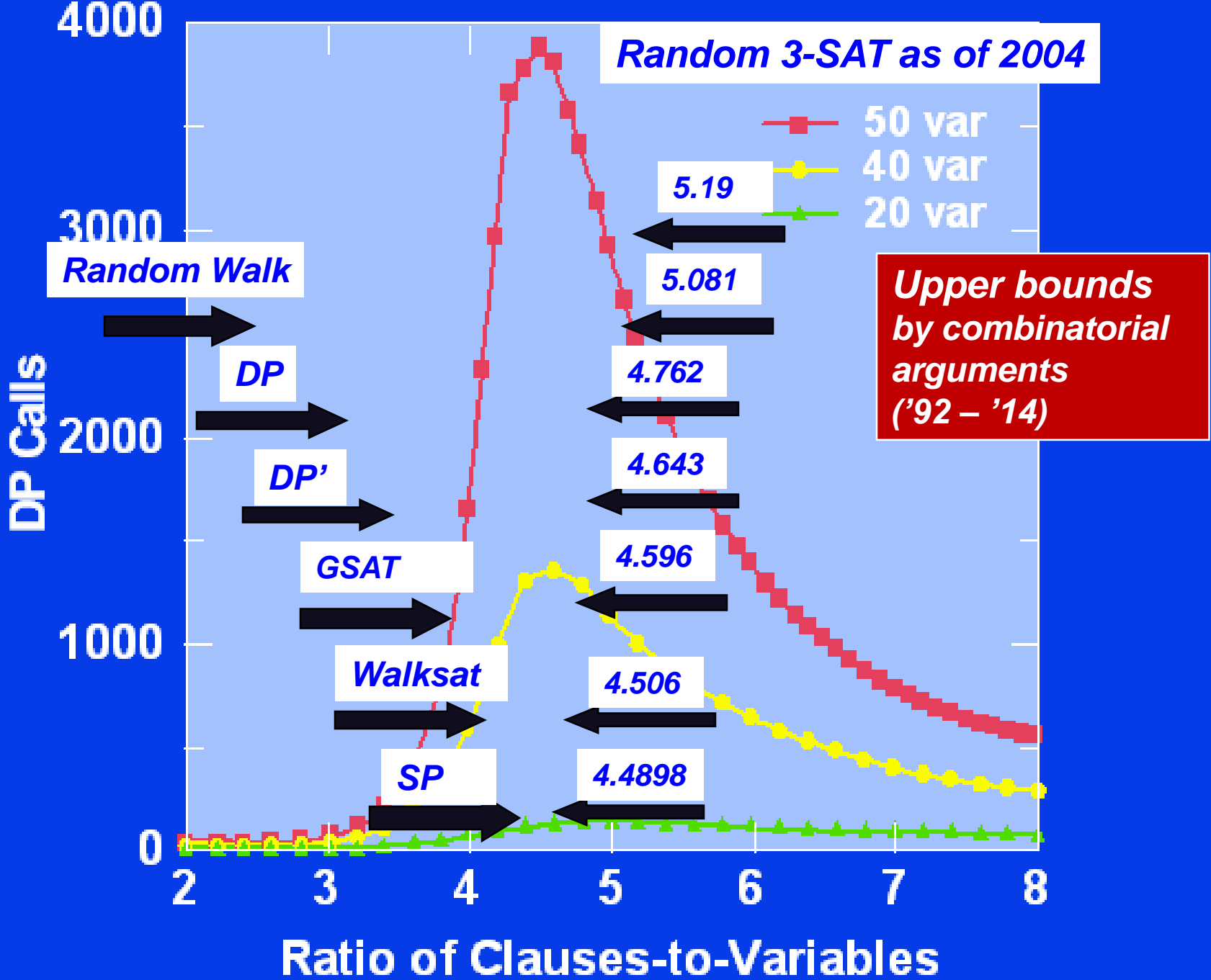
$$\text{If } E[X] < 1, \text{ then } 2^n * (7/8)^l < 1$$

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

$$l/n > 1/\log_2(8/7) = 5.19\dots$$

Average vs Number

- 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!



$2+p$ -SAT



Morph between 2-SAT and 3-SAT

- fraction p of 3-clauses
- fraction $(1-p)$ of 2-clauses

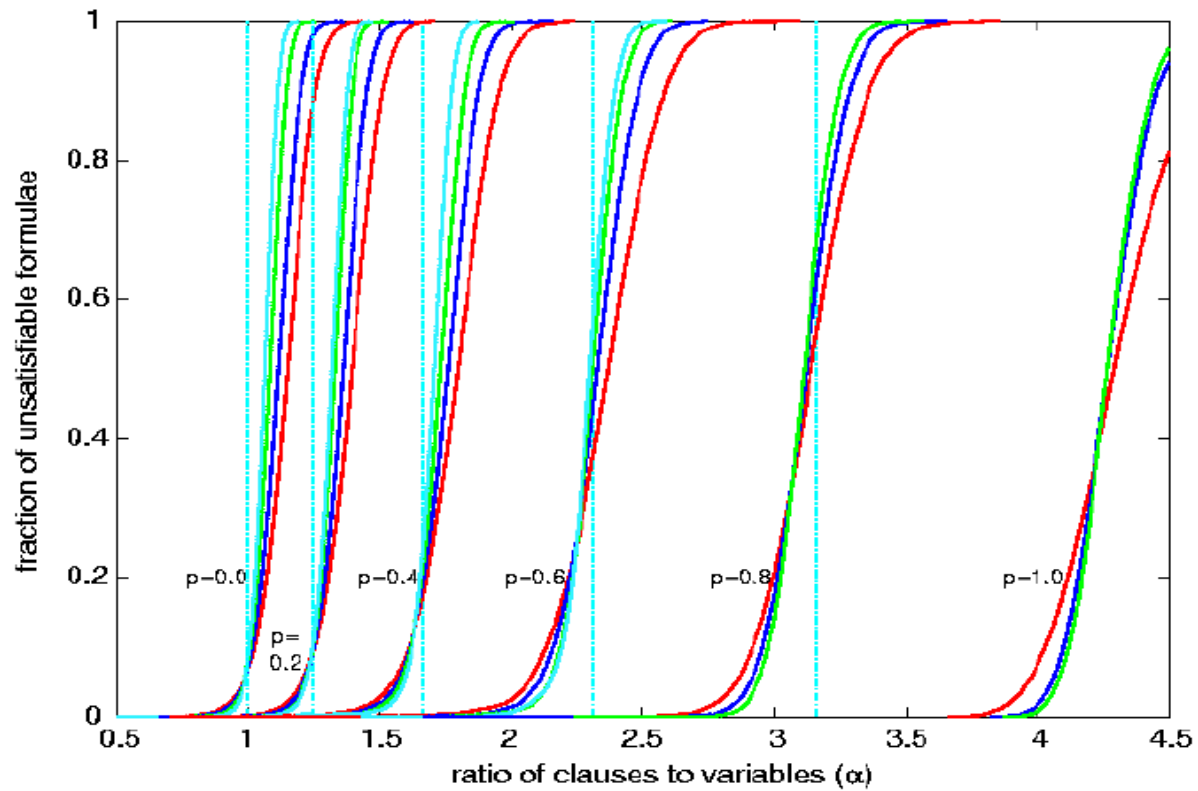
[Monasson et al 1999]

2+p-SAT



- Maps from P to NP
 - NP-complete for any $p > 0$
 - Insight into change from P to NP [*Monasson et al 1999*]

$2+p$ -SAT



Computational Cost: 2+p-SAT

Tractable substructure can dominate!

