Advanced Satisfiability

Mausam

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

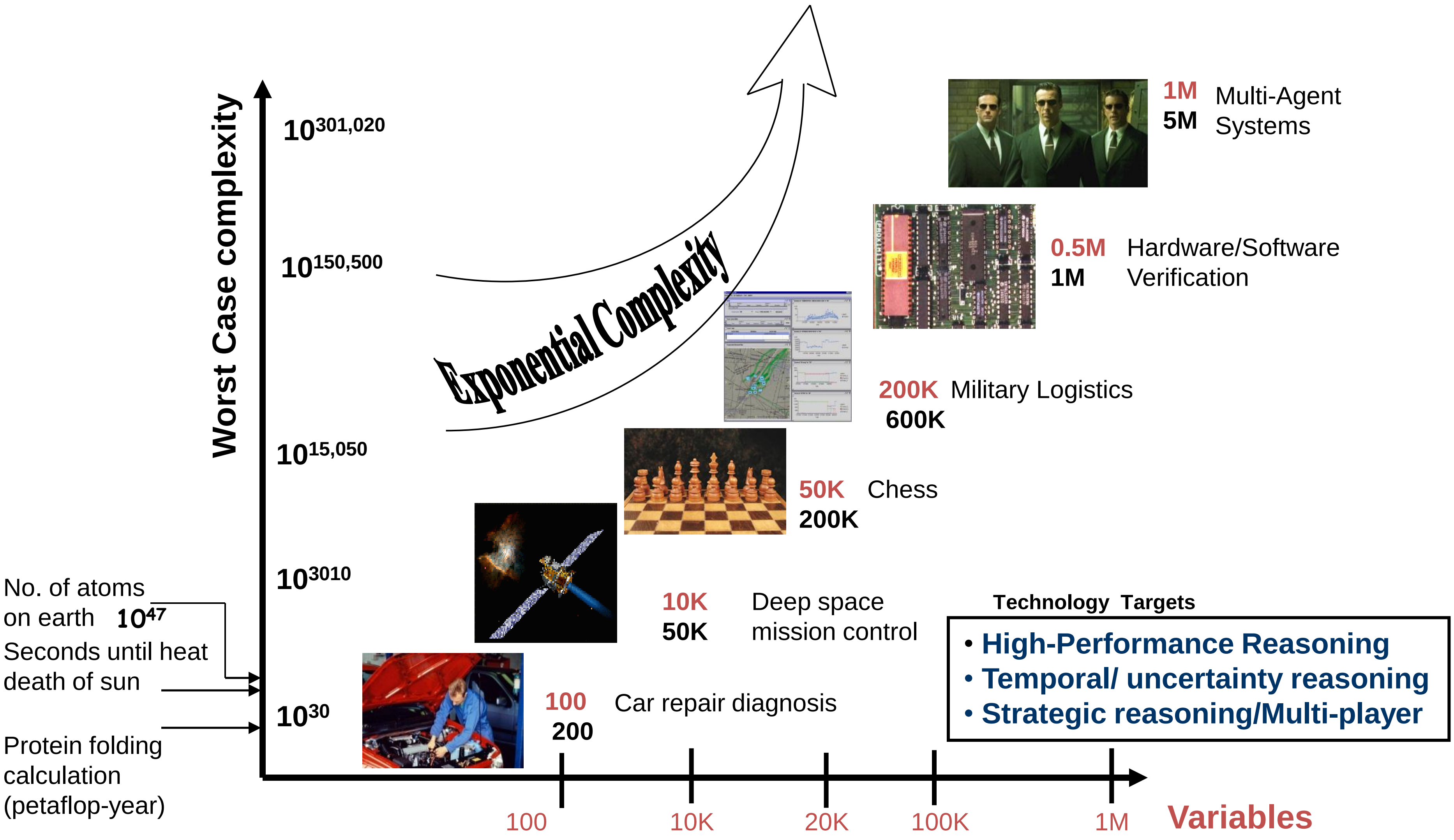
Why study Satisfiability?

- 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

DARPA Research Program



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

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 <u>certain parts</u> of the device – much is <u>hidden</u>

Model-Based, Consistency-Based Diagnosis

- Idea: create a logical formula that describes how the device <u>should</u> 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 <u>diagnosis</u> 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:*

  (GasInTank $\wedge$ FuelLineOK) $\rightarrow$ GasInEngine

  (GasInEngine $\wedge$ PowerToPlugs) $\rightarrow$ EngineRuns

  (BatteryOK $\wedge$ CablesOK) $\rightarrow$ PowerToPlugs

  (BatteryOK $\wedge$ ClockOK) $\rightarrow$ ClockRuns
- *Observations:*

  $\neg$ EngineRuns, GasInTank, ClockRuns

Example

- *Is* F ∪ Observations ∪ Assumptions *consistent?*

Example

- *Is* F ∪ Observations ∪ Assumptions *consistent?*

- F ∪ {¬EngineRuns, GasInTank, ClockRuns}
  ∪ { FuelLineOK, BatteryOK, CablesOK, ClockOK } → *false*
  - *Must restore consistency!*

Example

- *Is* F $\cup$ Observations $\cup$ Assumptions *consistent?*

- F $\cup$ {$\neg$EngineRuns, GasInTank, ClockRuns}
  $\cup$ { FuelLineOK, BatteryOK, CablesOK, ClockOK } → *false*
  - *Must restore consistency!*
- F $\cup$ {$\neg$EngineRuns, GasInTank, ClockRuns}
  $\cup$ { BatteryOK, CablesOK, ClockOK } → *false*
  - $\neg$ *FuelLineOK is a diagnosis*

Example

- *Is* F $\cup$ Observations $\cup$ Assumptions *consistent?*

- F $\cup$ {$\neg$EngineRuns, GasInTank, ClockRuns}
  $\cup$ { FuelLineOK, BatteryOK, CablesOK, ClockOK } → *false*
  - *Must restore consistency!*
- F $\cup$ {$\neg$EngineRuns, GasInTank, ClockRuns}
  $\cup$ { BatteryOK, CablesOK, ClockOK } → *false*
  - $\neg$ *FuelLineOK is a diagnosis*
- F $\cup$ {$\neg$EngineRuns, GasInTank, ClockRuns}
  $\cup$ {FuelLineOK, CablesOK, ClockOK } → *false*
  - $\neg$ *BatteryOK is <u>not</u> a diagnosis*

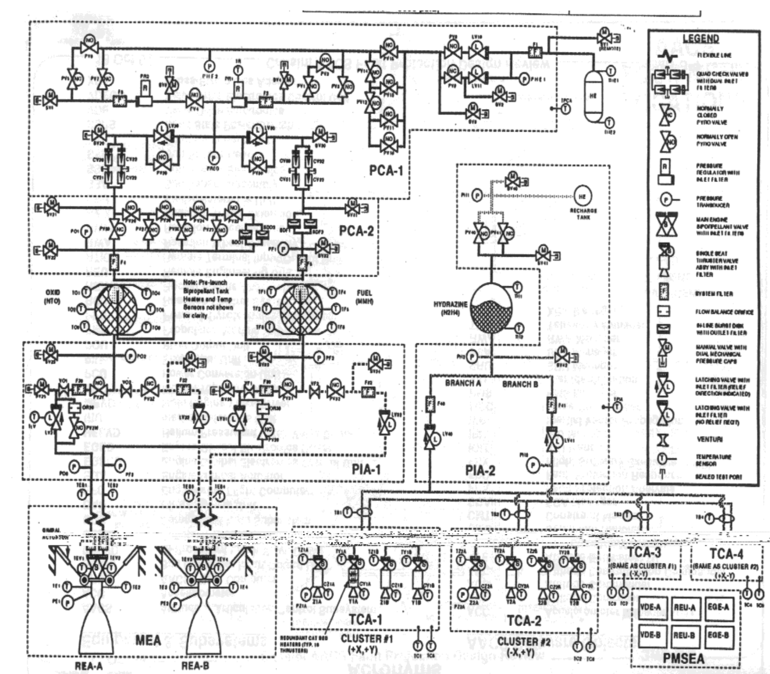
Complexity of Diagnosis

- 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

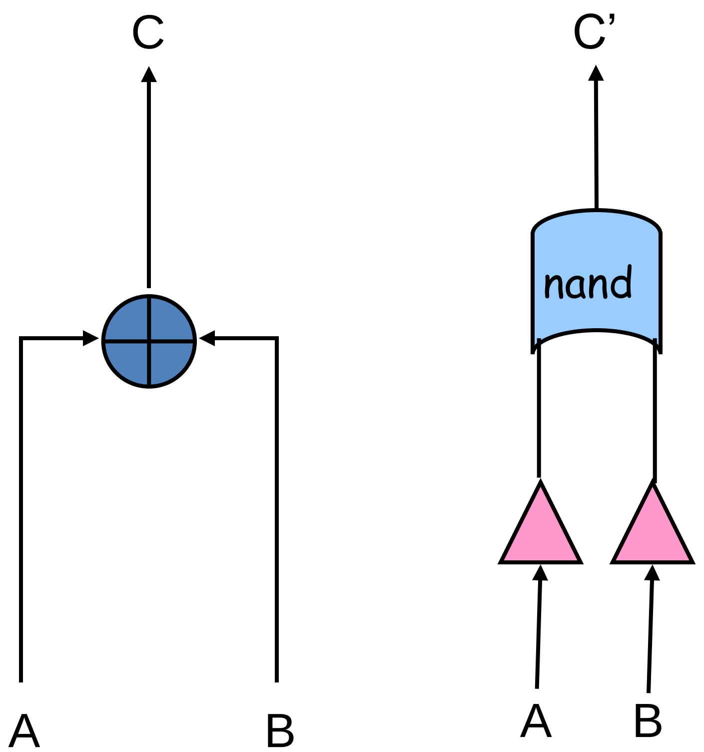
- Autonomous diagnosis & repair “Remote Agent”
- Compiled systems schematic to 7,000 var SAT problem



Deep Space One

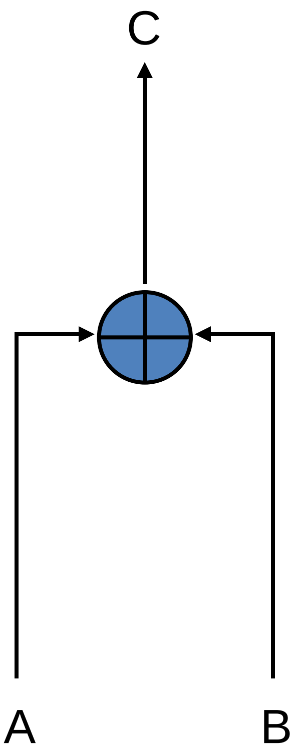
- a failed electronics unit
  - Remote Agent fixed by reactivating the unit.
- a failed sensor providing false information
  - Remote Agent recognized as unreliable and therefore correctly ignored.
- an altitude control thruster (a small engine for controlling the spacecraft's orientation) stuck in the "off" position
  - Remote Agent detected and compensated for by switching to a mode that did not rely on that thruster.

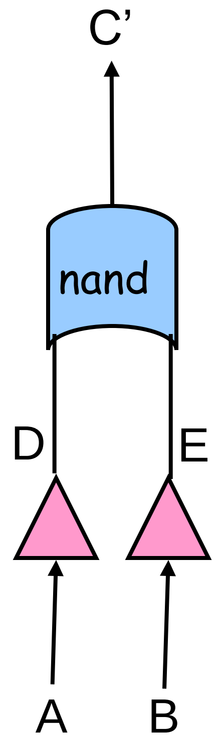
Testing Circuit Equivalence



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

Testing Circuit Equivalence





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

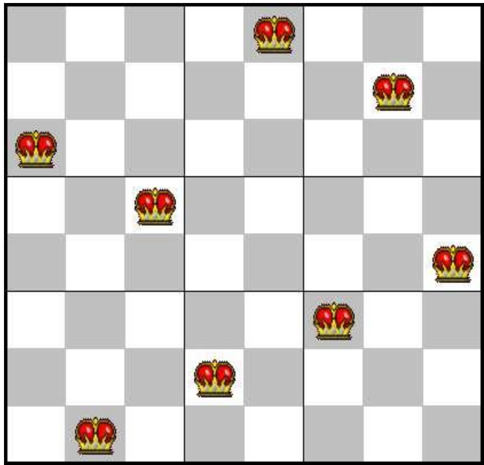
$$C' \equiv \neg(D \wedge E)$$

$$D \equiv \neg A$$

$$E \equiv \neg B$$

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

SAT Translation of N-Queens



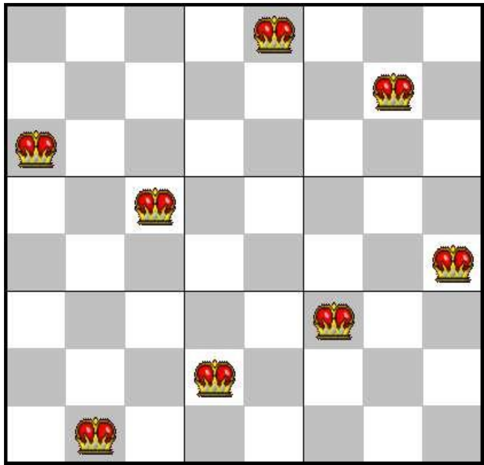
SAT Translation of N-Queens

- At least one queen each column:

  (Q11 v Q12 v Q13 v ... v Q18)

  (Q21 v Q22 v Q23 v ... v Q28)

  …



SAT Translation of N-Queens

- At least one queen each column:

  (Q11 v Q12 v Q13 v ... v Q18)

  (Q21 v Q22 v Q23 v ... v Q28)

  …

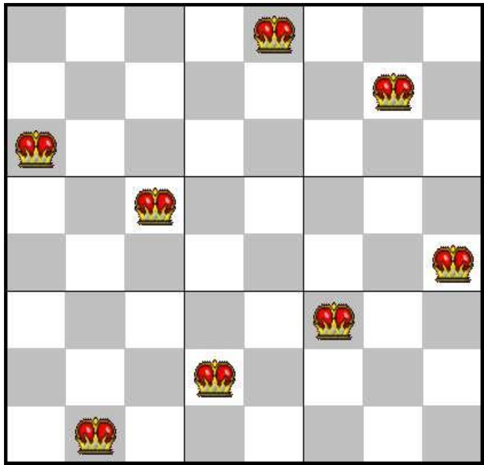
- No attacks:

  (~Q11 v ~Q12)

  (~Q11 v ~Q22)

  (~Q11 v ~Q21)

  …

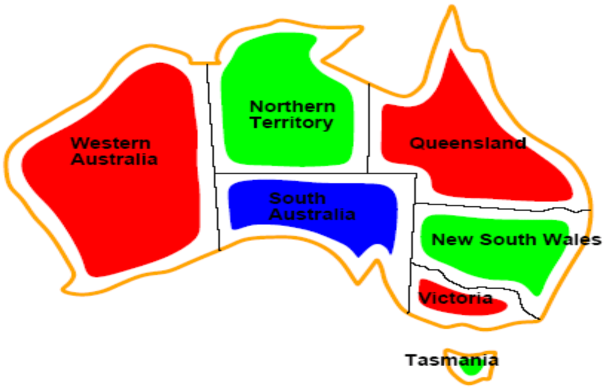


CSP → SAT

- A new SAT Variable for var-val pair

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

- Each var has at least 1 value
  - $X_{WA\text{-}r} \vee X_{WA\text{-}g} \vee X_{WA\text{-}b}$
- No var has two values
  - $\sim X_{WA\text{-}r} \vee \sim X_{WA\text{-}g}$
  - $\sim X_{WA\text{-}r} \vee \sim X_{WA\text{-}b}$
- Constraints
  - $\sim X_{WA\text{-}r} \vee \sim X_{NT\text{-}r}$



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

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

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

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

$$\left(\bigwedge_{i=0}^{k-1} \left(\overline{S_{i+1}} \equiv \rho(\overline{S_i}, \overline{I_i}) \right) \right) \wedge S_o \wedge \neg \gamma \left(S_k \right)$$

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:

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

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)
  - “Industrial-instances-only” **SAT Races** (2008, 2010)
  - 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

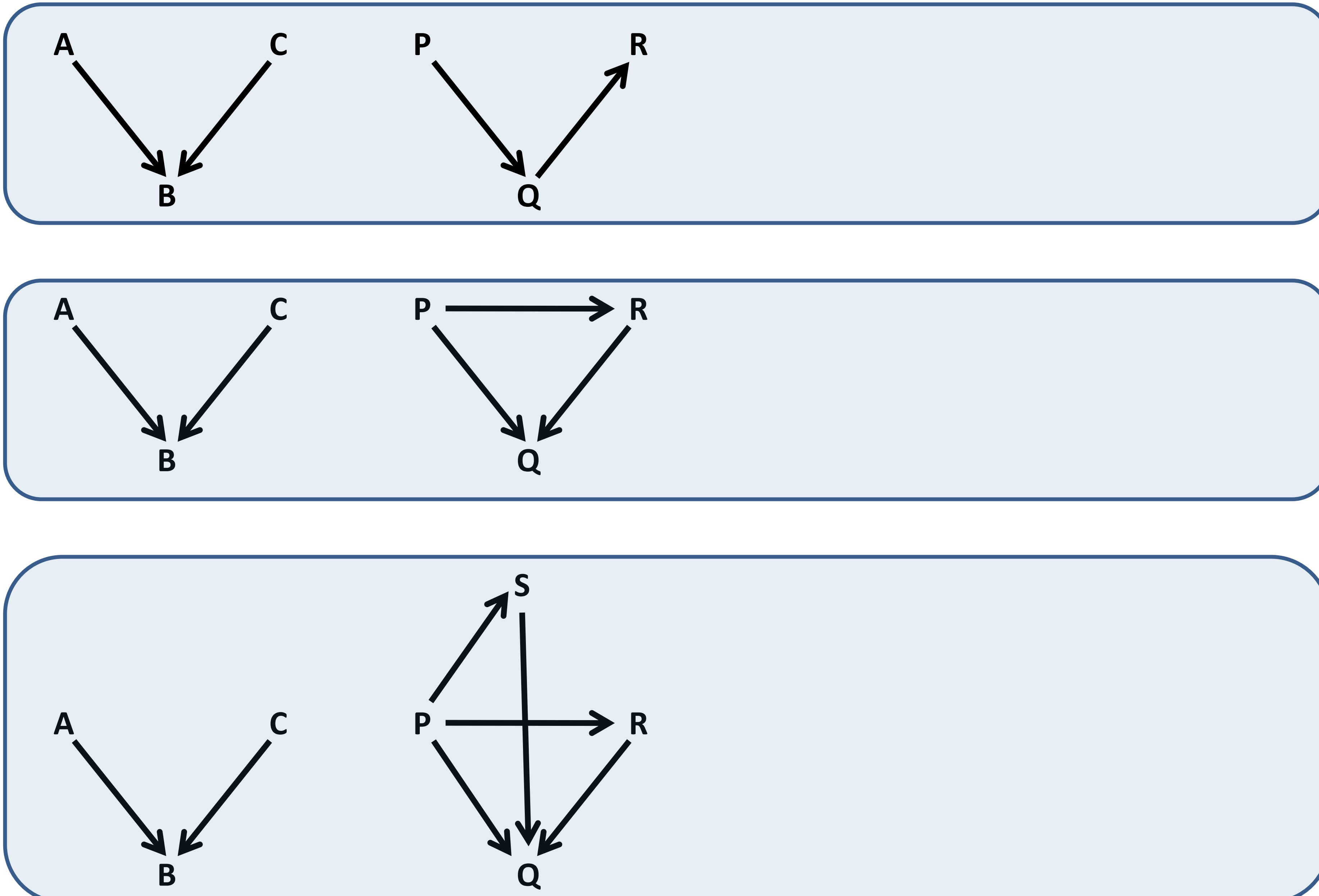
- *This constant improvement in SAT solvers is the key to the success of, e.g., SAT-based planning and verification*

Assignment 2: Graph Subset Mapping

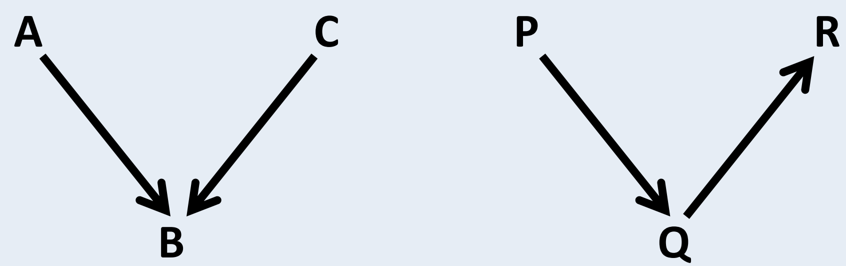
- Given two directed graphs G and G'
  - Check if G is a subset mapping to G'

- I.e. construct a one-one mapping (M) from all nodes of G to some nodes of G' s.t.
  - (n1,n2) in G → (M(n1), M(n2)) in G'
  - (n1,n2) not in G → (M(n1), M(n2)) not in G'

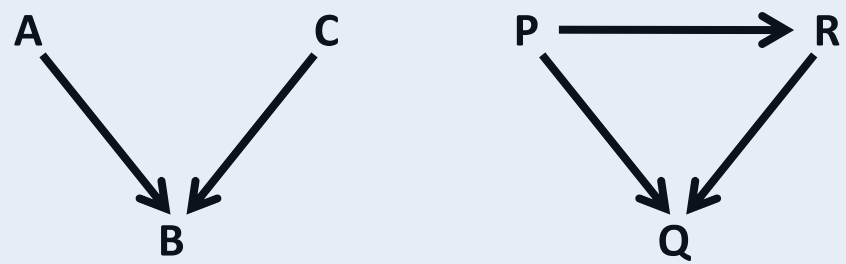
Graph G *Graph G’*

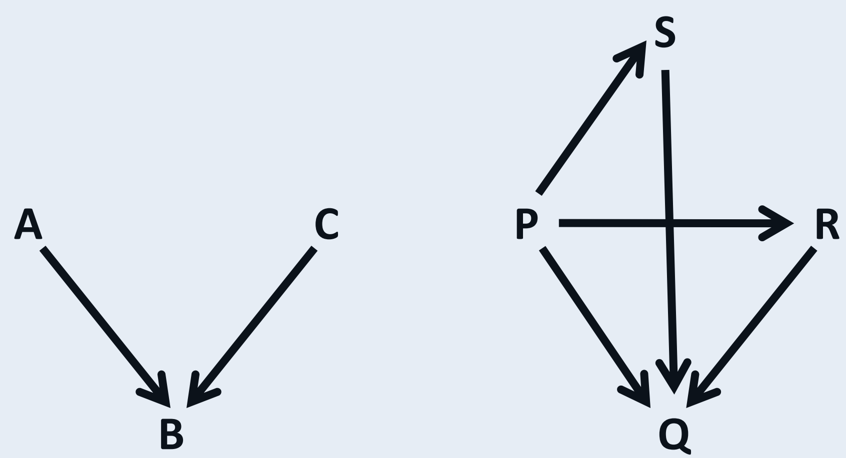


Graph G *Graph G'*

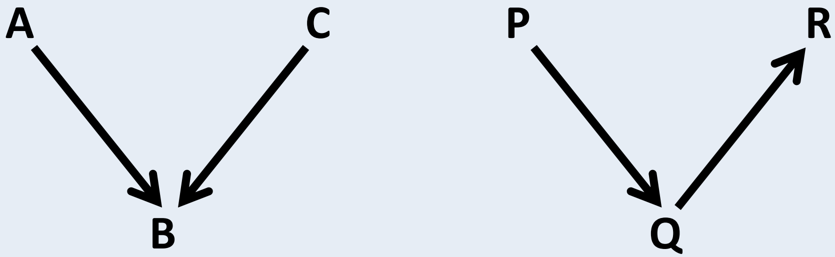


No, because the directionality of edges doesn't match.

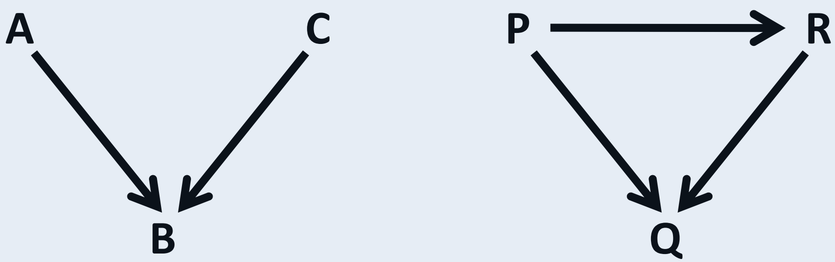




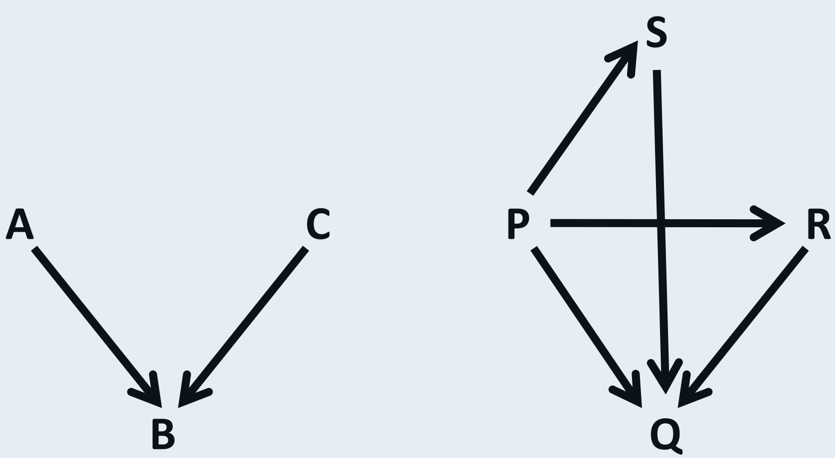
Graph G *Graph G'*



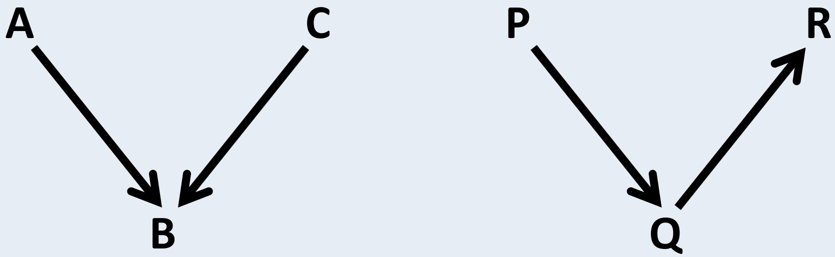
No, because the directionality of edges doesn't match.



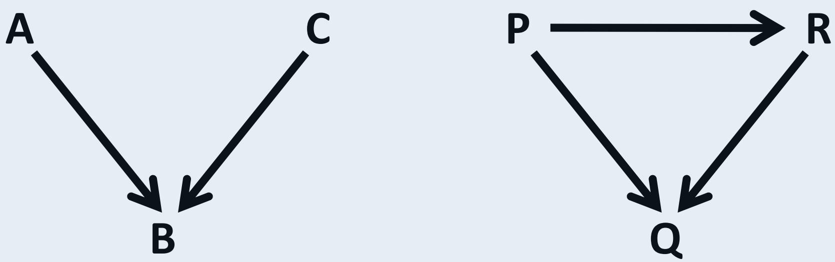
No, because there is no edge between A and C in *G* whereas there is one between P and R in *G'*.



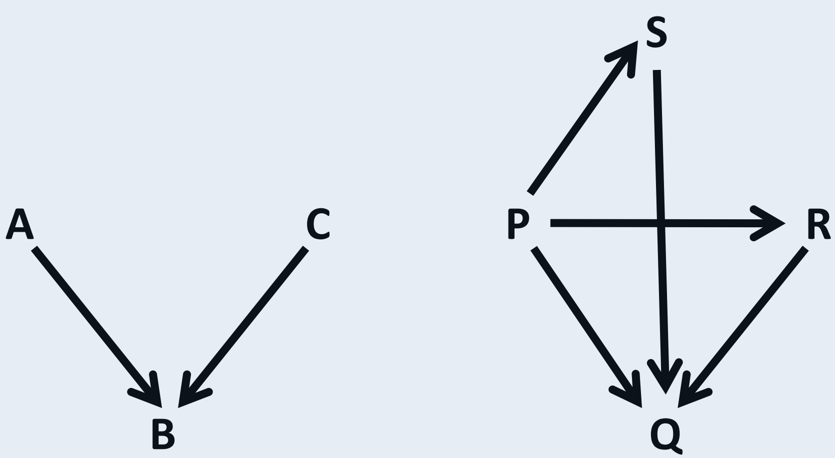
Graph G *Graph G'*



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 s1$\rightarrow$s2 by flipping x. Then:

  #unsat(s2) = #unsat(s1) – make(s1,x) + break(s1,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

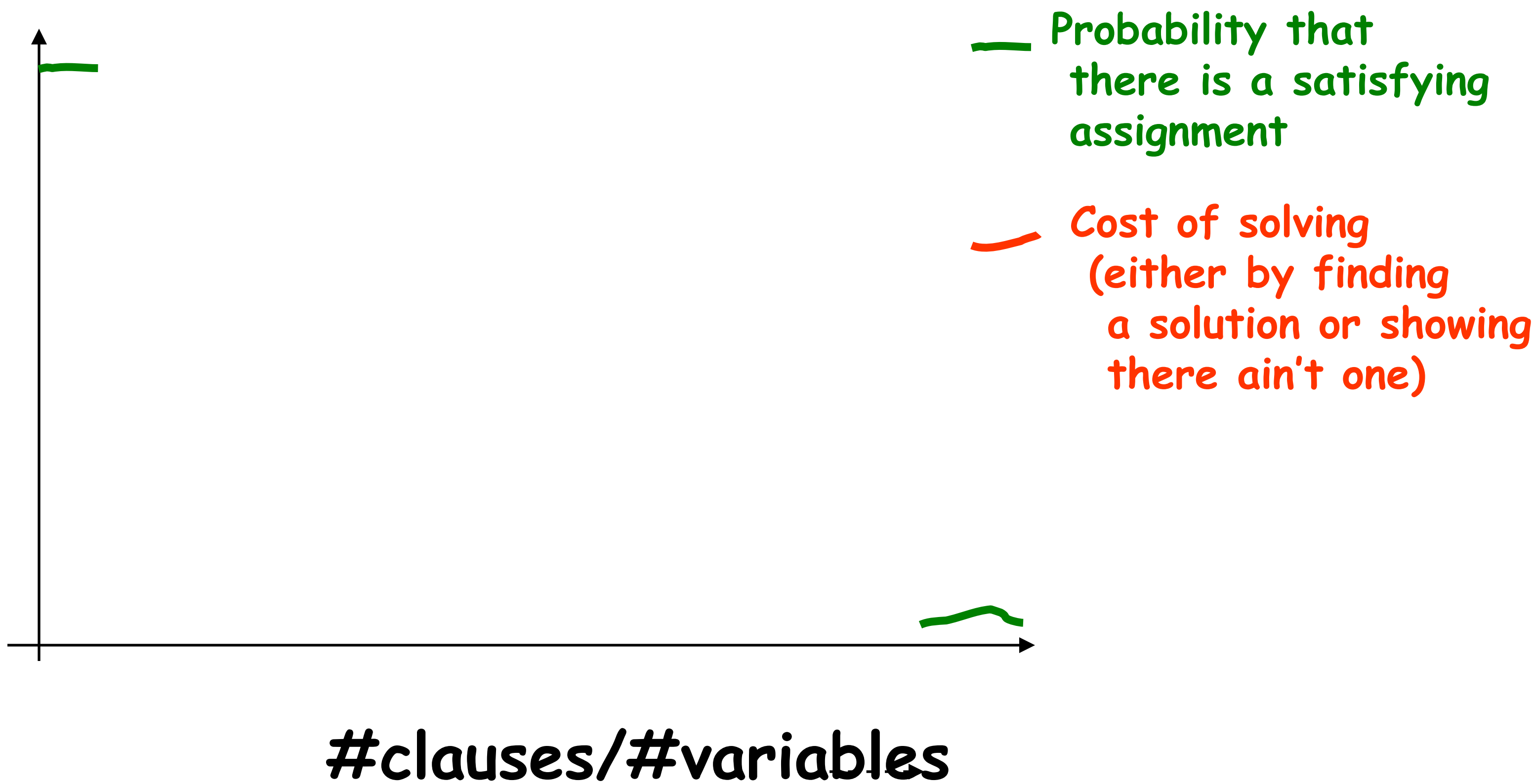
Hardness of 3-sat as a function of #clauses/#variables

Probability that there is a satisfying assignment

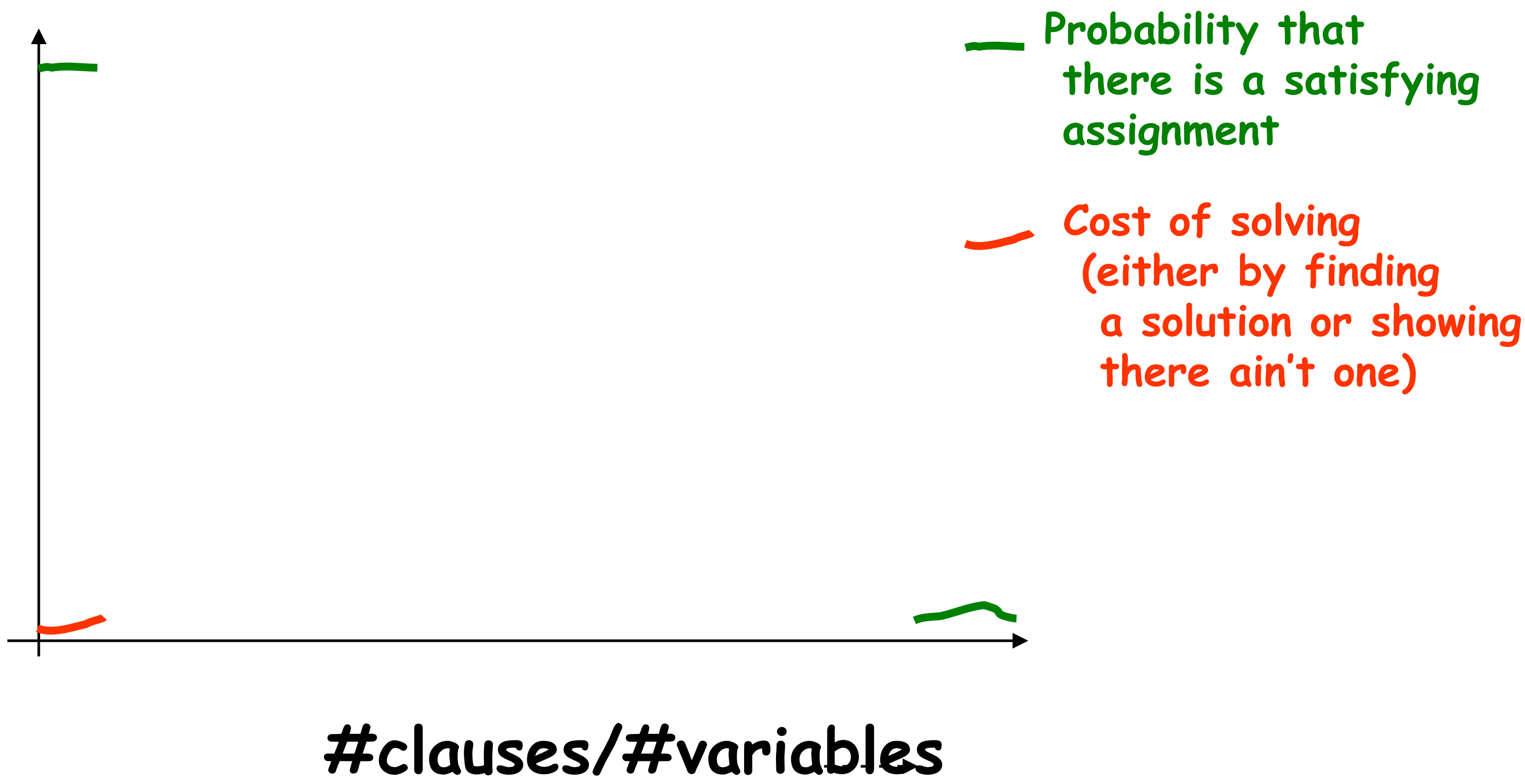
Cost of solving (either by finding a solution or showing there ain't one)

#clauses/#variables

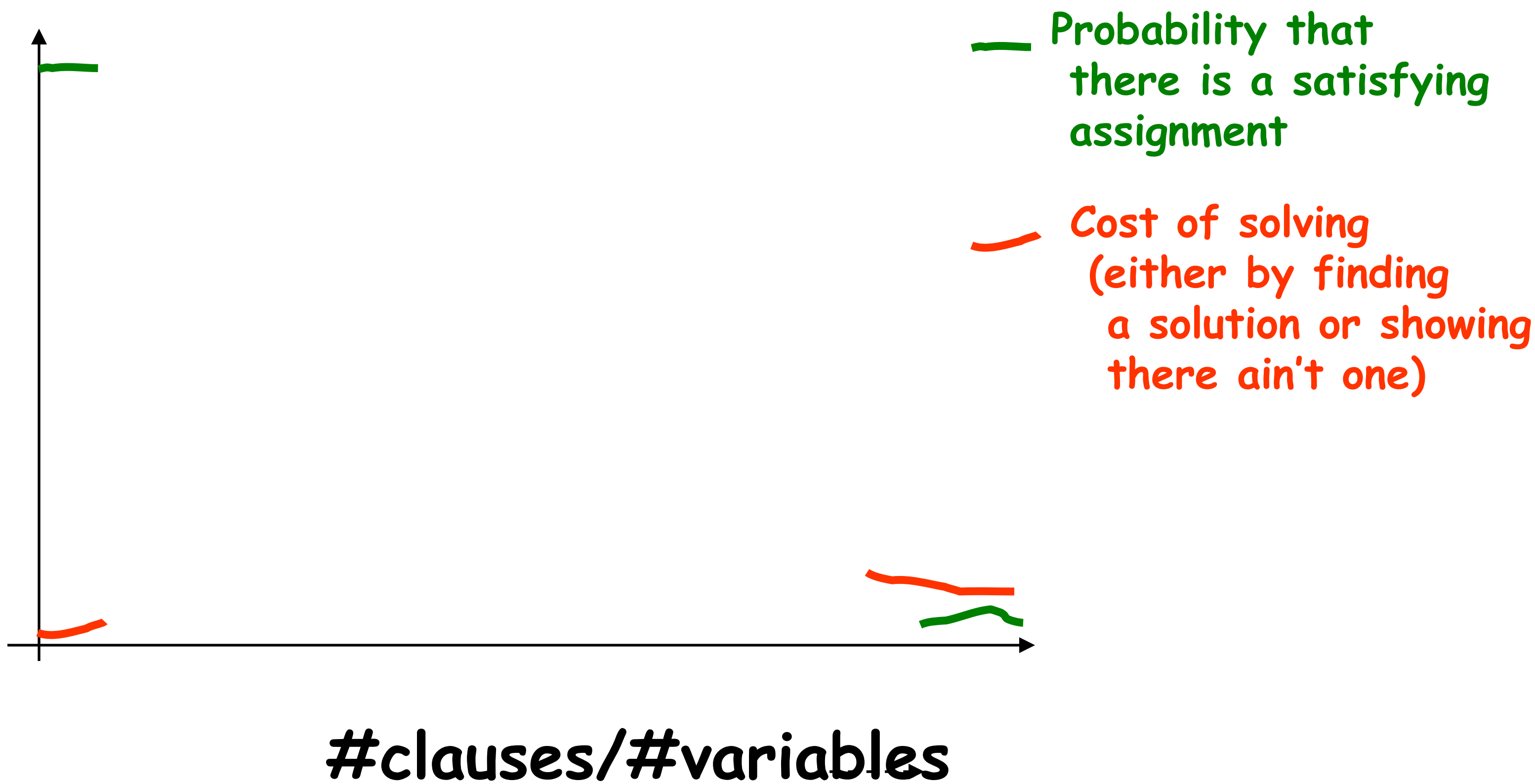
Hardness of 3-sat as a function of #clauses/#variables



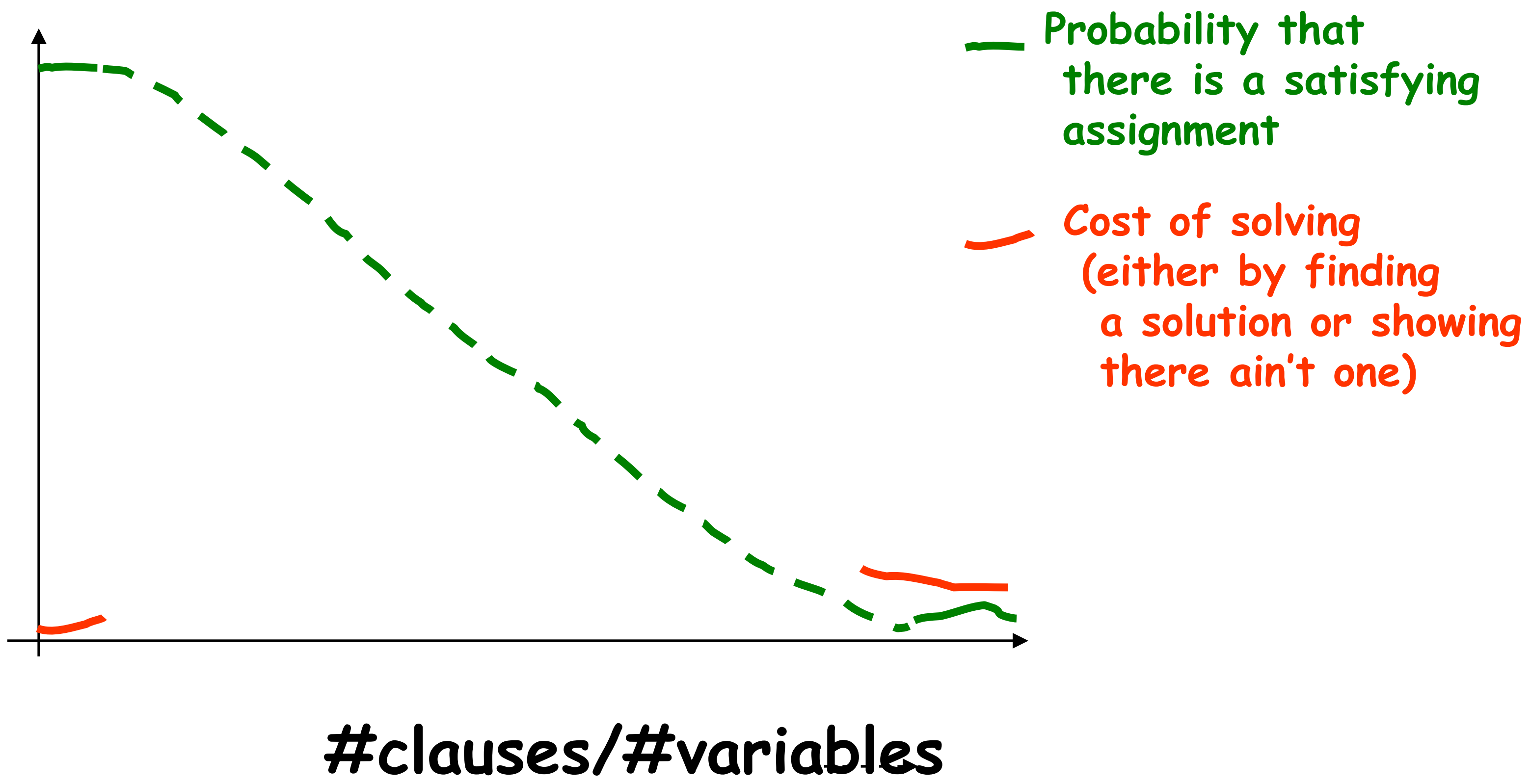
Hardness of 3-sat as a function of #clauses/#variables



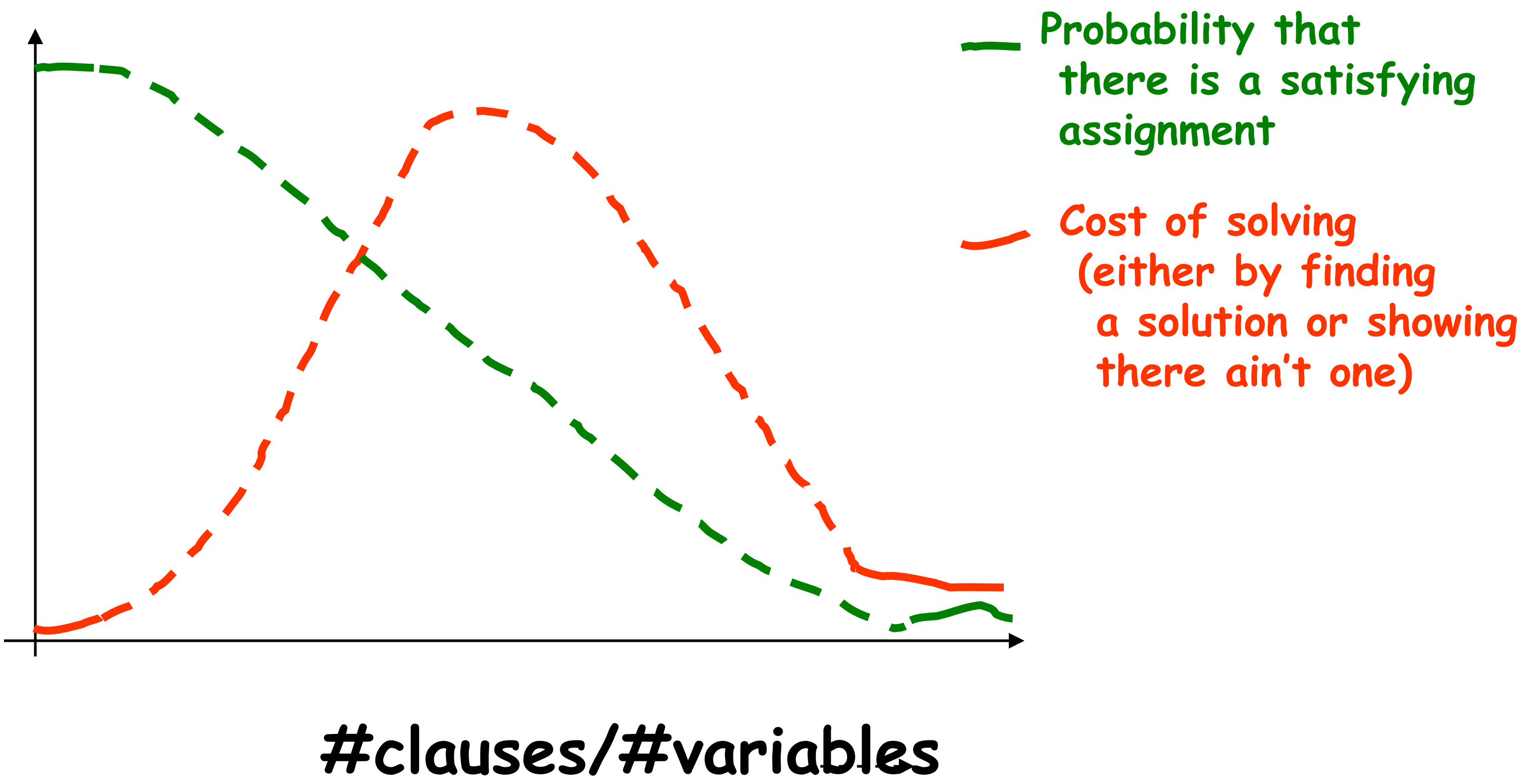
Hardness of 3-sat as a function of #clauses/#variables



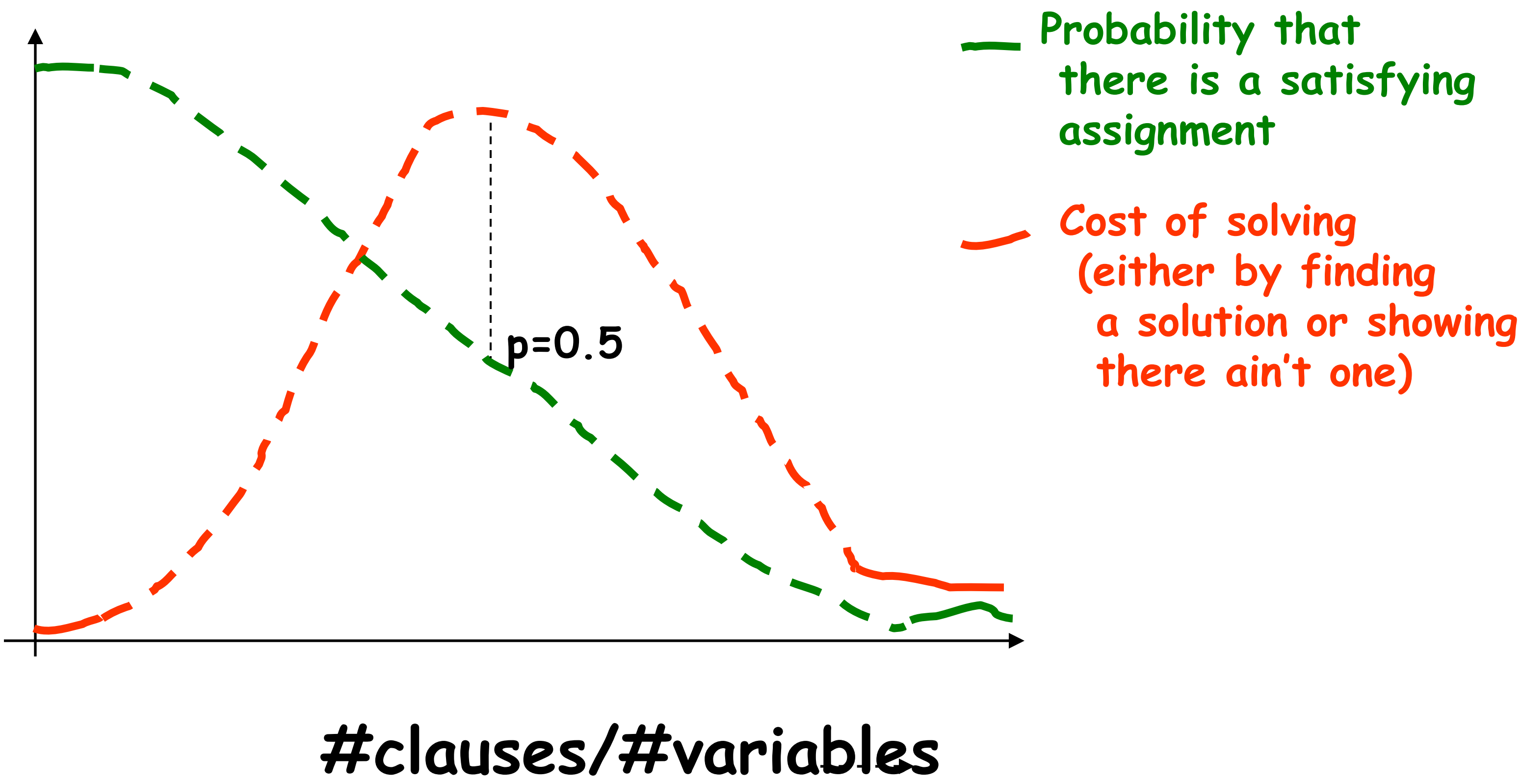
Hardness of 3-sat as a function of #clauses/#variables



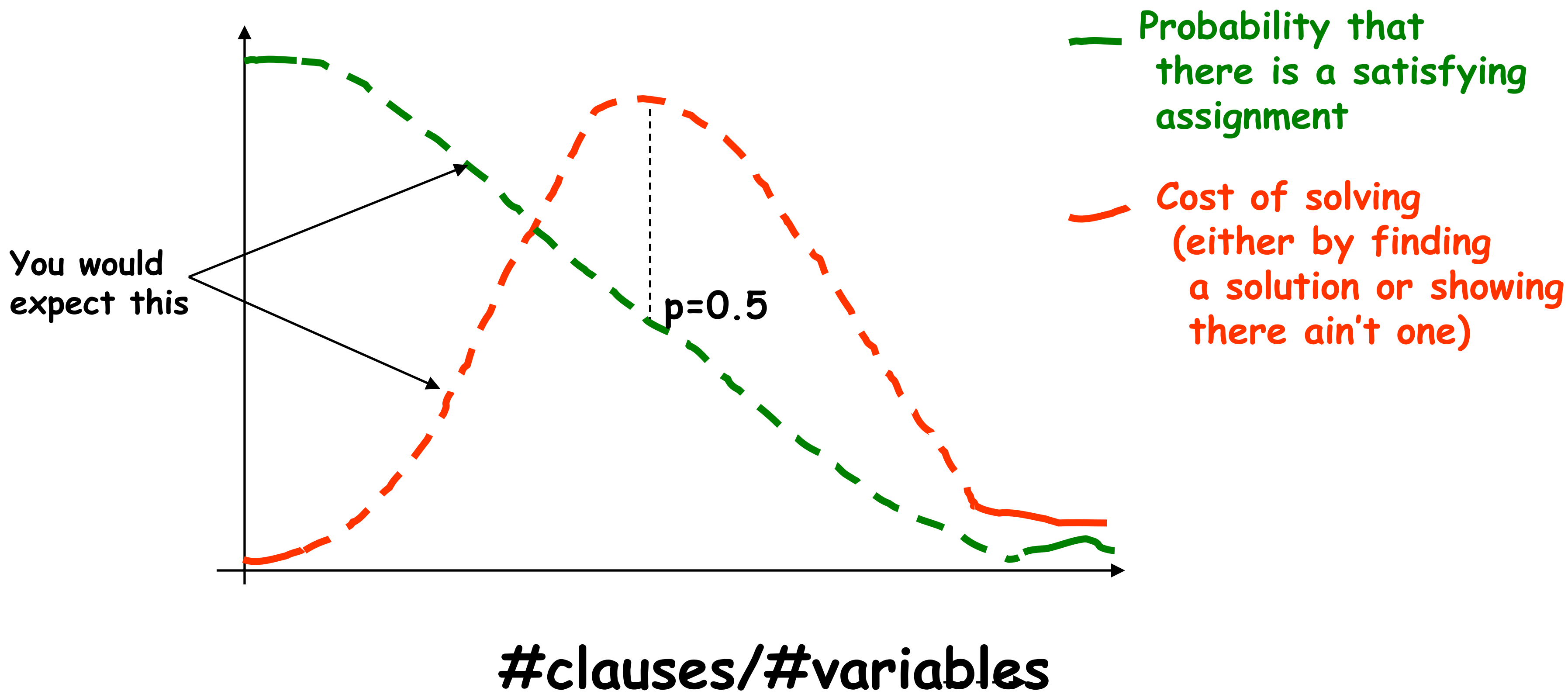
Hardness of 3-sat as a function of #clauses/#variables



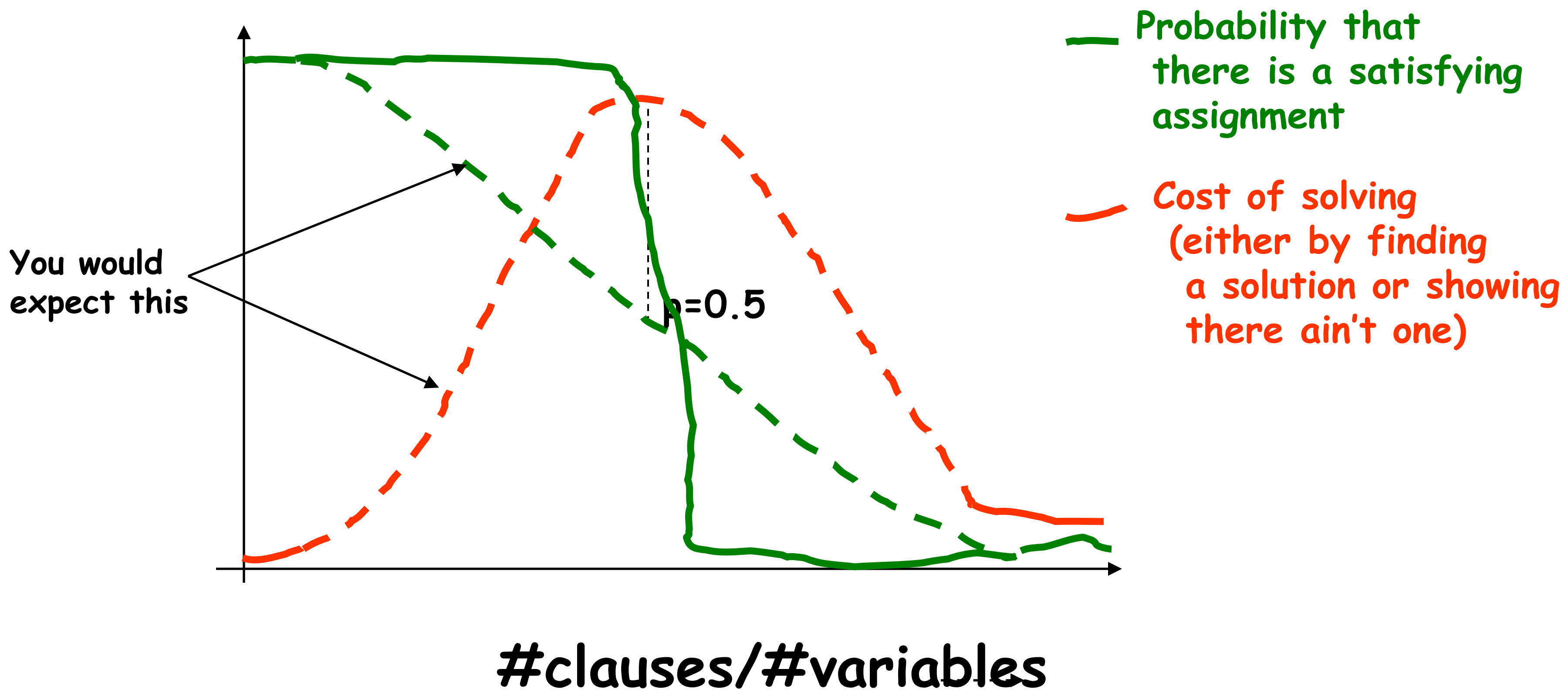
Hardness of 3-sat as a function of #clauses/#variables



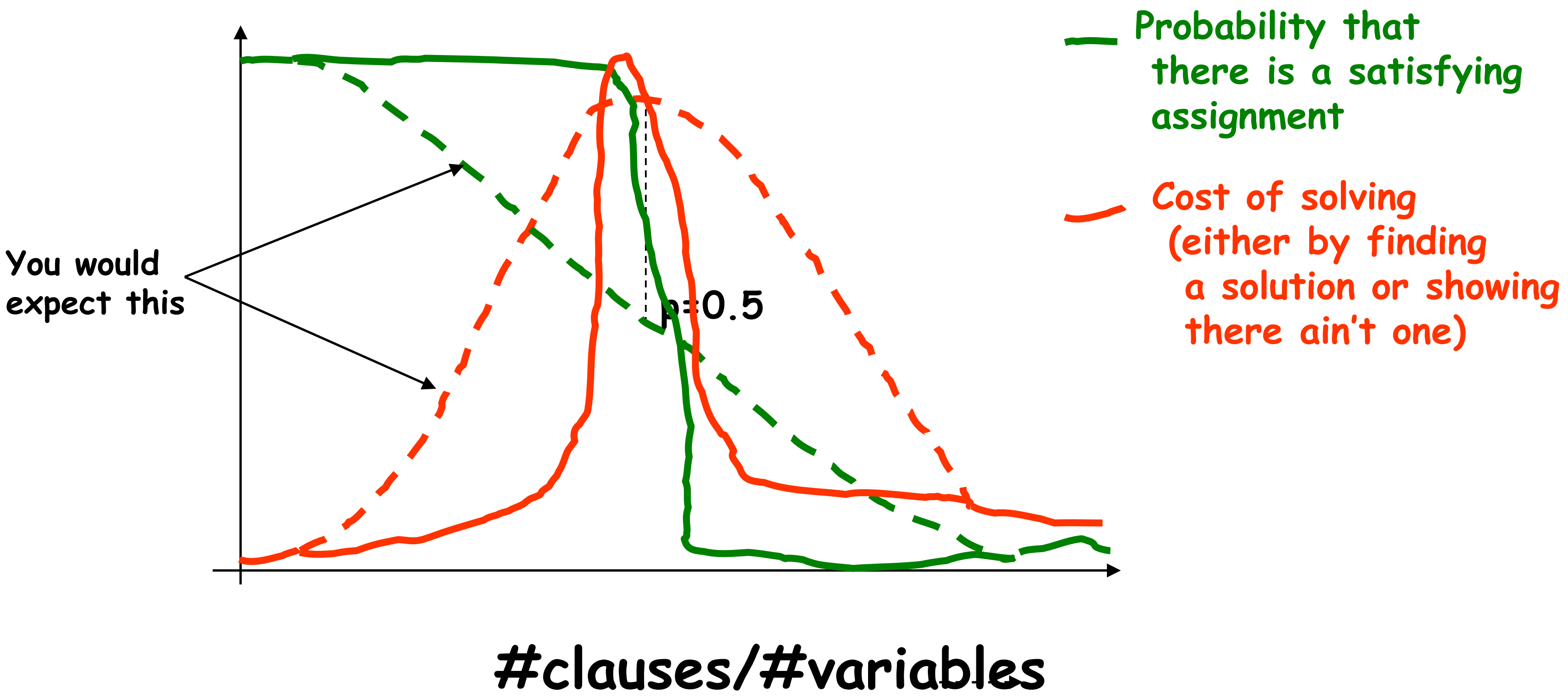
Hardness of 3-sat as a function of #clauses/#variables



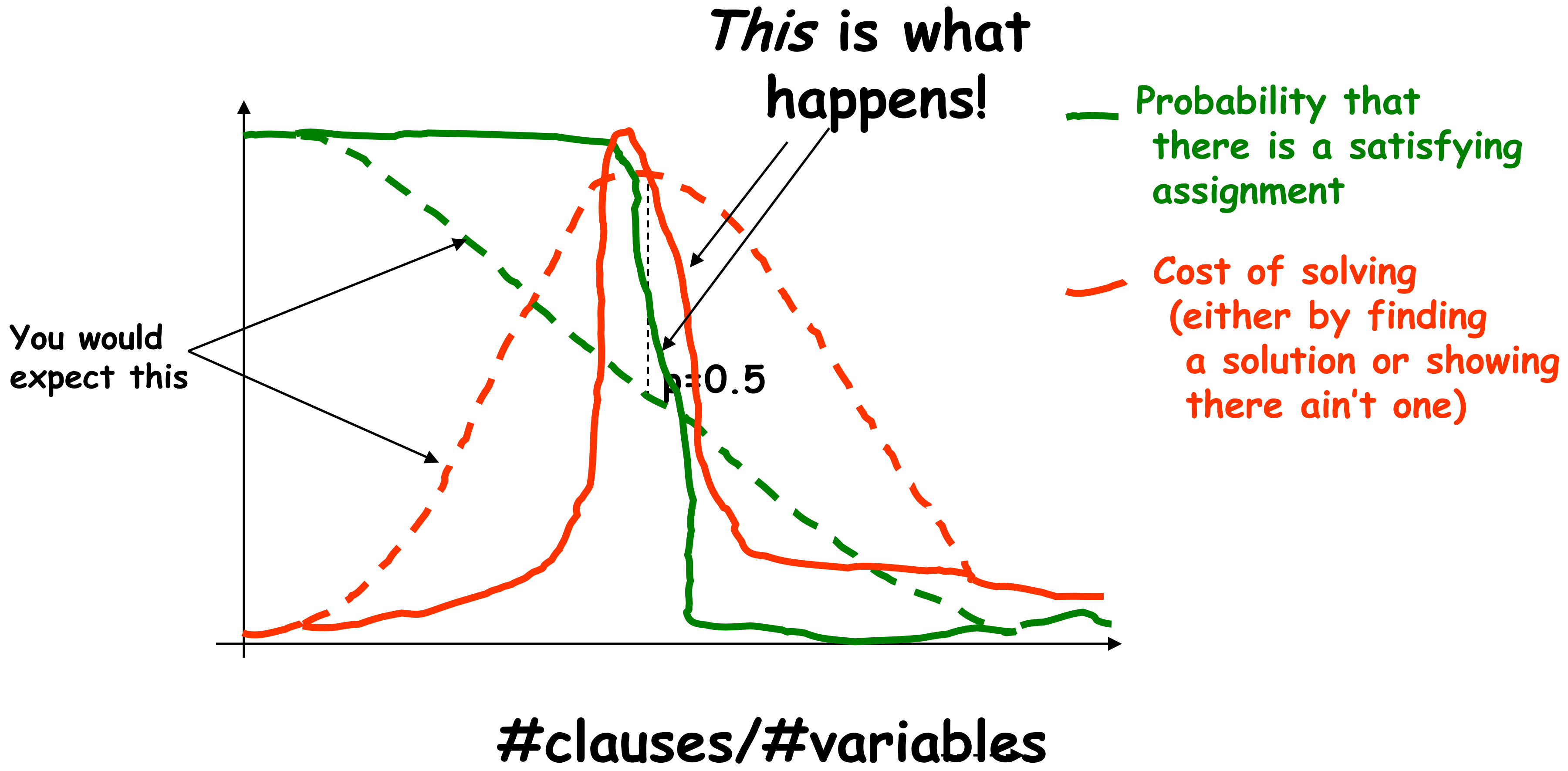
Hardness of 3-sat as a function of #clauses/#variables



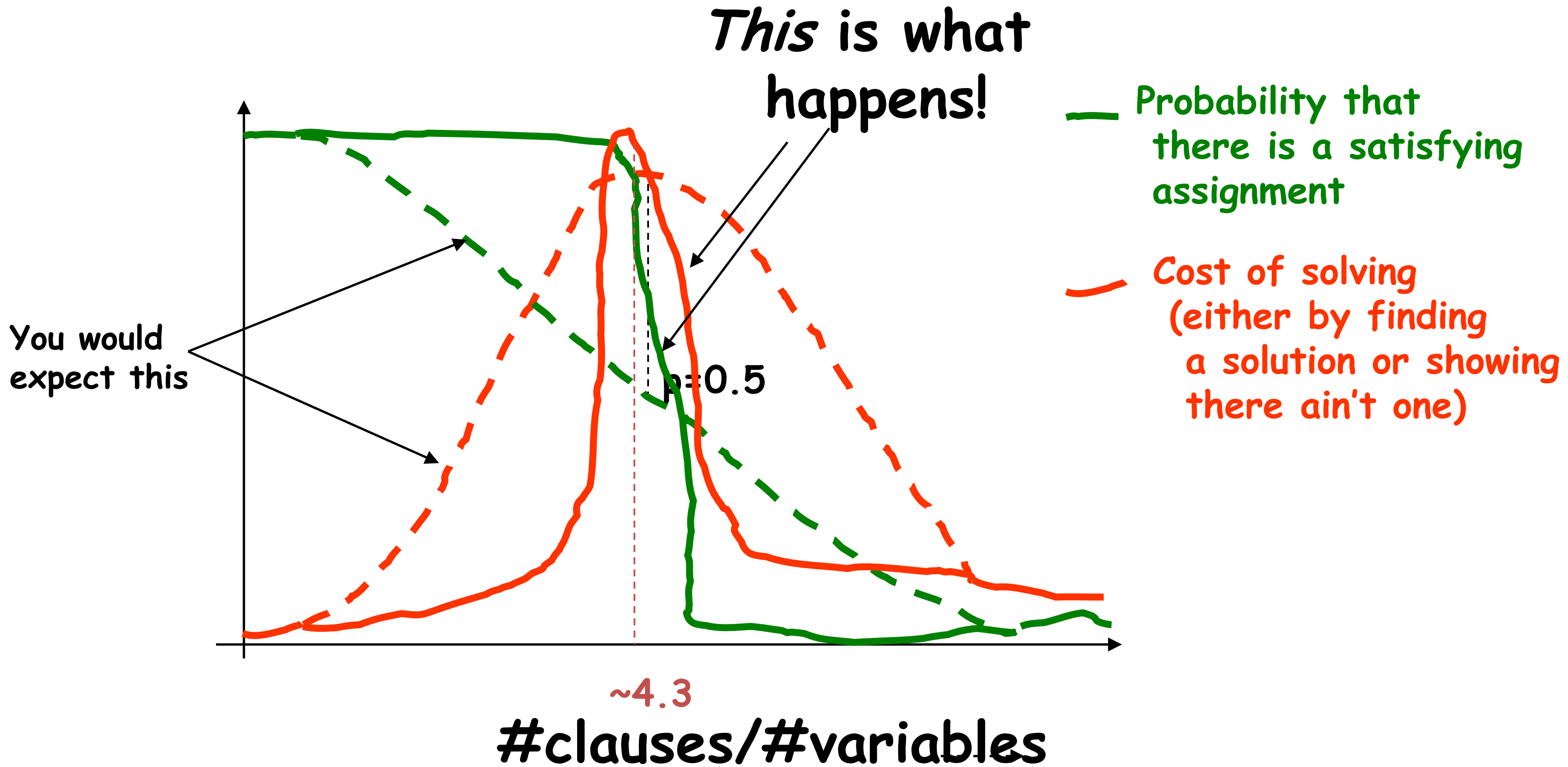
Hardness of 3-sat as a function of #clauses/#variables



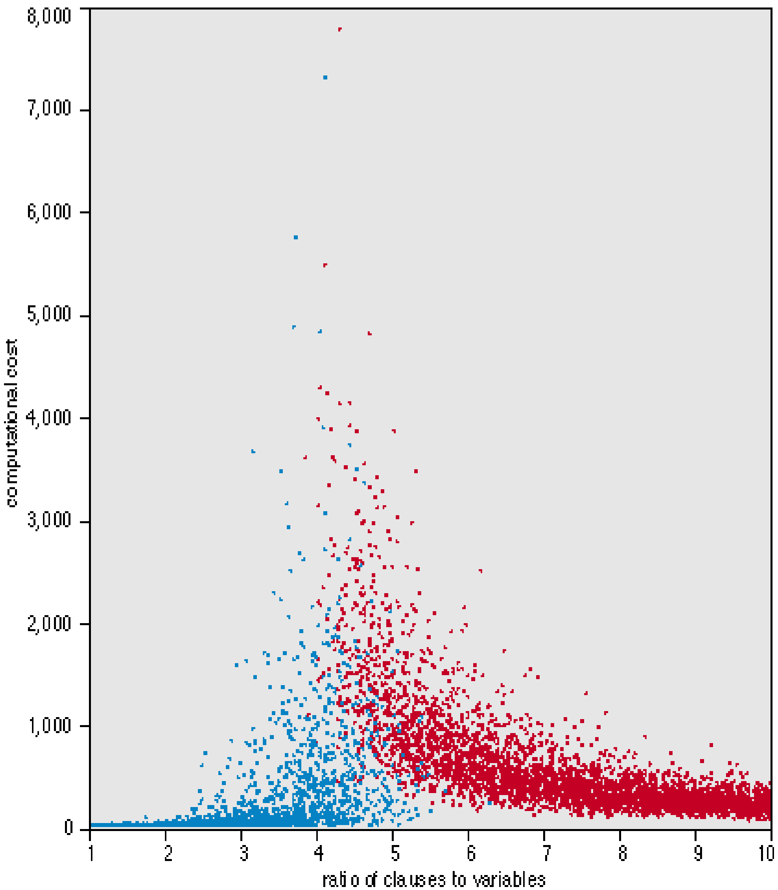
Hardness of 3-sat as a function of #clauses/#variables



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



Random 3-SAT

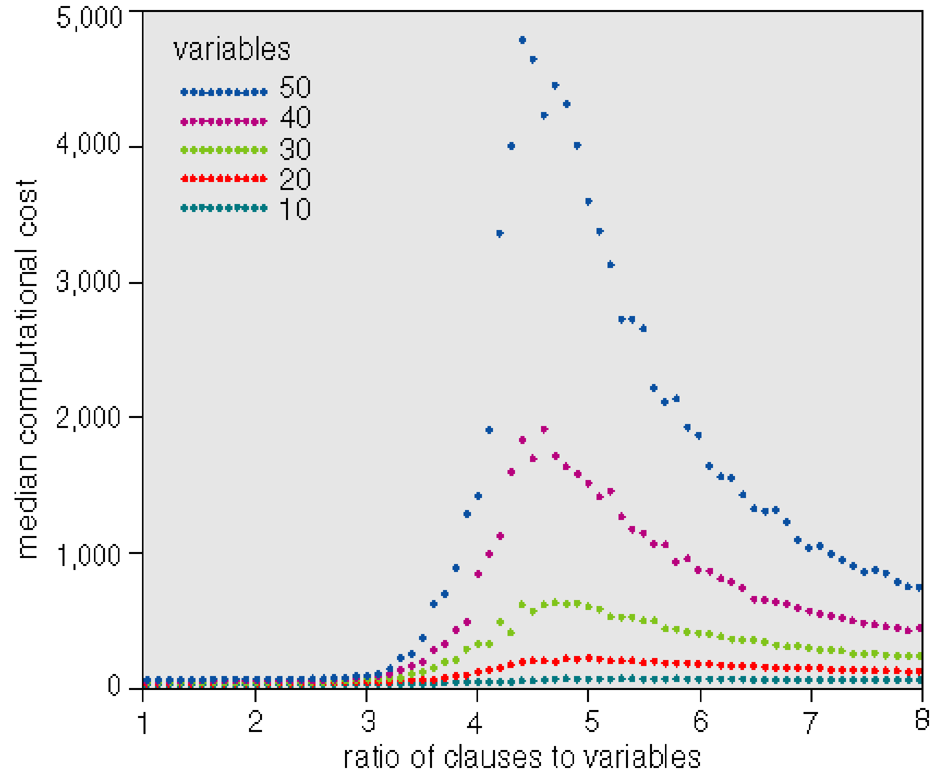


- Random 3-SAT
  - sample uniformly from space of all possible 3-clauses
  - *n* variables, *l* clauses
- Which are the hard instances?
  - around $l/n = 4.3$

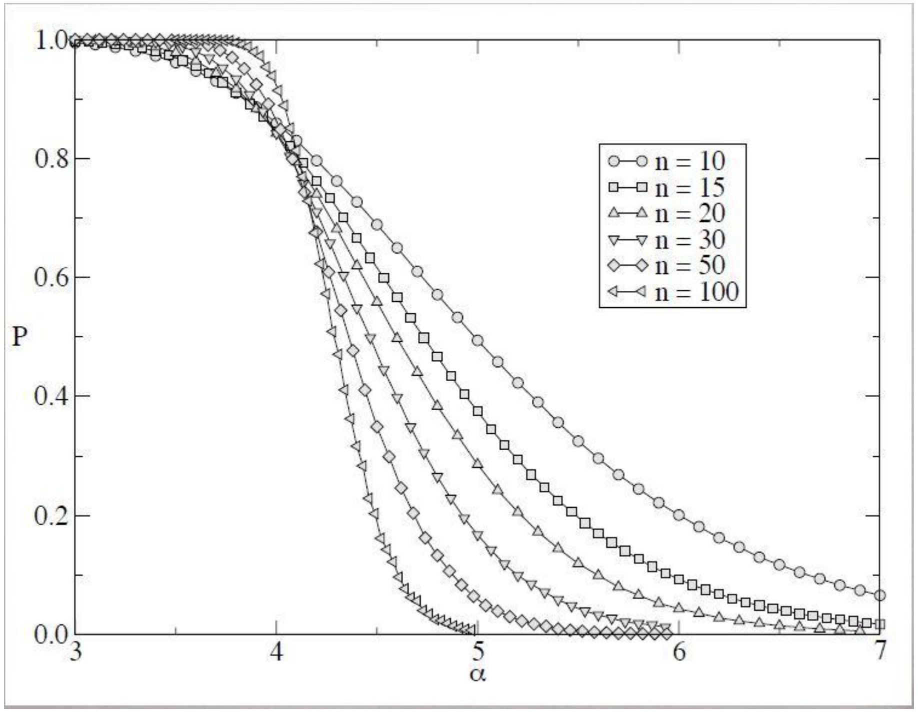
Random 3-SAT

- Varying problem size, *n*

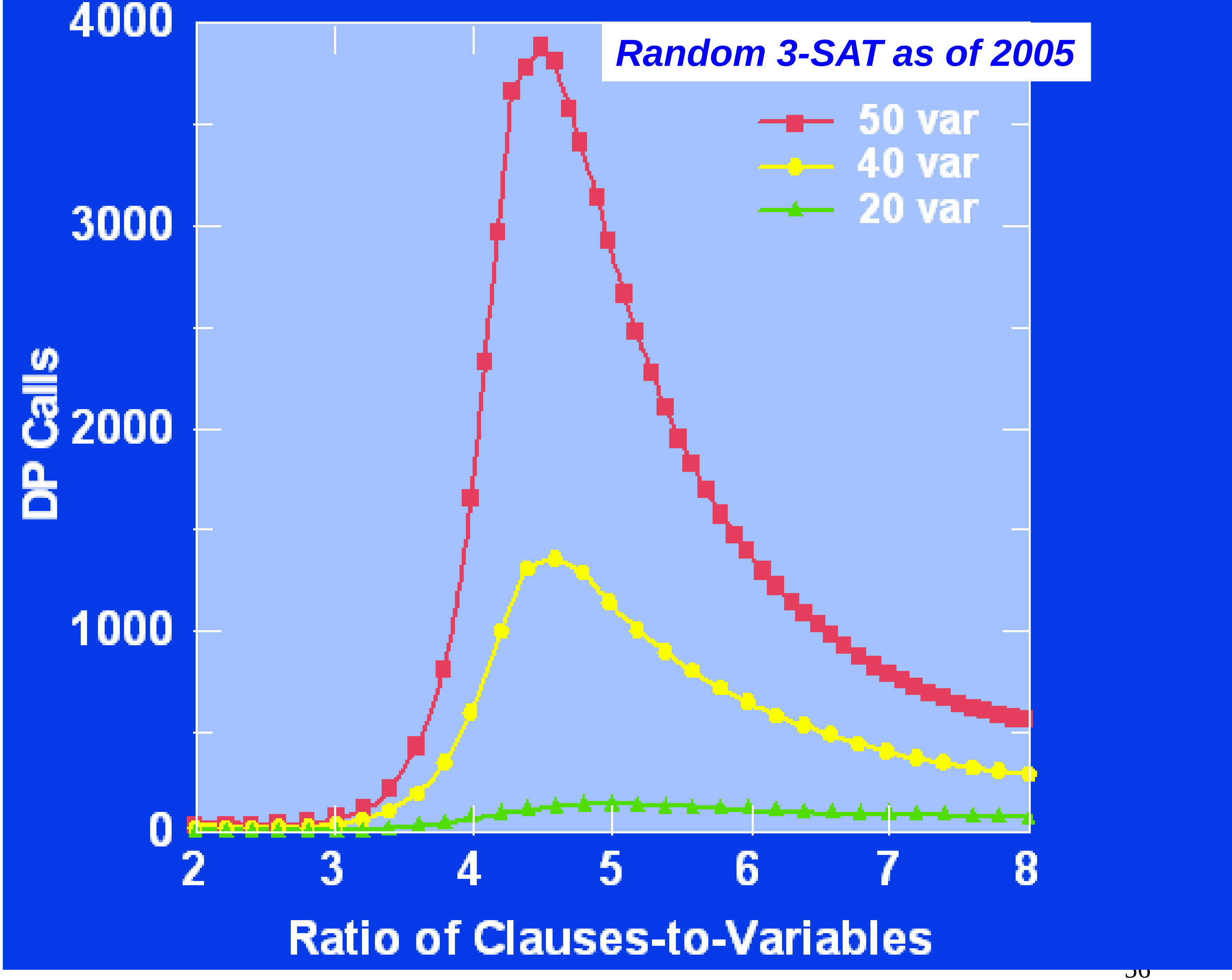
- Complexity peak appears to be largely invariant of algorithm



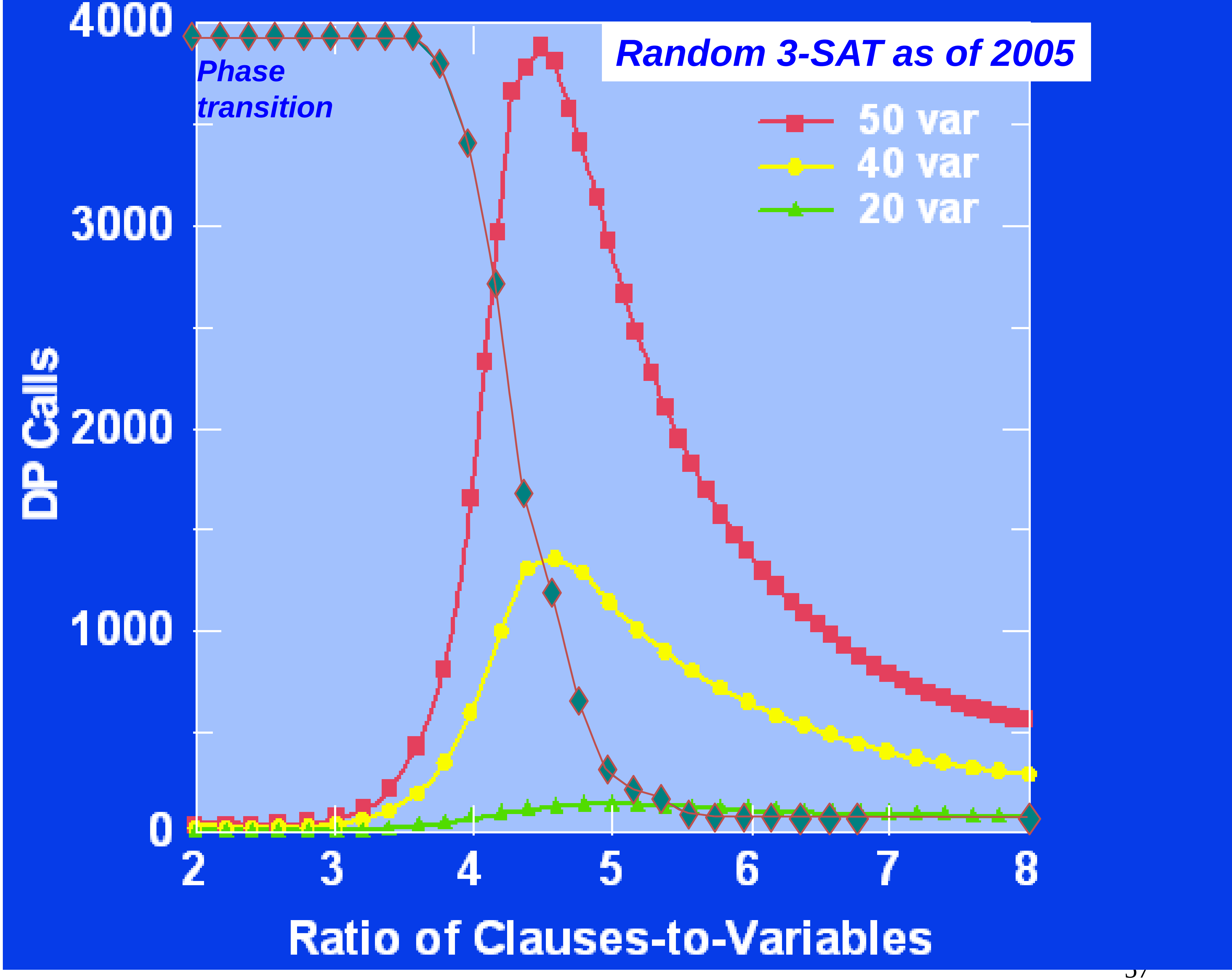
Random 3-SAT



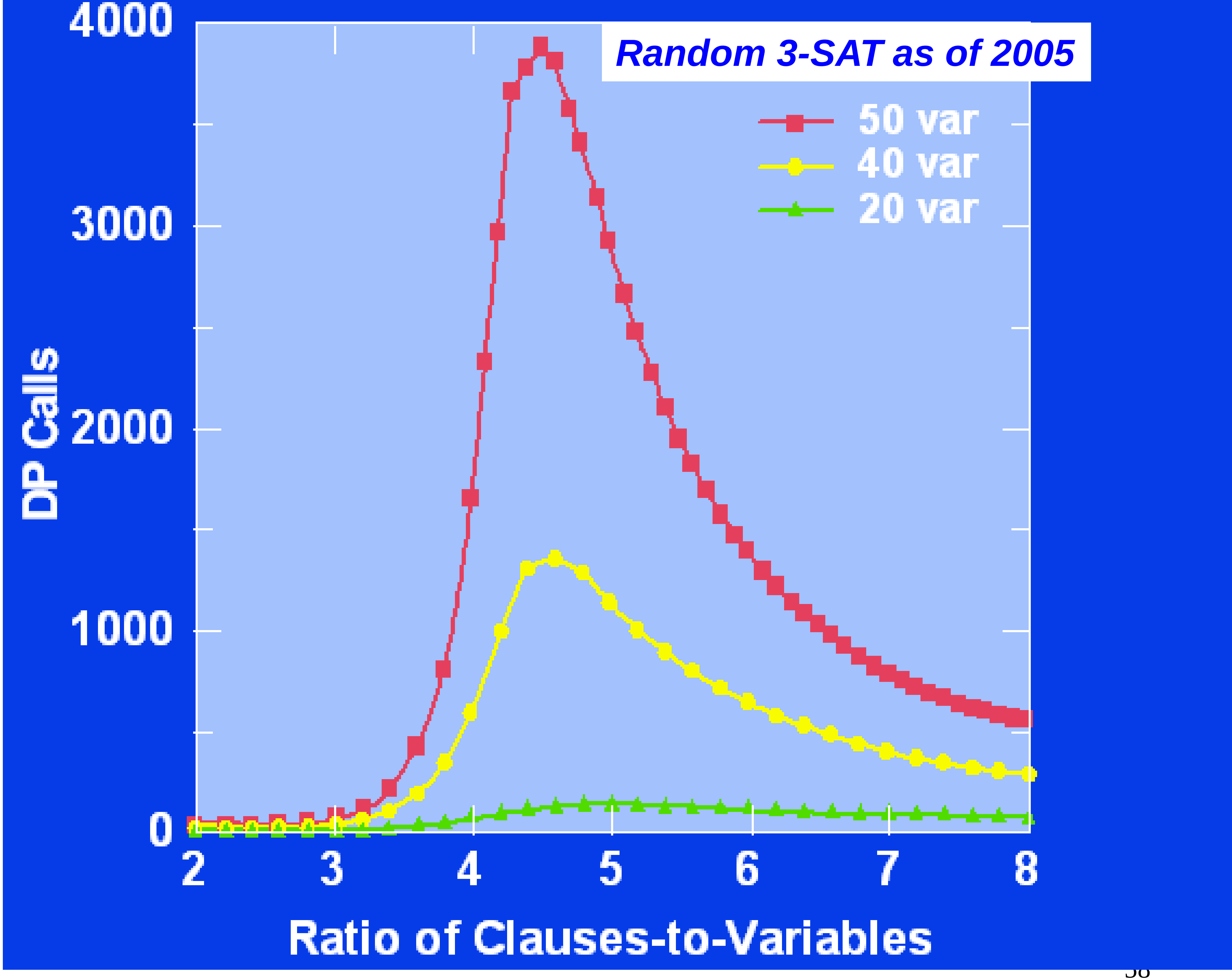
- Complexity peak coincides with solubility transition
  - l/n < 4.3 problems under-constrained and SAT
  - l/n > 4.3 problems over-constrained and UNSAT
  - l/n=4.3, problems on “knife-edge” between SAT and UNSAT



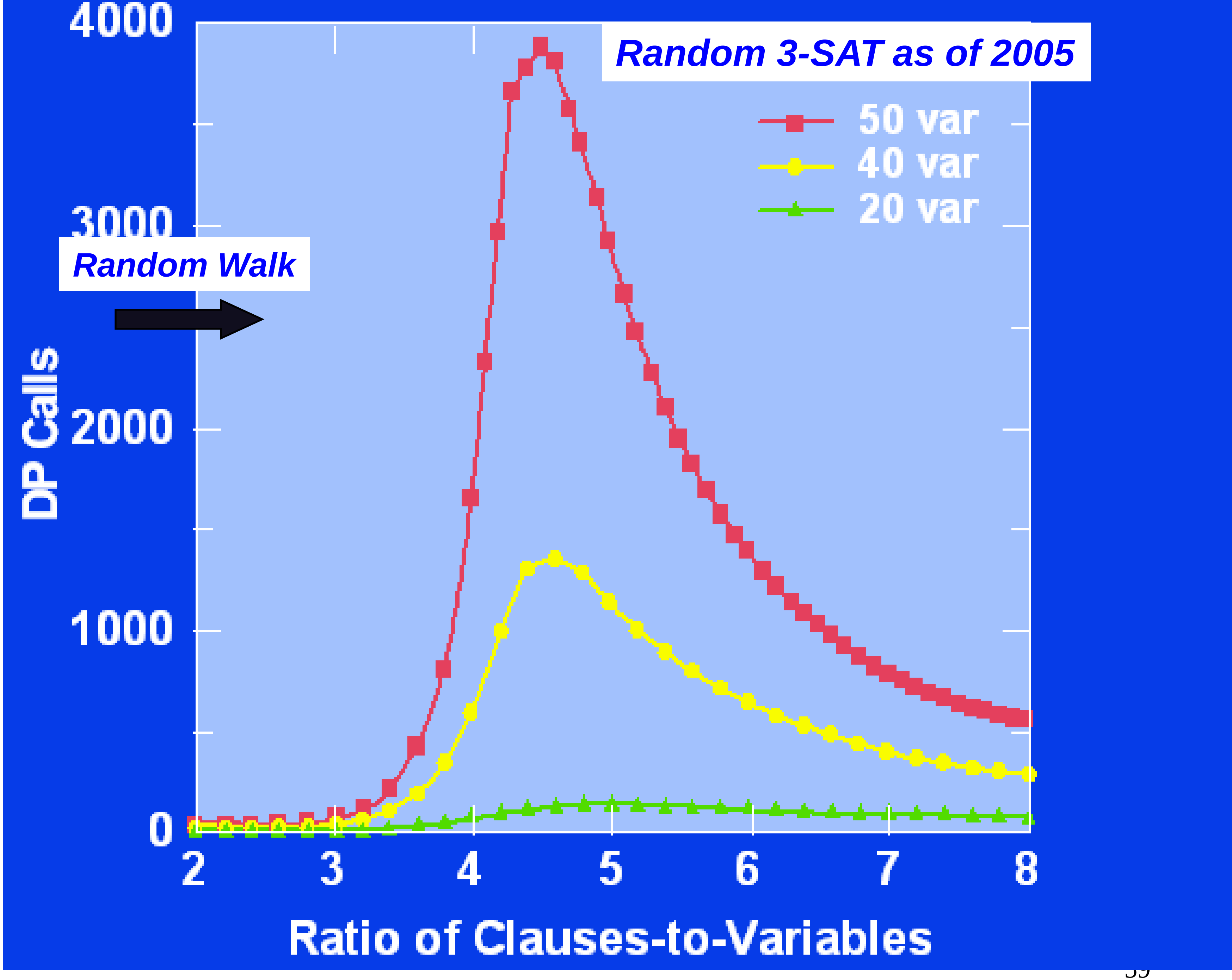
Mitchell, Selman, and Levesque '92



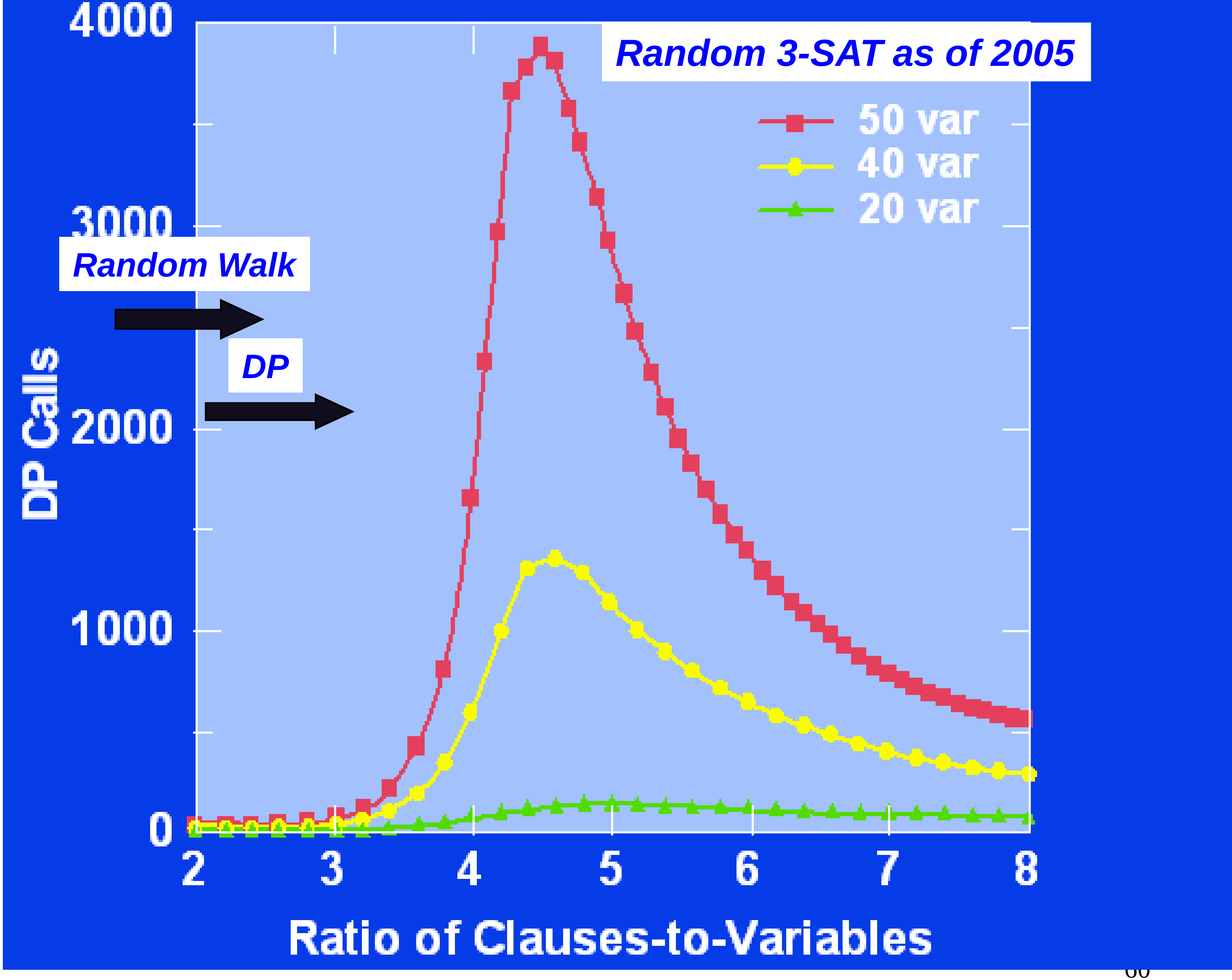
Mitchell, Selman, and Levesque '92



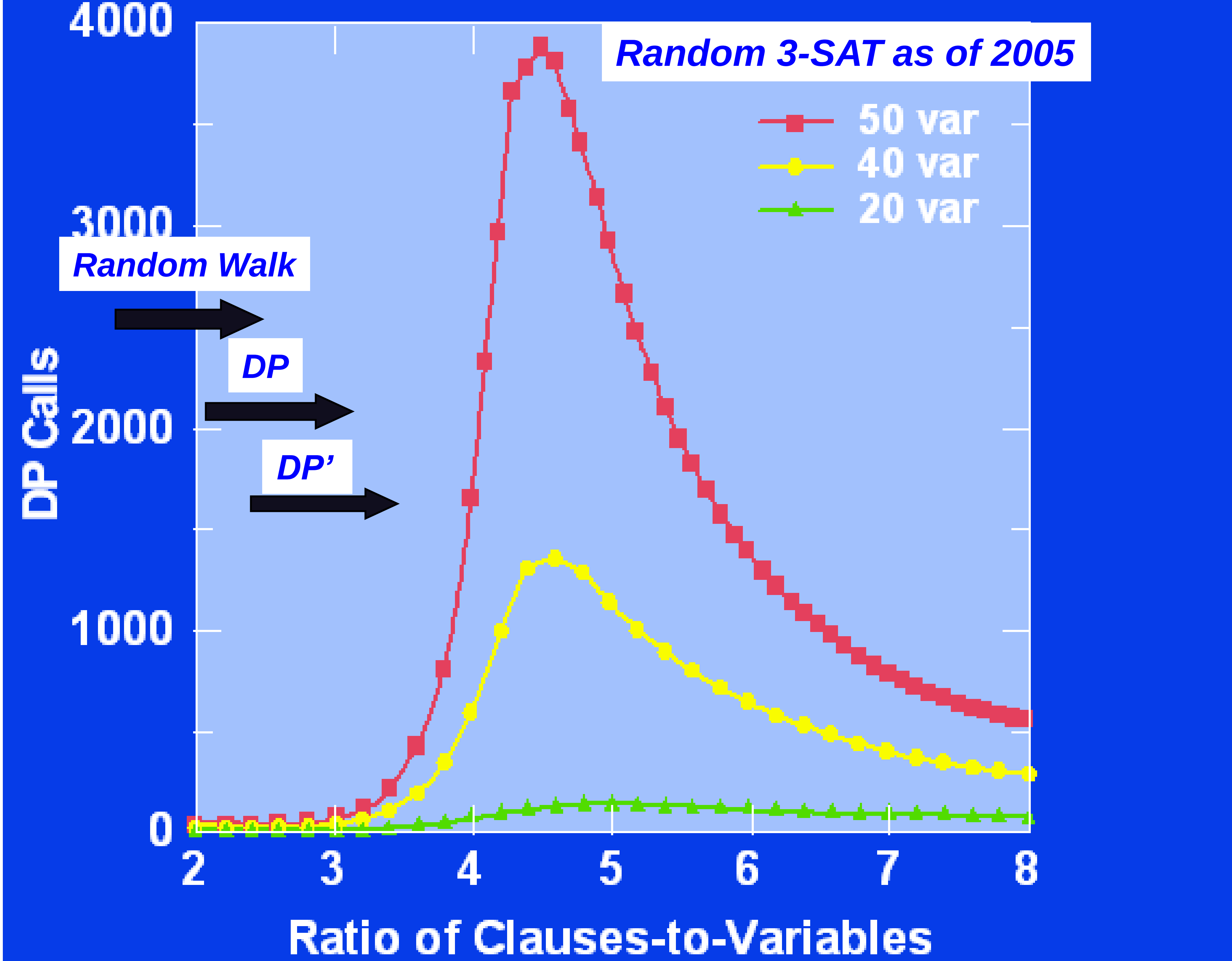
Mitchell, Selman, and Levesque ’92



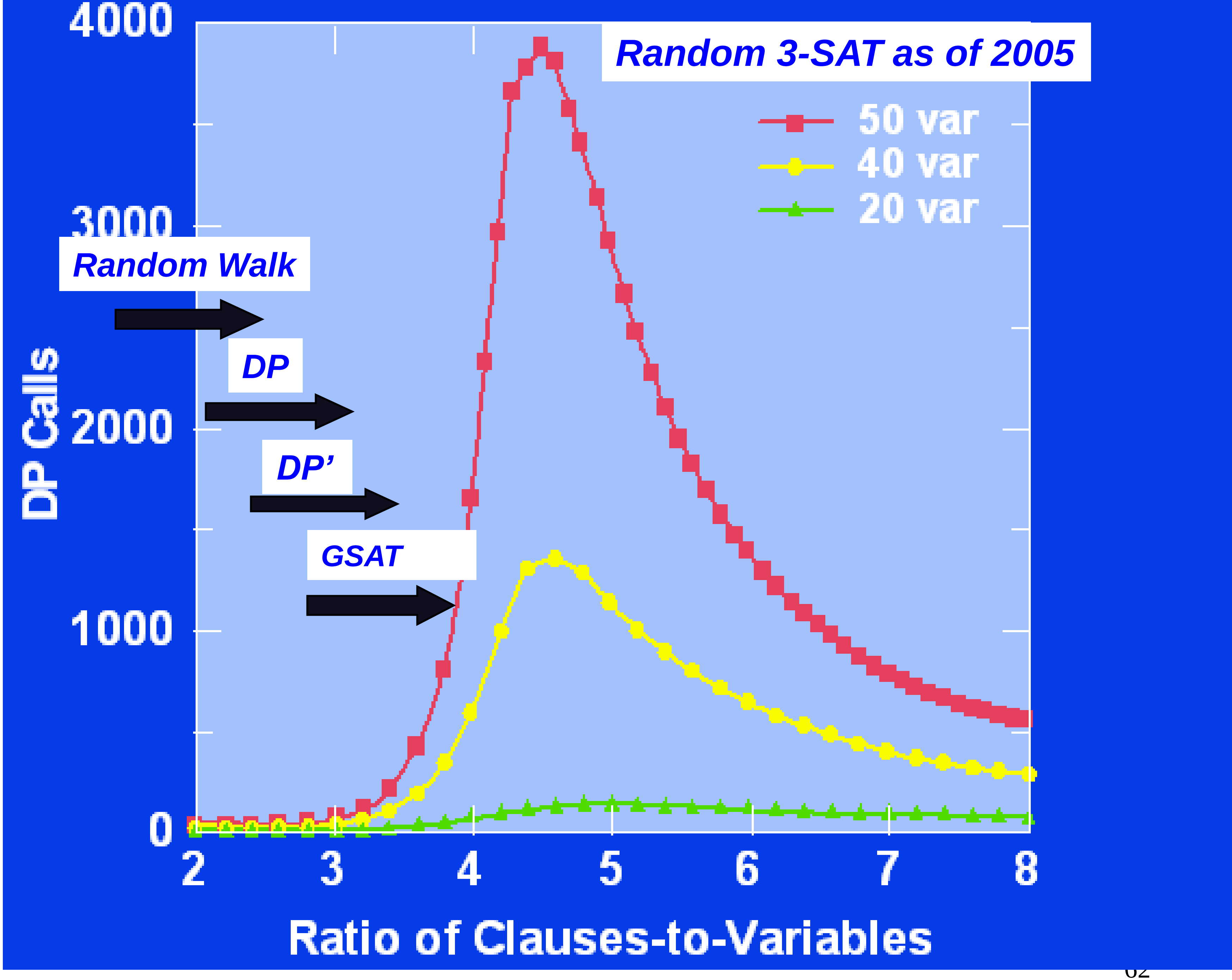
Mitchell, Selman, and Levesque '92



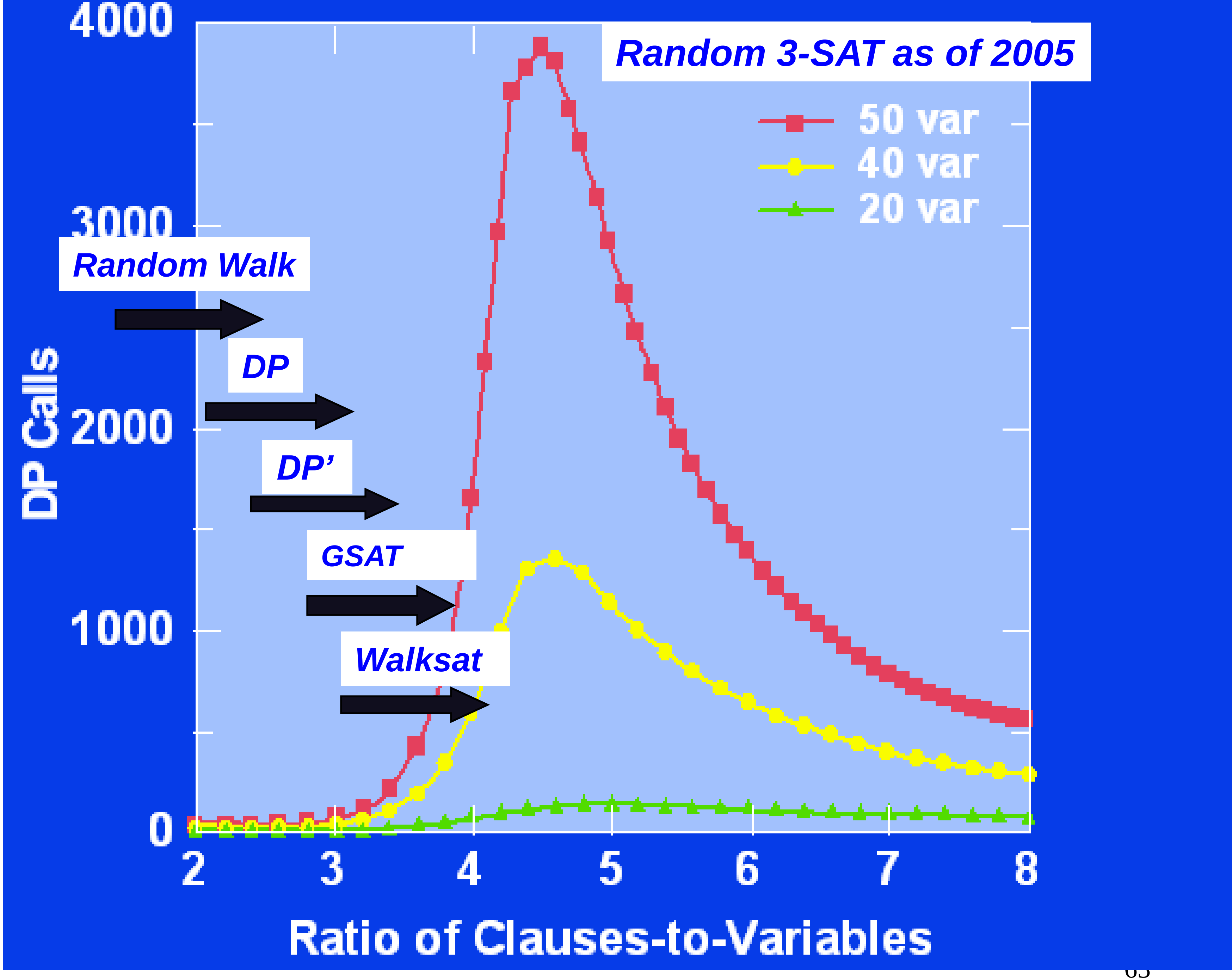
Mitchell, Selman, and Levesque '92



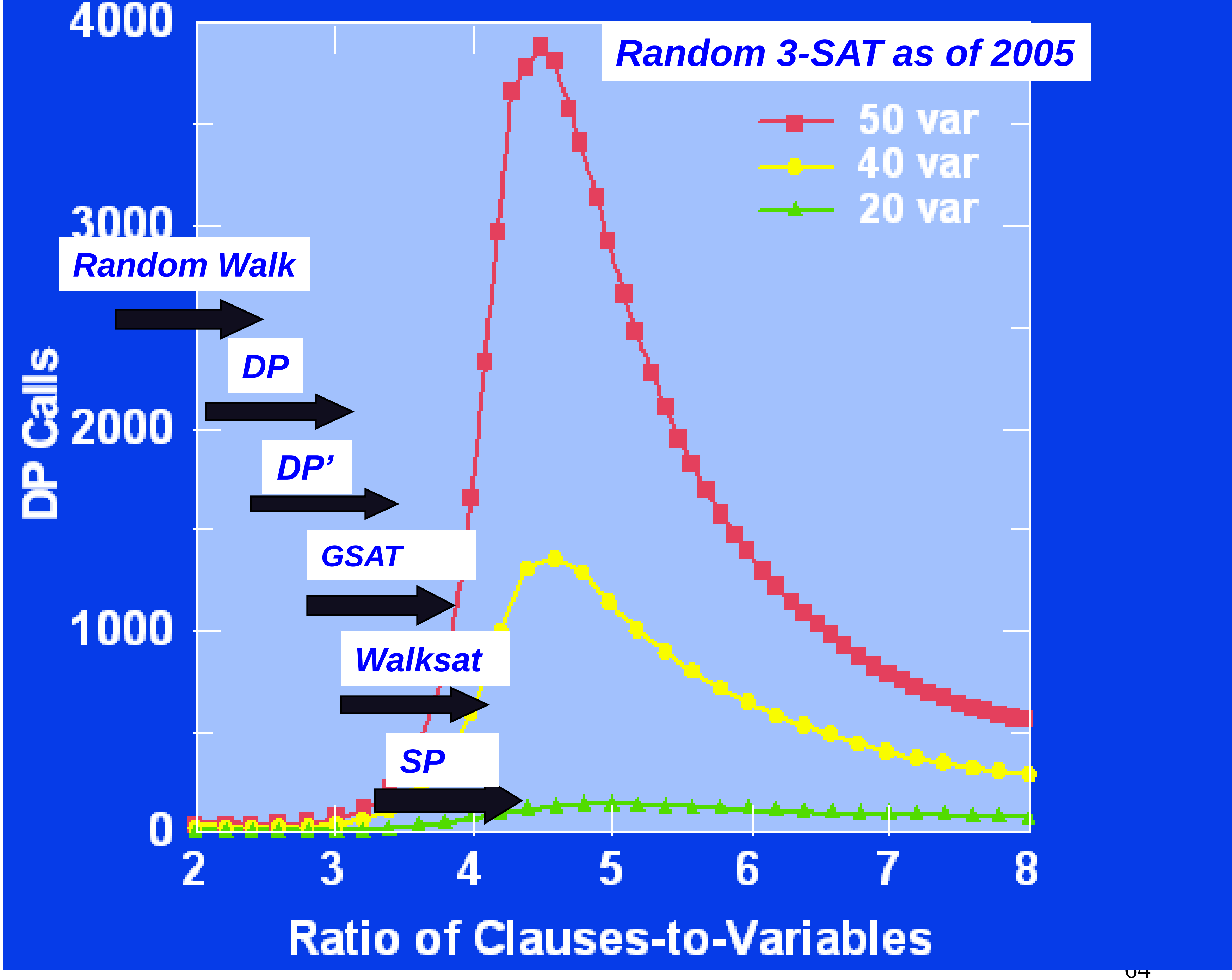
Mitchell, Selman, and Levesque '92



Mitchell, Selman, and Levesque '92



Mitchell, Selman, and Levesque '92



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

    prob(X>=1) <= E[X]

    E[X] = 0.p(X=0) + 1.p(X=1) + 2.p(X=2) + 3.p(X=3) + …

    >= 1.p(X=1) + 1.p(X=2) + 1.p(X=3) + …

    >= p(X>=1)

3SAT phase transition

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

    For any statistic X

    $prob(X>=1) <= E[X]$

    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

    prob(X>=1) <= 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

    prob(X>=1) <= 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

    prob(X>=1) <= 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

    prob(X>=1) <= E[X]

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

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

    If E[X] < 1, then prob(X>=1) = prob(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

    prob(X>=1) <= E[X]

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

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

    If $E[X] < 1$, 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

    prob(X>=1) <= E[X]

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

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

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

    $n + l \log2(7/8) < 0$

3SAT phase transition

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

    For any statistic X

    prob(X>=1) <= E[X]

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

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

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

    $n + l \log 2(7/8) < 0$

    $l/n > 1/\log 2(8/7) = 5.19\ldots$

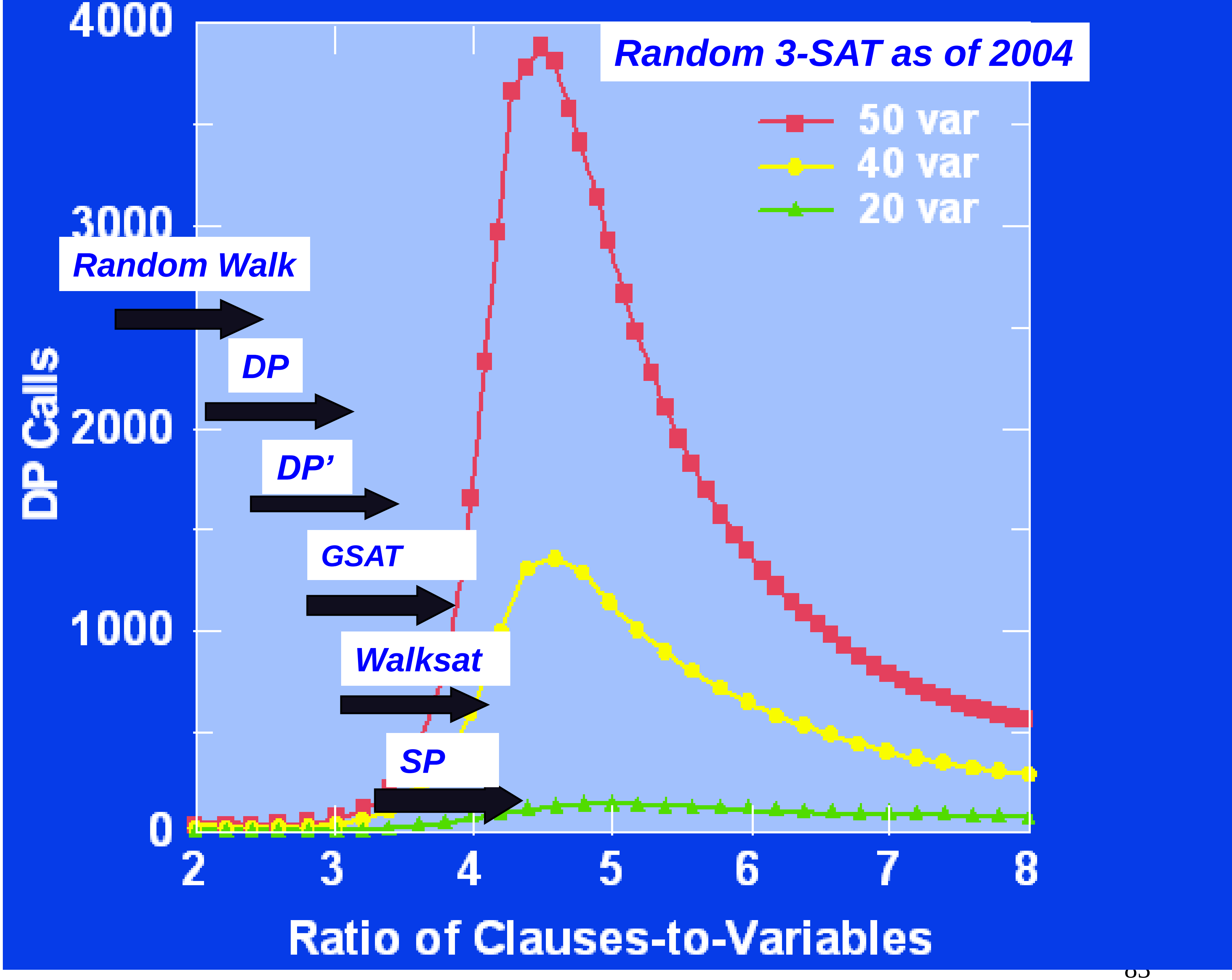
A Heavy Tail

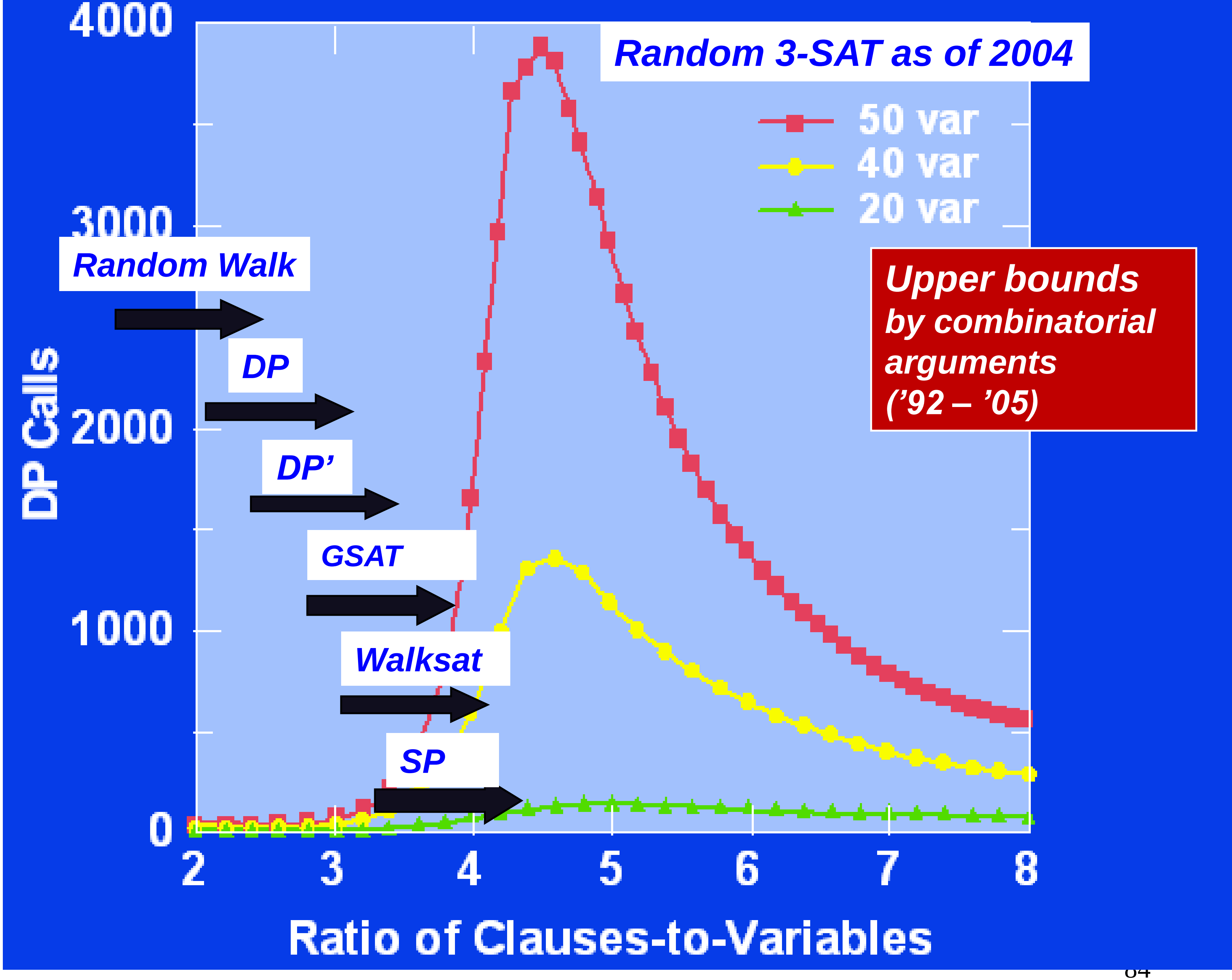
- But the transition is much lower at l/n ~ 4.27. What going on?

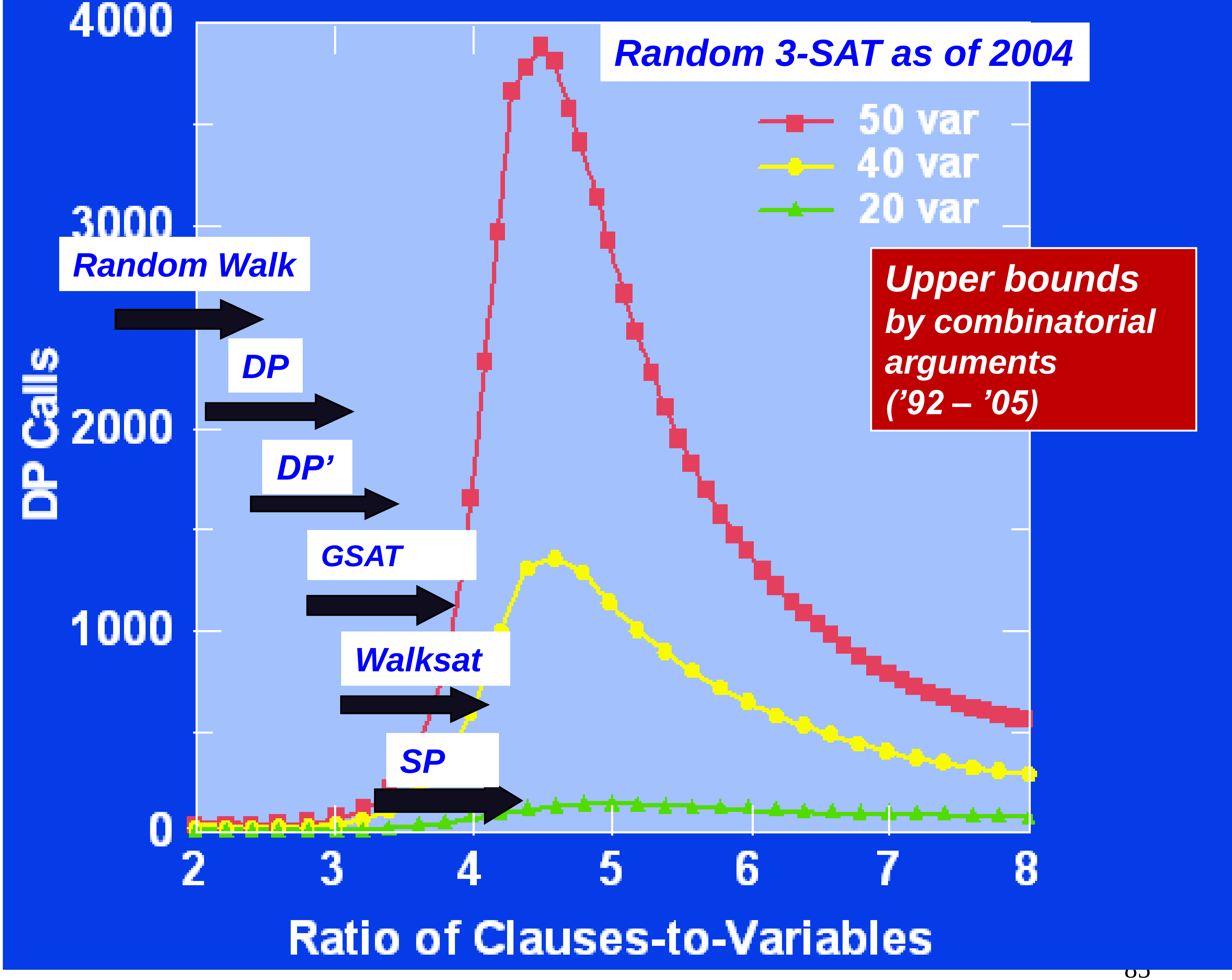
A Heavy Tail

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

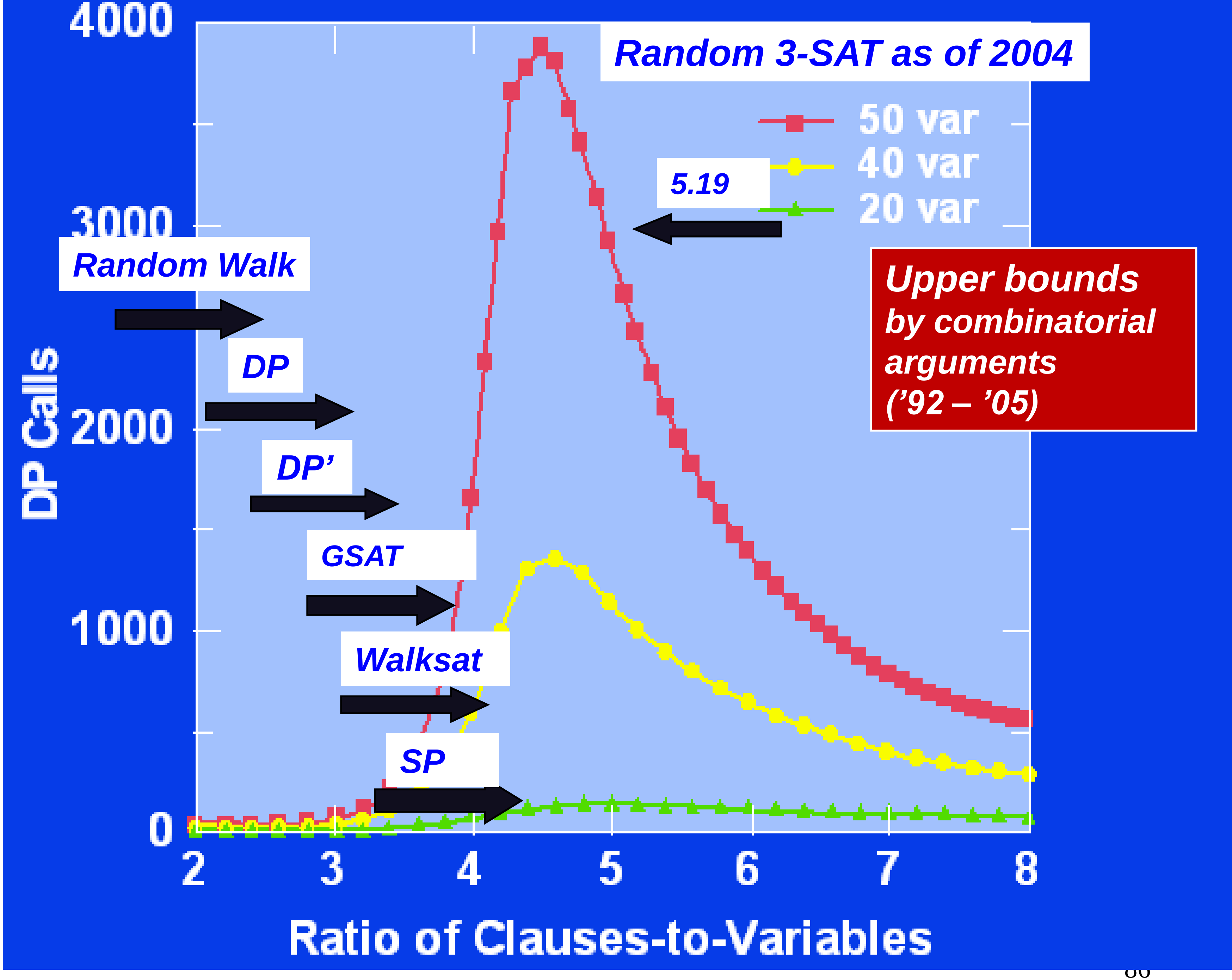
- In the range 4.27 < l/n < 5.19,
  - the average no. of solutions is exponentially large.
- Occasionally, there are exponentially many...
  - ...but most of the time there are none!
- Large average doesn’t prove satisfiability!

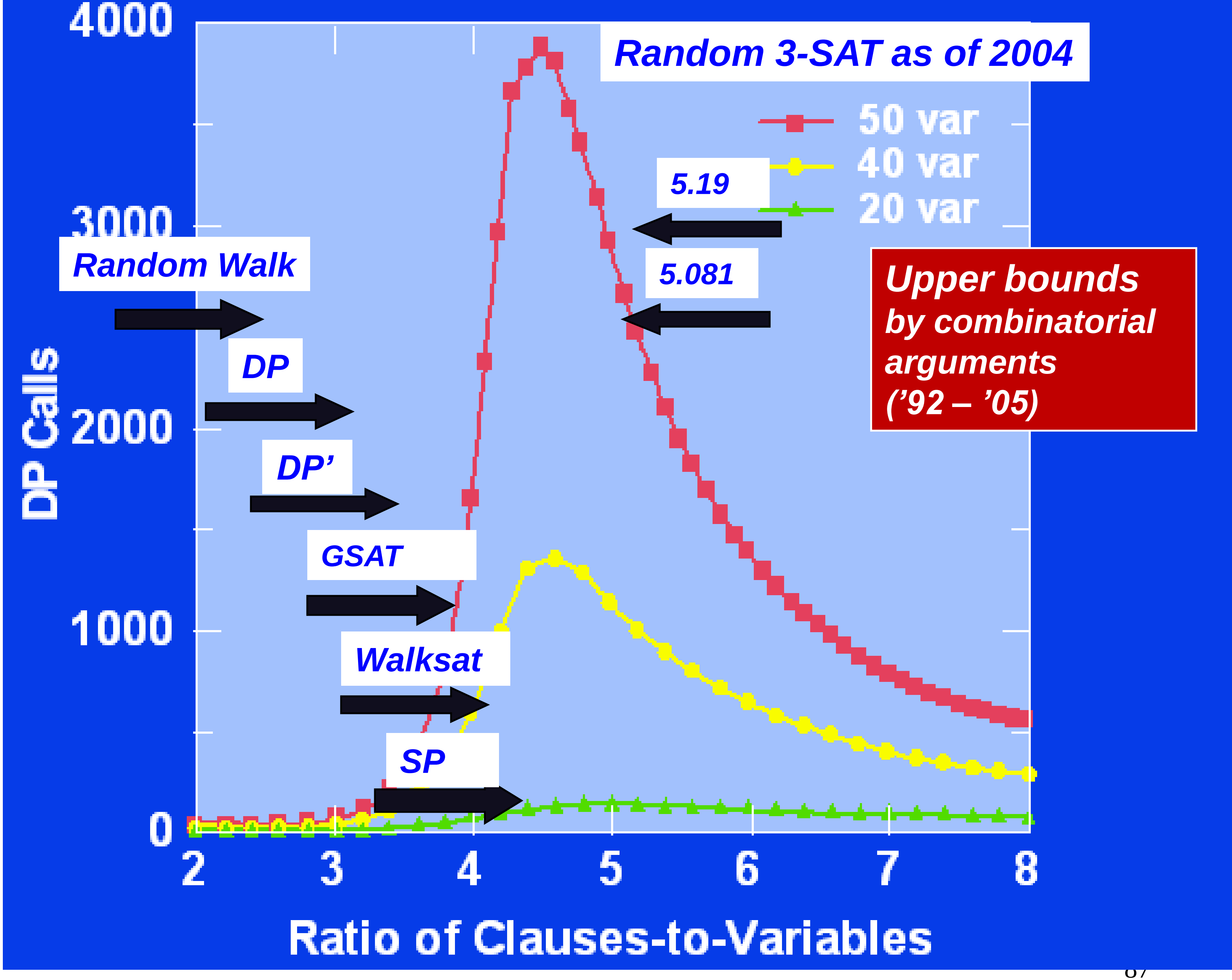

Random 3-SAT as of 2004
50 var
40 var
20 var
DP Calls
Ratio of Clauses-to-Variables
4000
3000
2000
1000
0
2
3
4
5
6
7
8
Random Walk
DP
DP’
GSAT
Walksat
SP

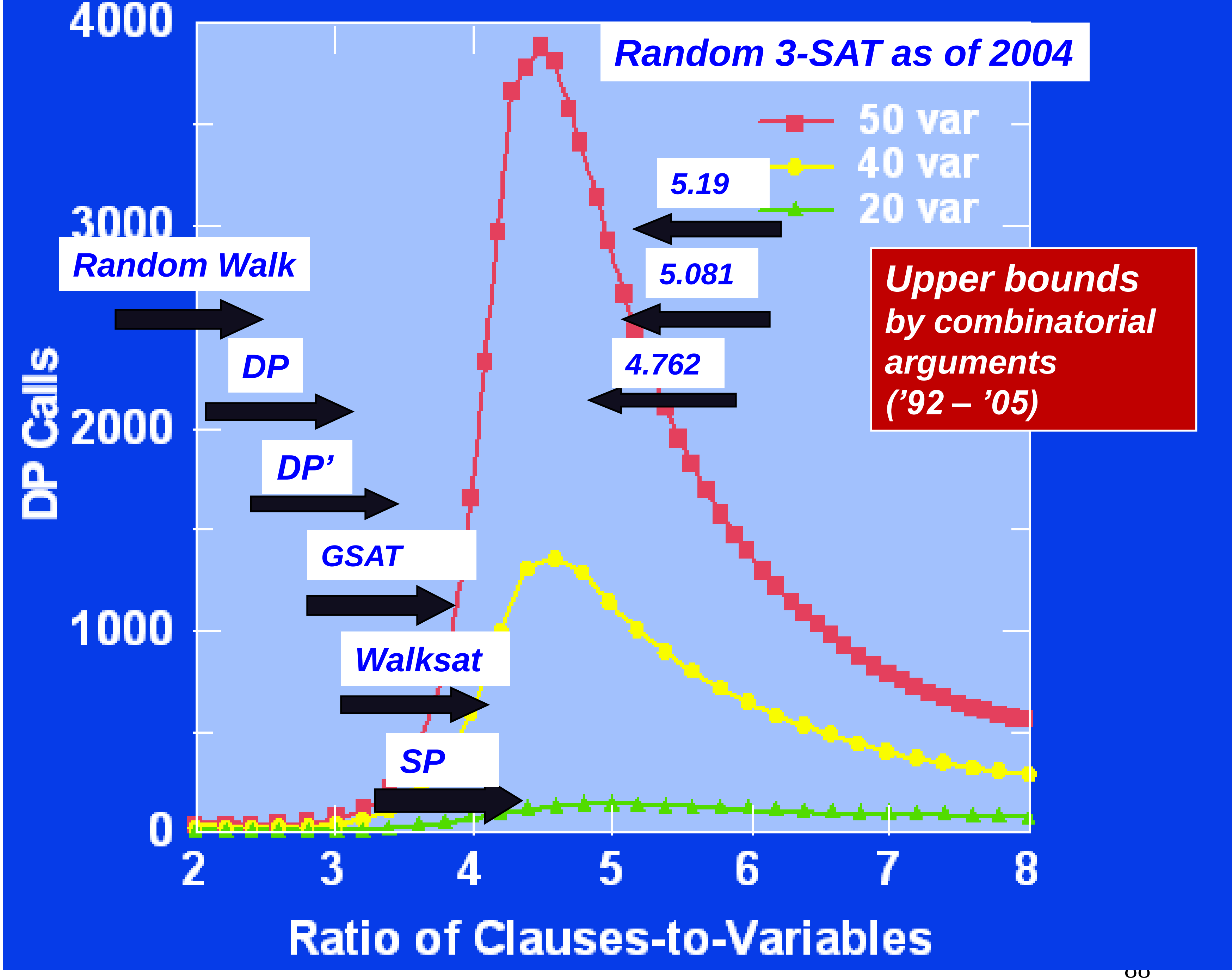

Random 3-SAT as of 2004
50 var
40 var
20 var
Upper bounds by combinatorial arguments ('92 – '05)
Random Walk
DP
DP'
GSAT
Walksat
SP
DP Calls
4000
3000
2000
1000
0
2
3
4
5
6
7
8
Ratio of Clauses-to-Variables



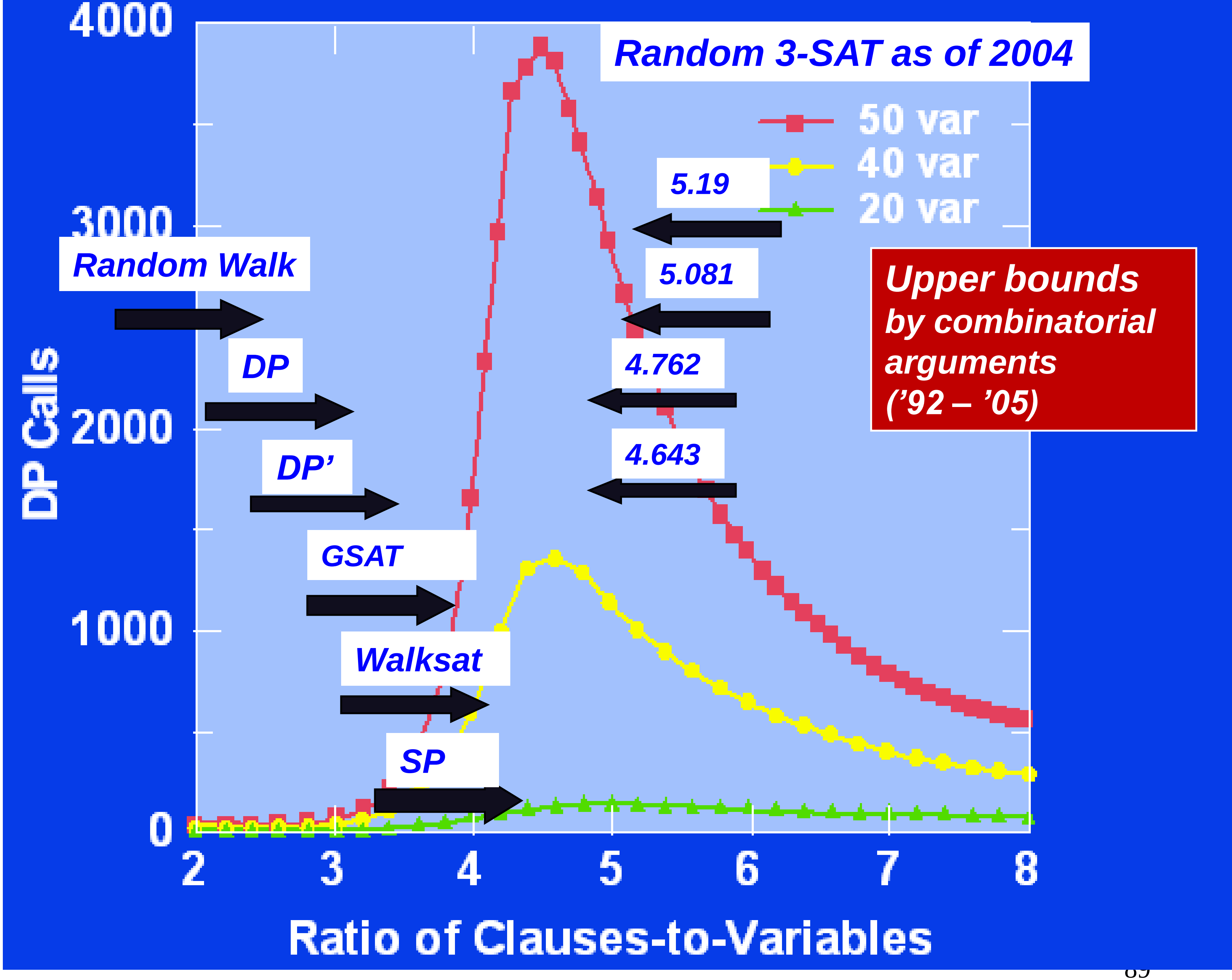
Random 3-SAT as of 2004
50 var
40 var
20 var
Upper bounds by combinatorial arguments ('92 – '05)
Random Walk
DP
DP'
GSAT
Walksat
SP
DP Calls
4000
3000
2000
1000
0
2
3
4
5
6
7
8
Ratio of Clauses-to-Variables

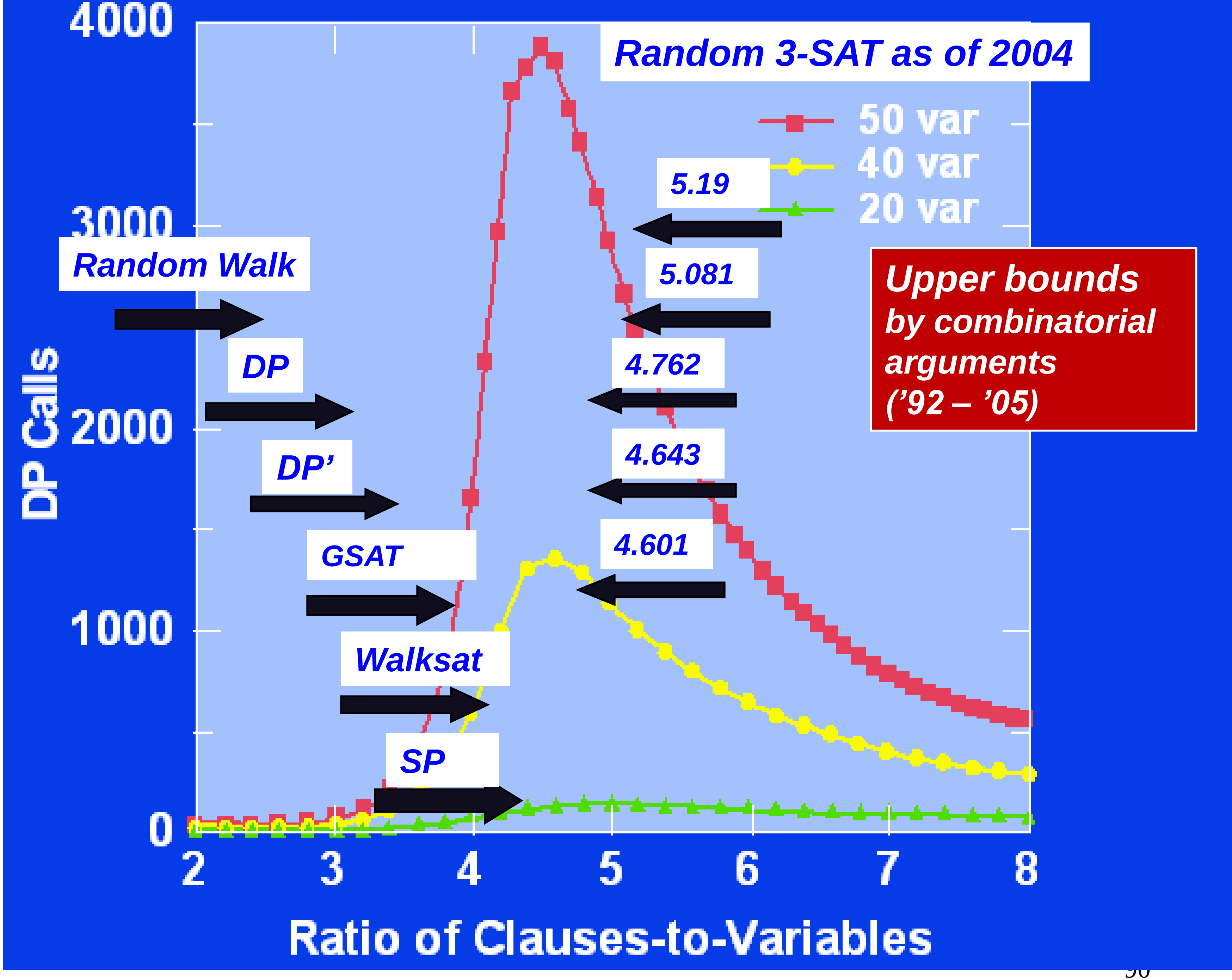

Random 3-SAT as of 2004
50 var
40 var
20 var
5.19
Upper bounds
by combinatorial
arguments
('92 – '05)
Random Walk
DP
DP'
GSAT
Walksat
SP
DP Calls
4000
3000
2000
1000
0
2
3
4
5
6
7
8
Ratio of Clauses-to-Variables


Random 3-SAT as of 2004
DP Calls
4000
3000
2000
1000
0
2
3
4
5
6
7
8
Ratio of Clauses-to-Variables
50 var
40 var
20 var
Random Walk
DP
DP'
GSAT
Walksat
SP
5.19
5.081
Upper bounds
by combinatorial
arguments
('92 – '05)

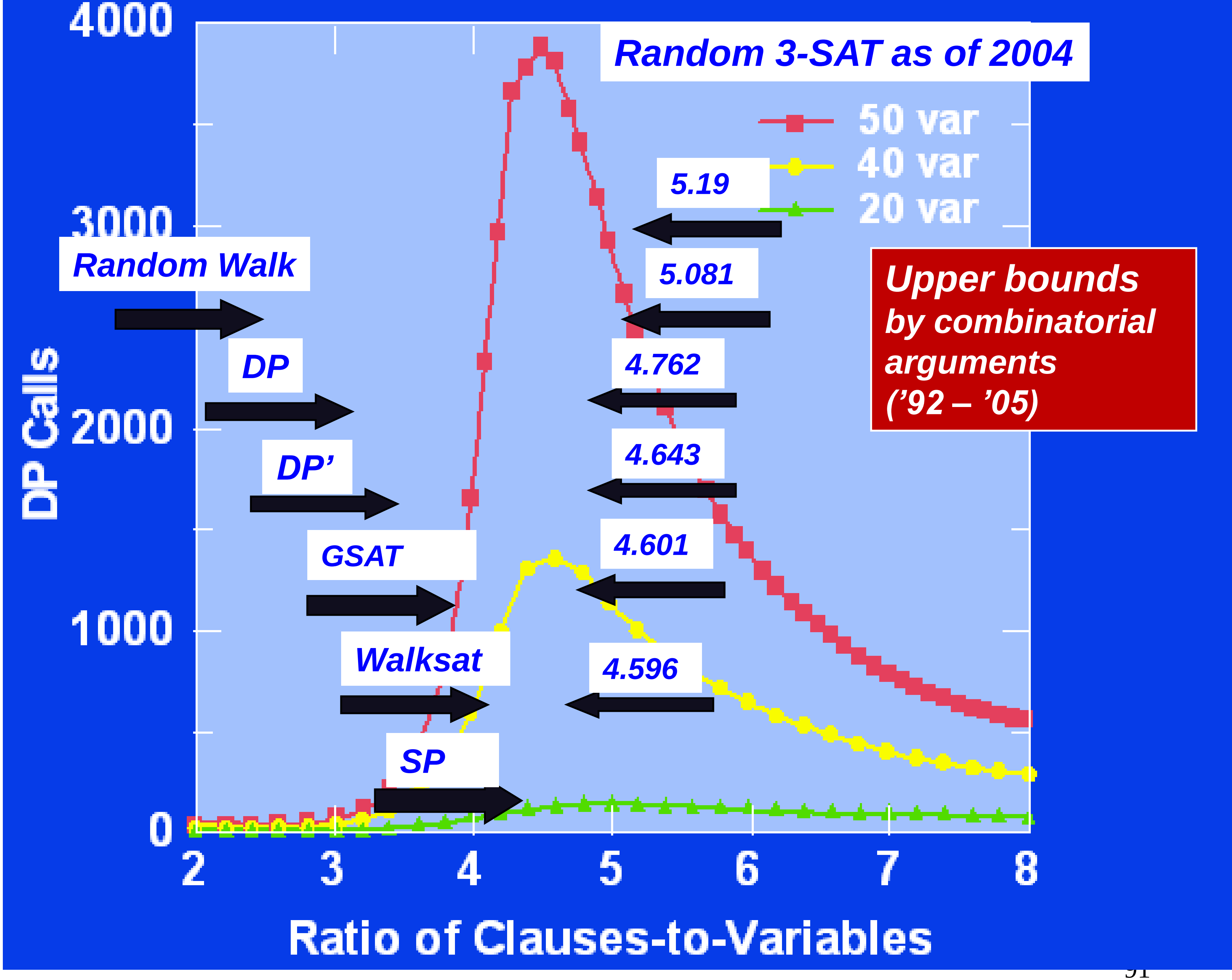


Random 3-SAT as of 2004
50 var
40 var
20 var
DP Calls
4000
3000
2000
1000
0
2
3
4
5
6
7
8
Ratio of Clauses-to-Variables
Random Walk
DP
DP'
GSAT
Walksat
SP
5.19
5.081
4.762
Upper bounds
by combinatorial
arguments
('92 – '05)

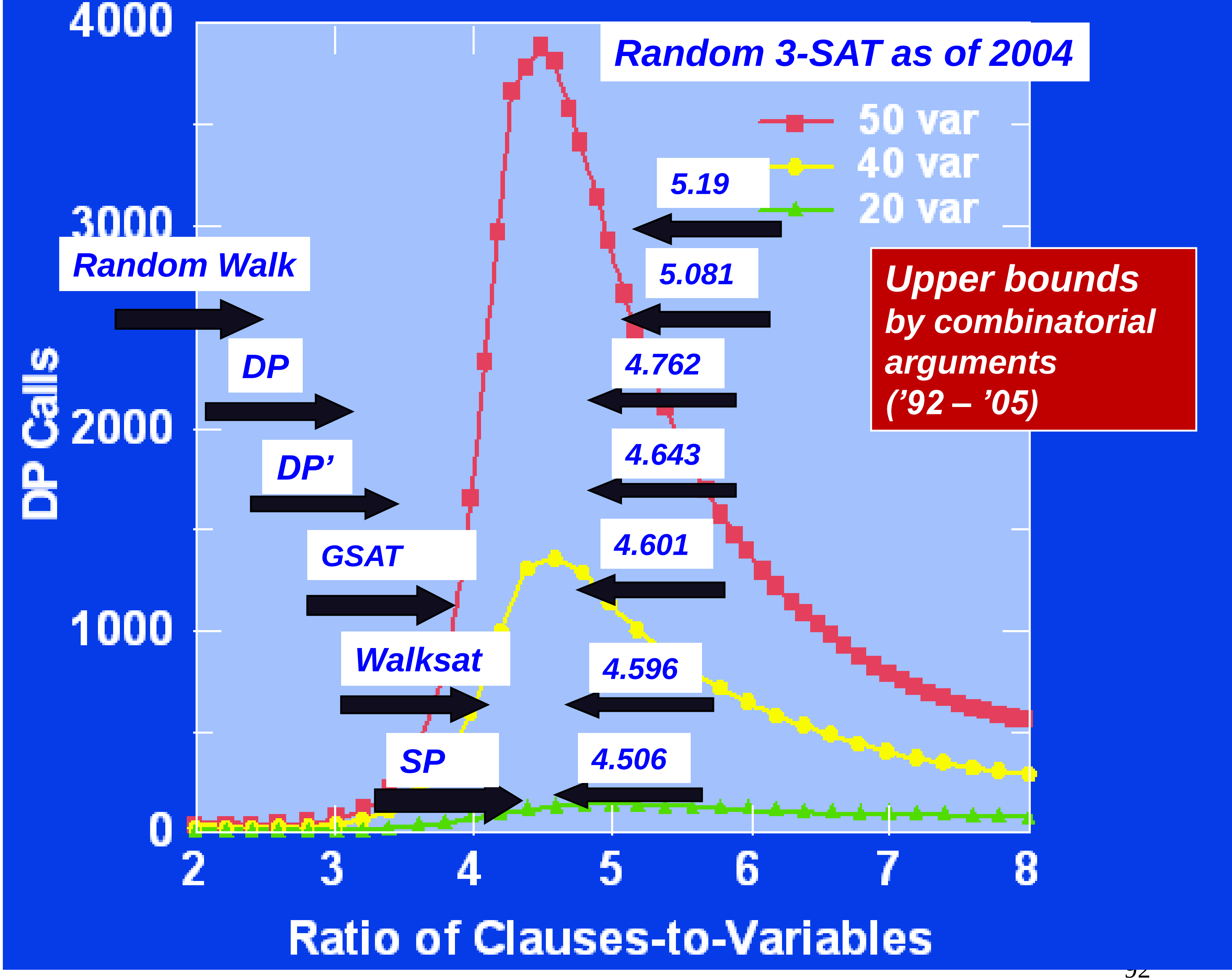

Random 3-SAT as of 2004
50 var
40 var
20 var
DP Calls
4000
3000
2000
1000
0
2
3
4
5
6
7
8
Ratio of Clauses-to-Variables
Random Walk
DP
DP'
GSAT
Walksat
SP
5.19
5.081
4.762
4.643
Upper bounds
by combinatorial
arguments
('92 – '05)



Random 3-SAT as of 2004
50 var
40 var
20 var
DP Calls
4000
3000
2000
1000
0
2
3
4
5
6
7
8
Ratio of Clauses-to-Variables
Random Walk
DP
DP'
GSAT
Walksat
SP
5.19
5.081
4.762
4.643
4.601
Upper bounds
by combinatorial
arguments
('92 – '05)



Random 3-SAT as of 2004
50 var
40 var
20 var
DP Calls
4000
3000
2000
1000
0
2
3
4
5
6
7
8
Ratio of Clauses-to-Variables
Random Walk
DP
DP'
GSAT
Walksat
SP
5.19
5.081
4.762
4.643
4.601
4.596
Upper bounds by combinatorial arguments ('92 – '05)

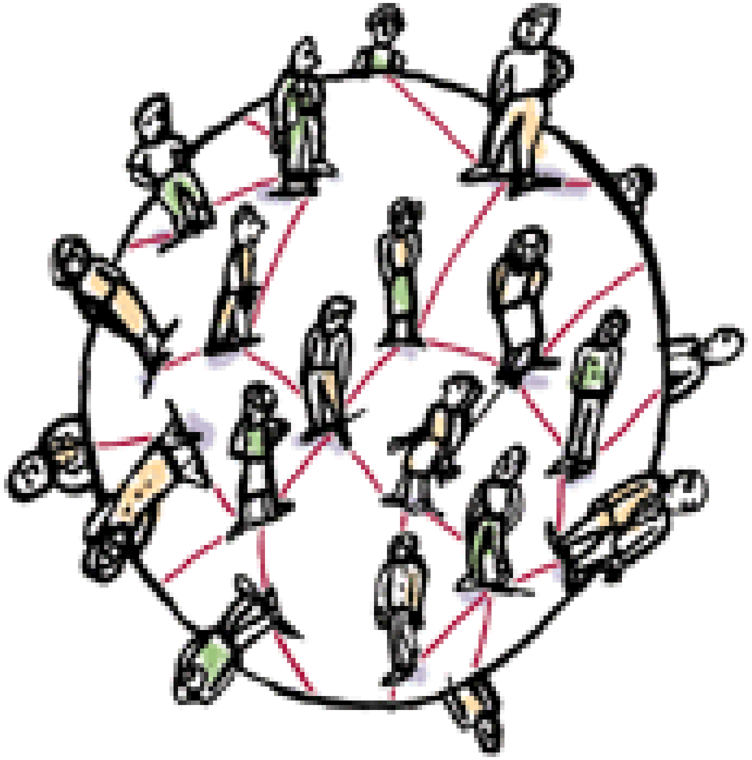


Random 3-SAT as of 2004
50 var
40 var
20 var
Upper bounds by combinatorial arguments ('92 – '05)
Random Walk
DP
DP'
GSAT
Walksat
SP
5.19
5.081
4.762
4.643
4.601
4.596
4.506
DP Calls
4000
3000
2000
1000
0
2
3
4
5
6
7
8
Ratio of Clauses-to-Variables

Real versus Random

- Real graphs tend to be sparse
  - dense random graphs contains 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

Real World DPLL

Observation: Complete backtrack style search SAT solvers (e.g. DPLL) display a remarkably wide range of run times.

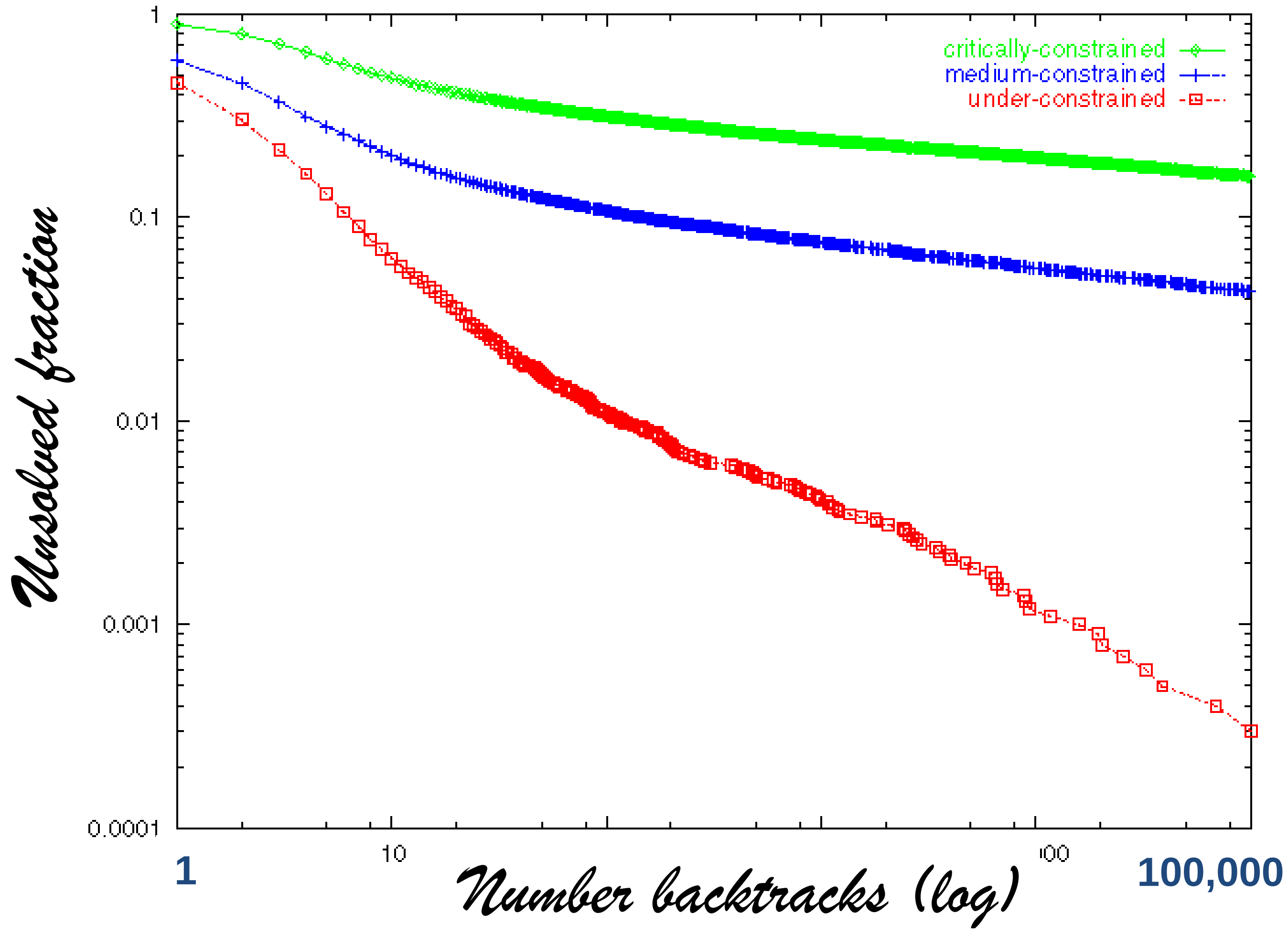
Even when repeatedly solving the same problem instance; variable branching is choice randomized.

Run time distributions are often “heavy-tailed”.

Orders of magnitude difference in run time on different runs.

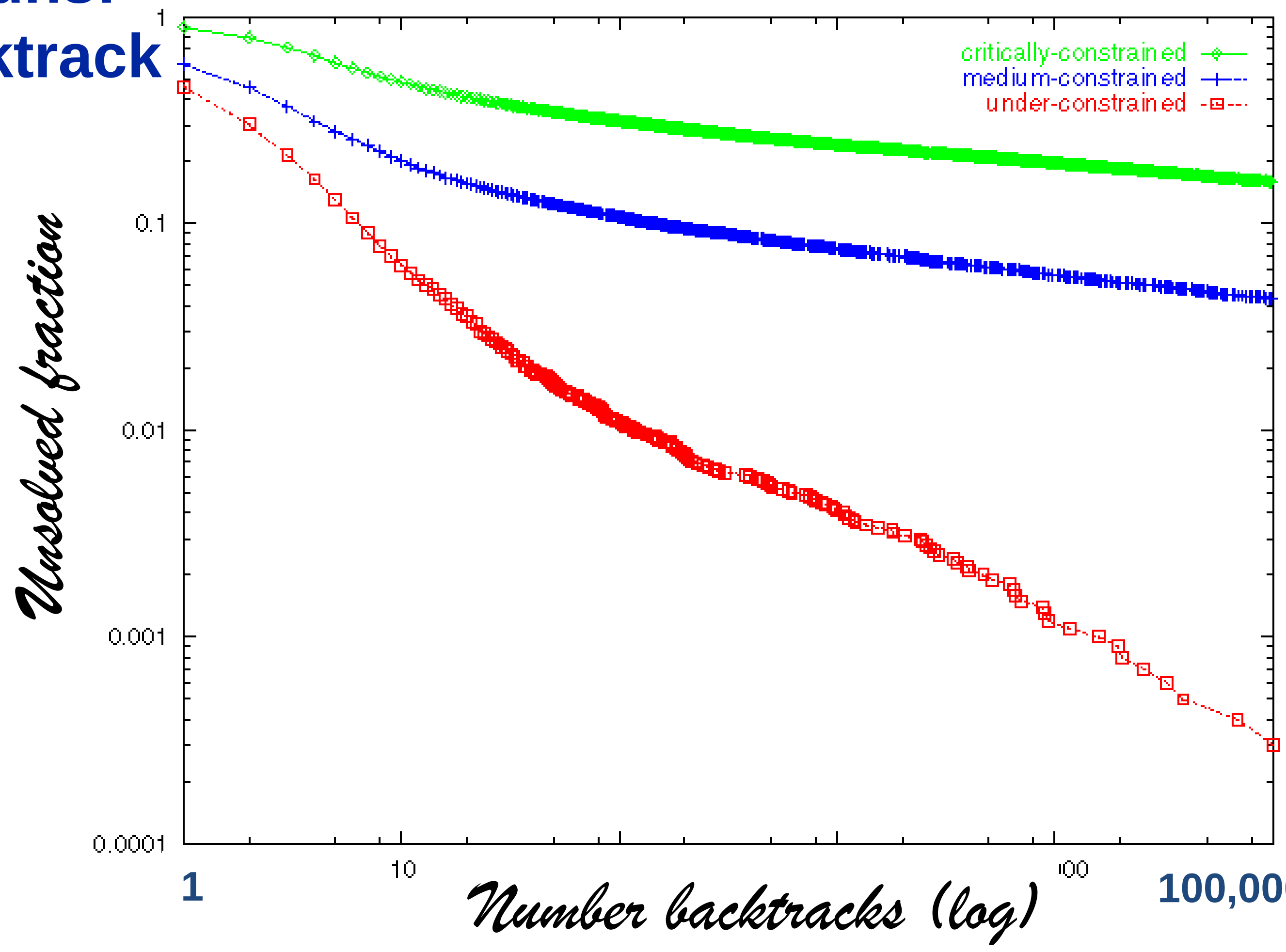
(Gomes et al. 1998; 2000)

Heavy Tails on Structured Problems



Heavy Tails on Structured Problems

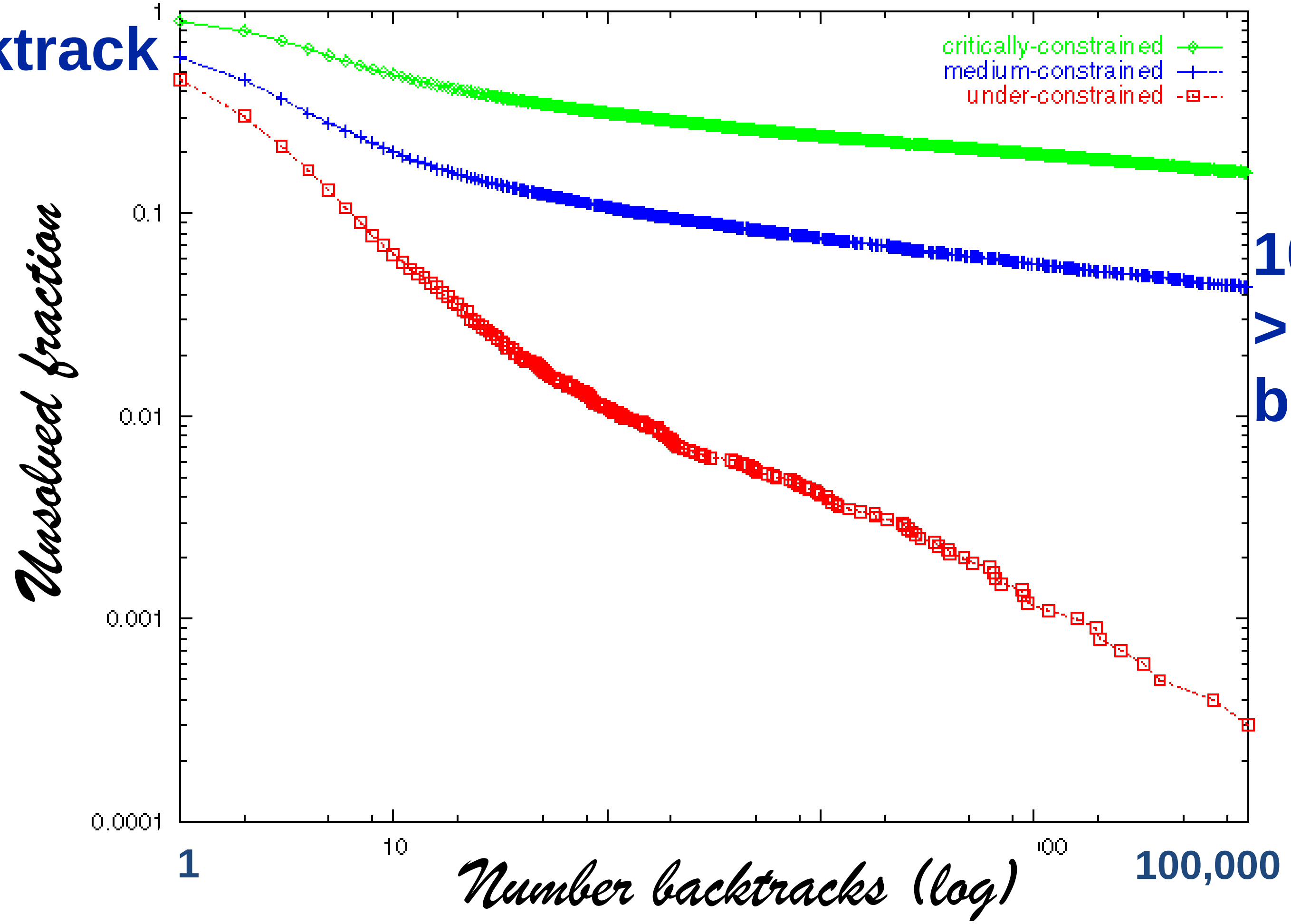
50% runs:
1 backtrack



1 **100,000**

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

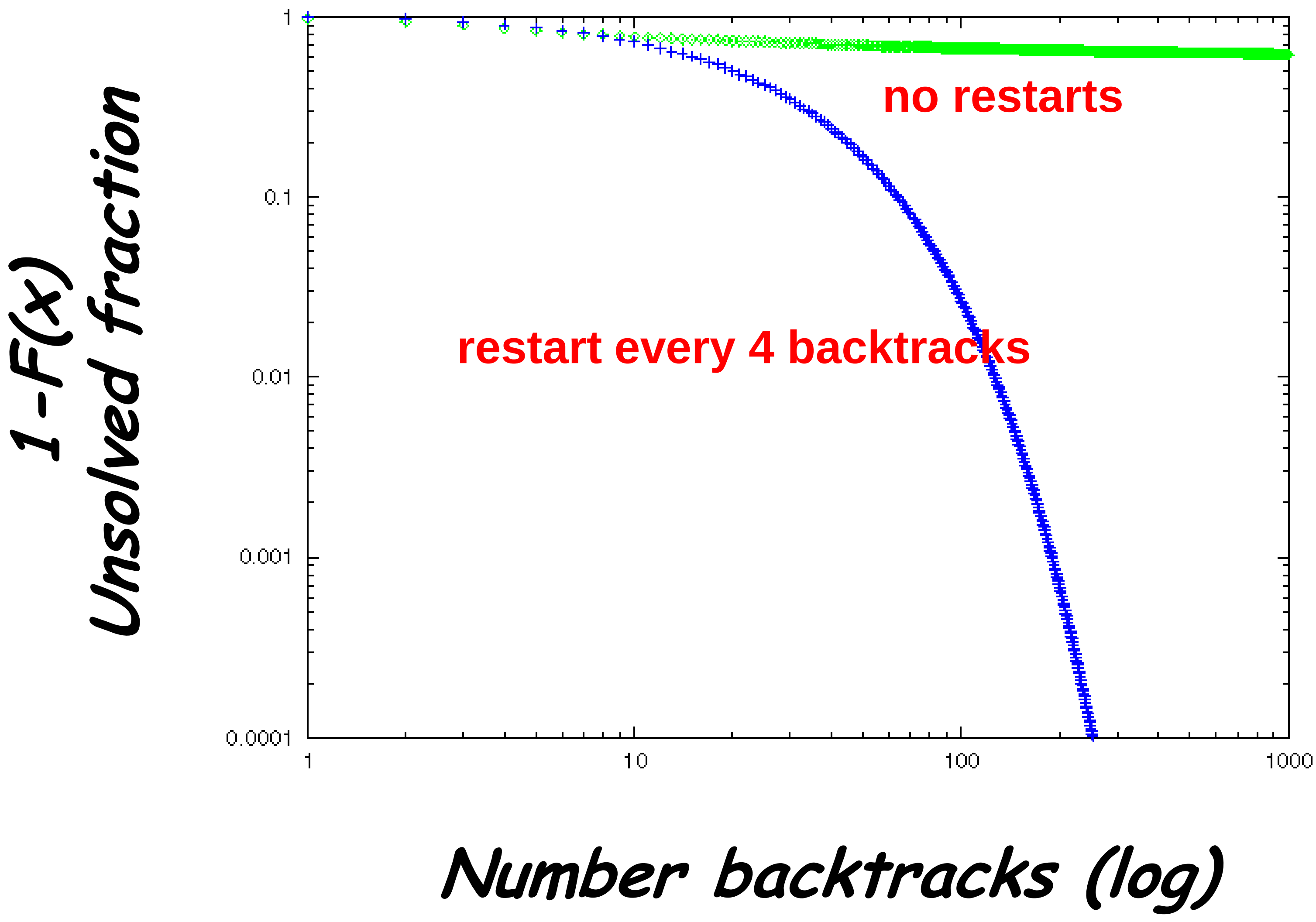
Cutoff and restart search after a fixed number of backtracks

Provably Eliminates heavy tails

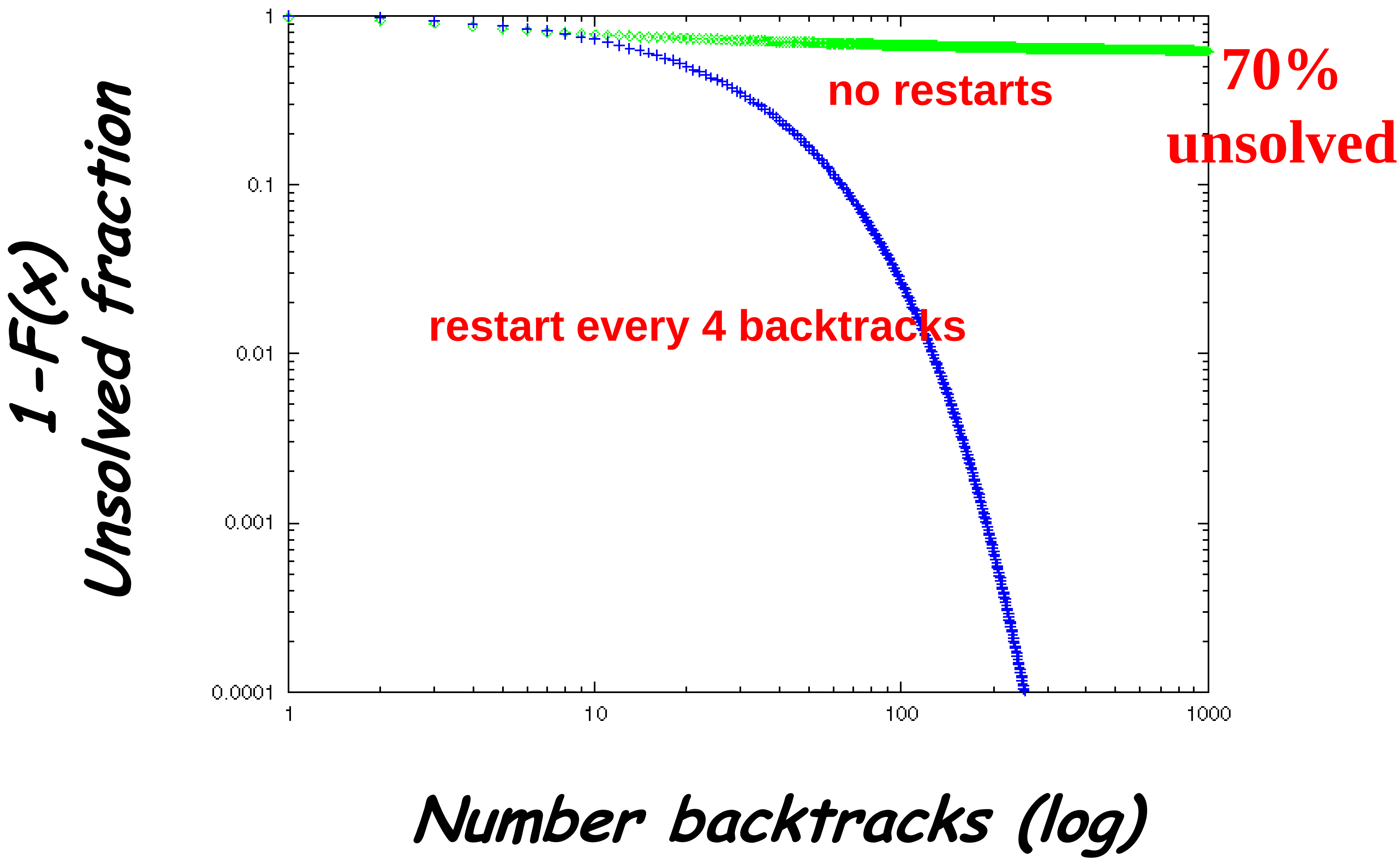
In practice: rapid restarts with low cutoff can dramatically improve performance (Gomes et al. 1998, 1999)

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

Restarts

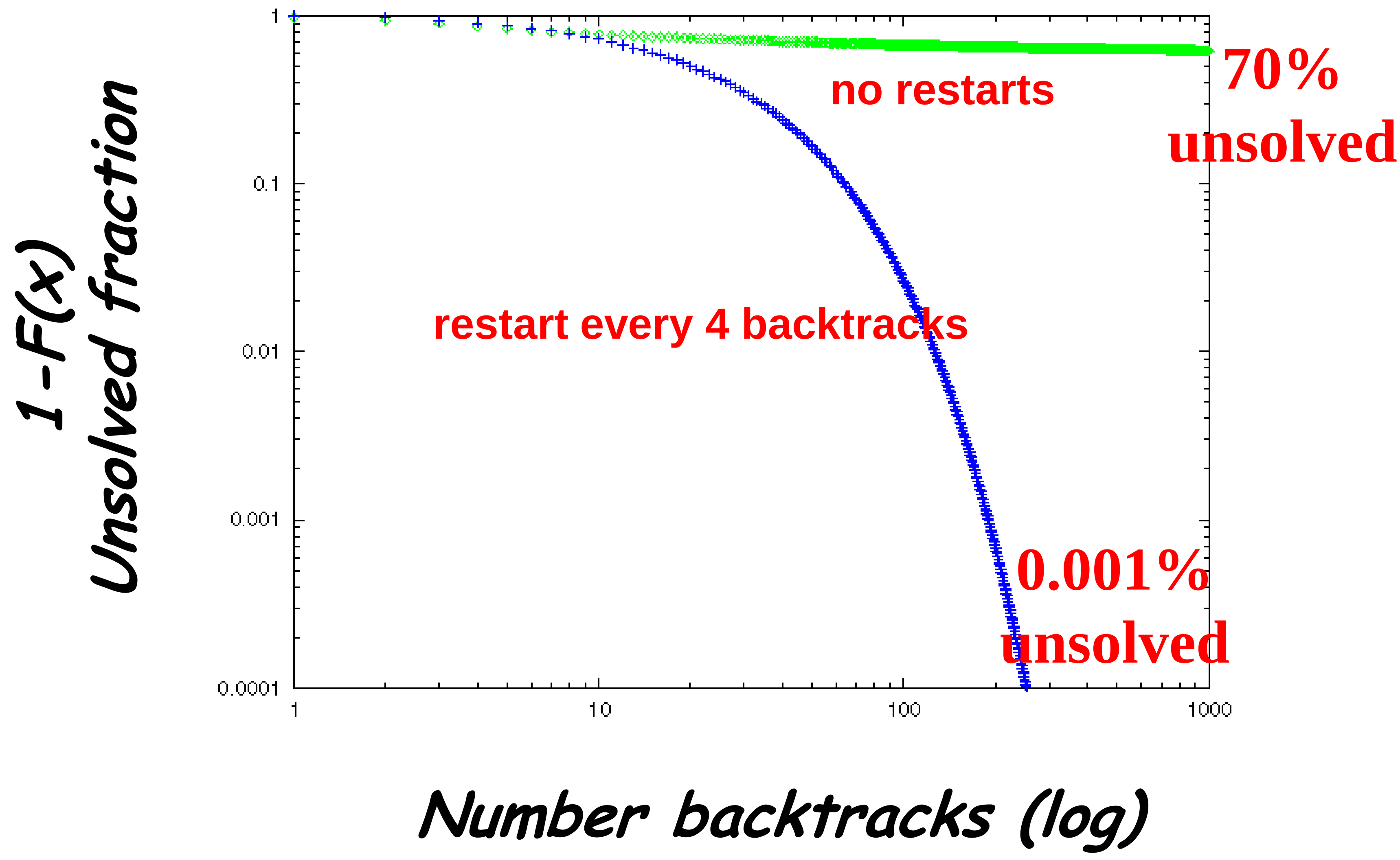

1-F(x)
Unsolved fraction
1
0.1
0.01
0.001
0.0001
no restarts
restart every 4 backtracks
1
10
100
1000
Number backtracks (log)

Restarts



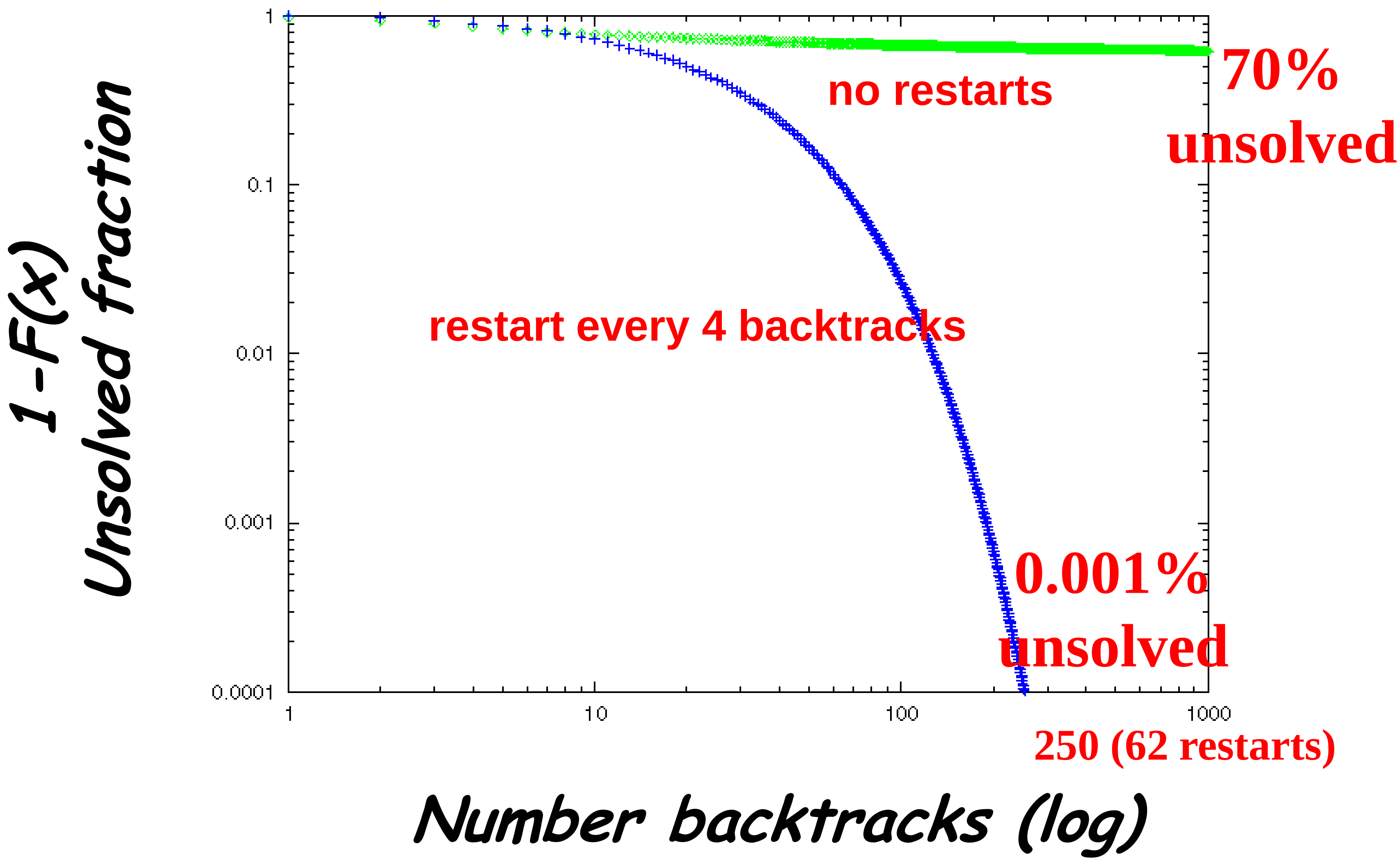
1
0.1
0.01
0.001
0.0001
1
10
100
1000
70%
unsolved
no restarts
restart every 4 backtracks
1-F(x)
Unsolved fraction
Number backtracks (log)

Restarts

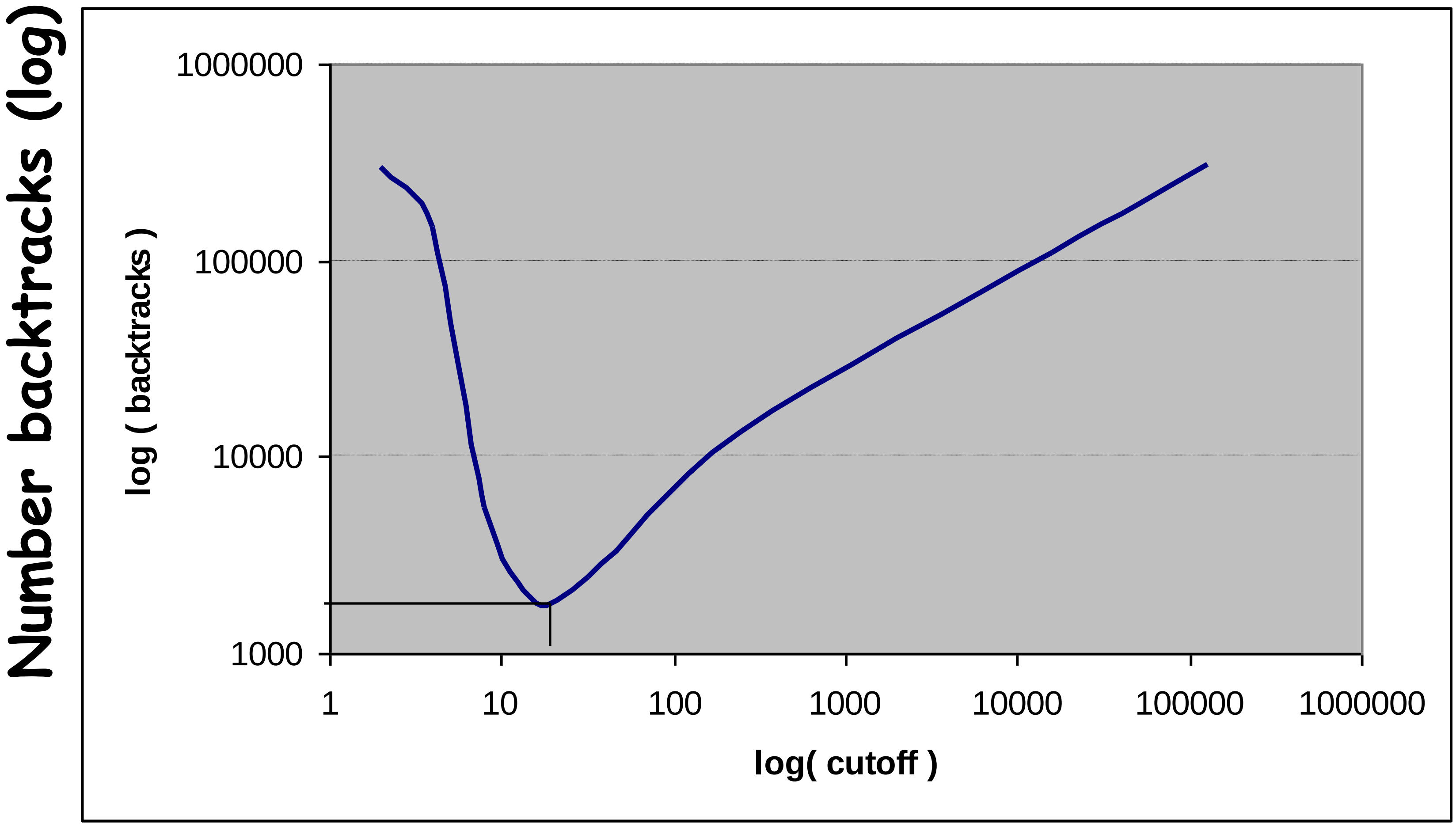


1-F(x)
Unsolved fraction
1
0.1
0.01
0.001
0.0001
1
10
100
1000
no restarts
70%
unsolved
restart every 4 backtracks
0.001%
unsolved
Number backtracks (log)

Restarts

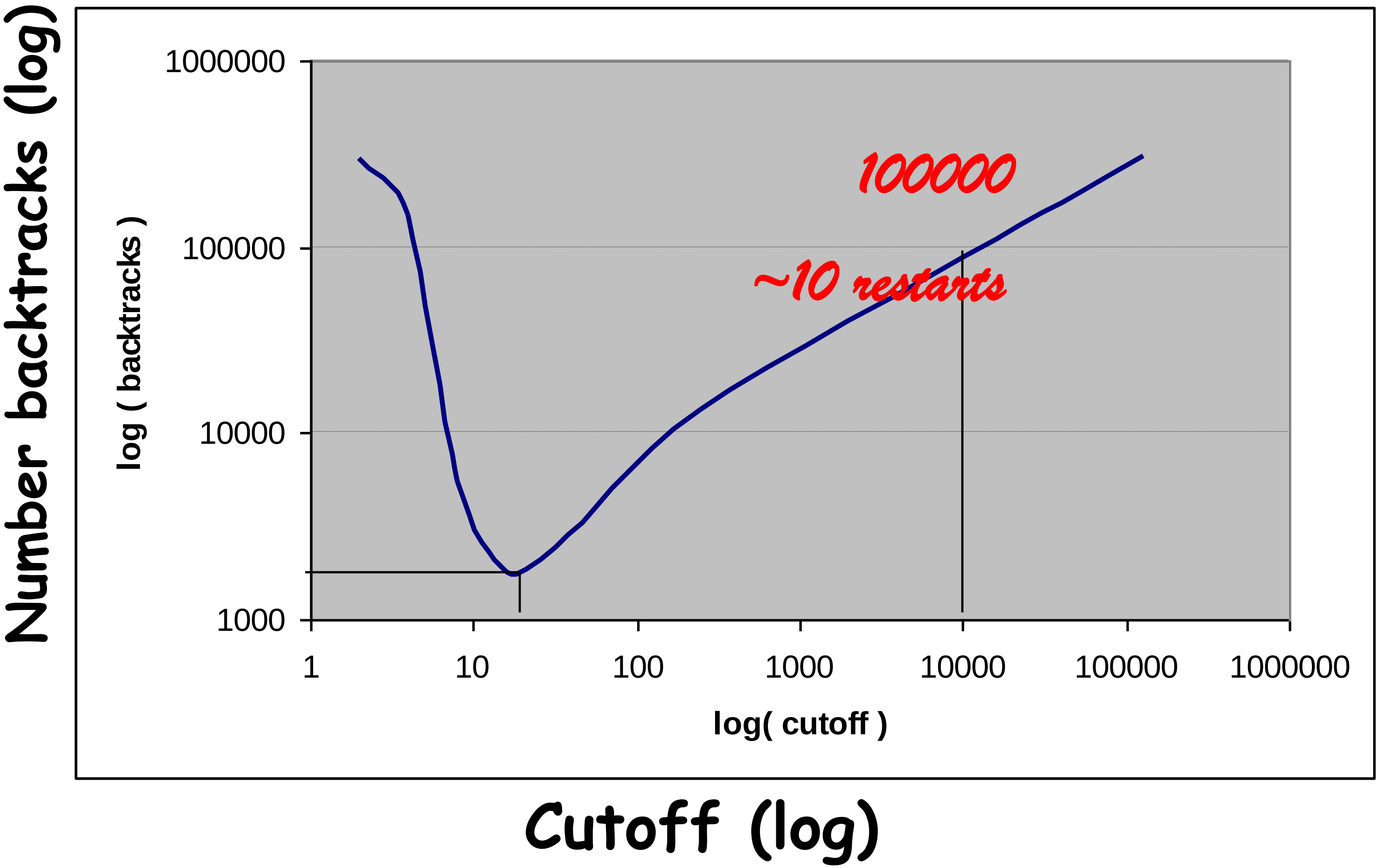

1
0.1
0.01
0.001
0.0001
1
10
100
1000
1-F(x)
Unsolved fraction
no restarts
70%
unsolved
restart every 4 backtracks
0.001%
unsolved
250 (62 restarts)
Number backtracks (log)

Example of Rapid Restart Speedup

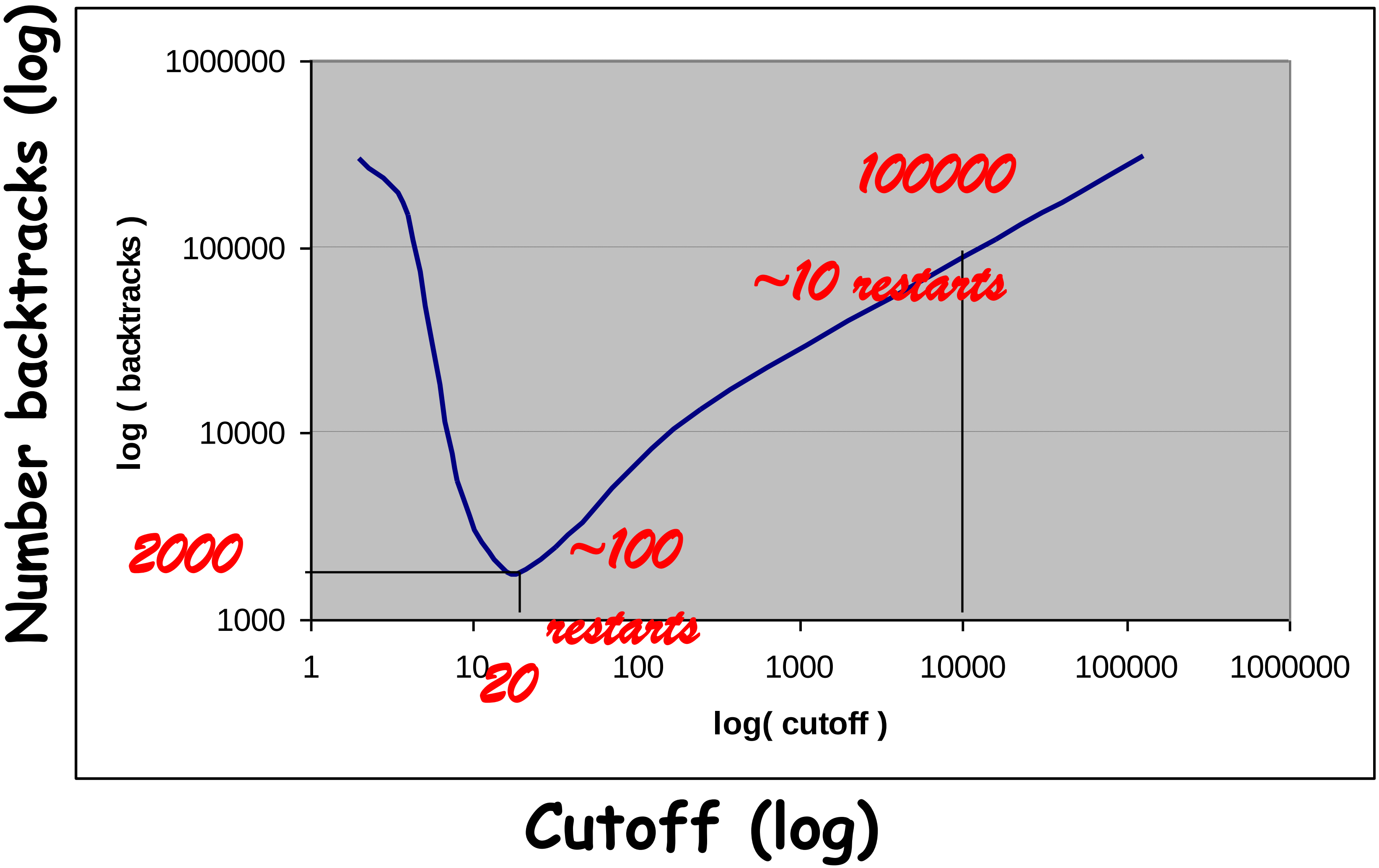


Cutoff (log)

Example of Rapid Restart Speedup



Example of Rapid Restart Speedup



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?

Backdoors: intuitions

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 $O((n-c)\ k^{(c+2)})$

- 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;***

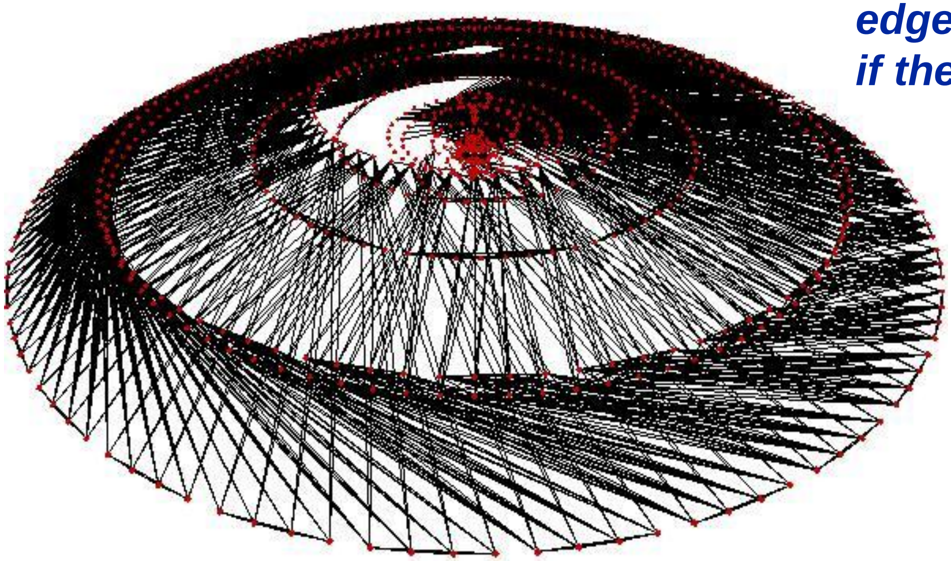
Backdoors can be surprisingly small

instance	# vars	# clauses	backdoor	fract.
logistics.d	6783	437431	12	0.0018
3bitadd_32	8704	32316	53	0.0061
pipe_01	7736	26087	23	0.0030
qg_30_1	1235	8523	14	0.0113
qg_35_1	1597	10658	15	0.0094

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)***

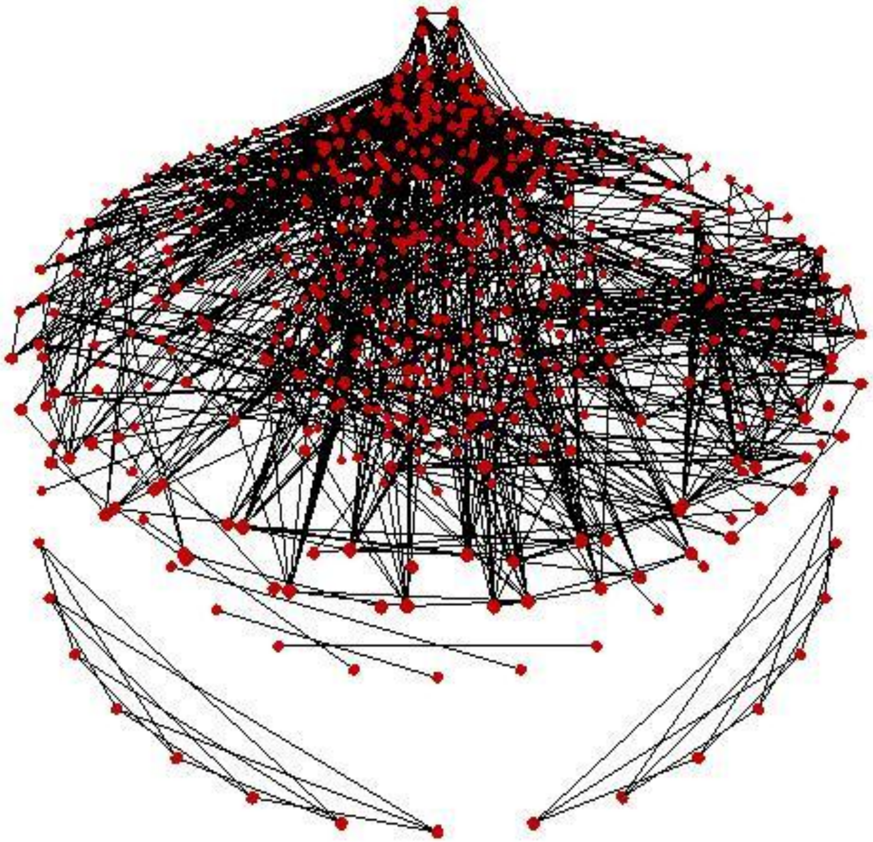
Backdoors capture critical problem resources (bottlenecks).

Backdoors --- “seeing is believing”

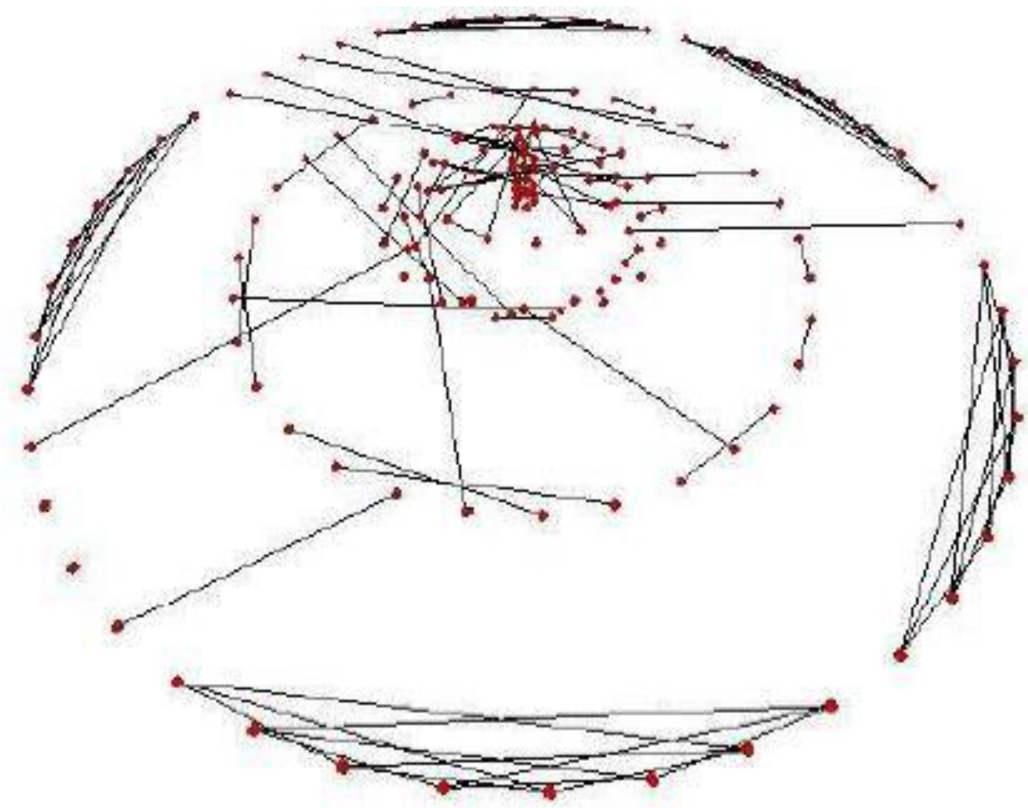


Constraint graph of reasoning problem. One node per variable: edge between two variables if they share a constraint.

Logistics_b.cnf planning formula.
843 vars, 7,301 clauses, approx min backdoor 16
(backdoor set = reasoning shortcut)

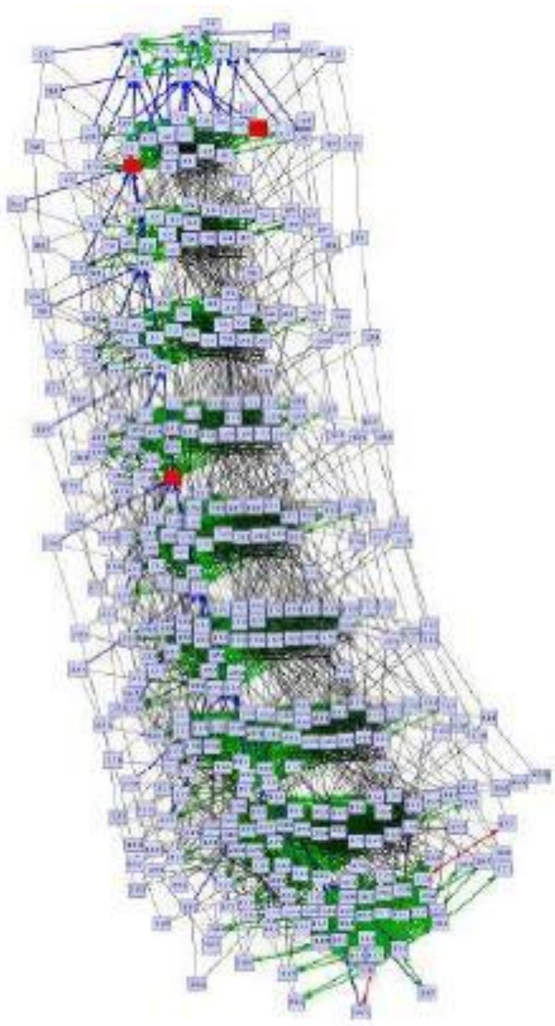


Logistics.b.cnf after setting 5 backdoor vars.

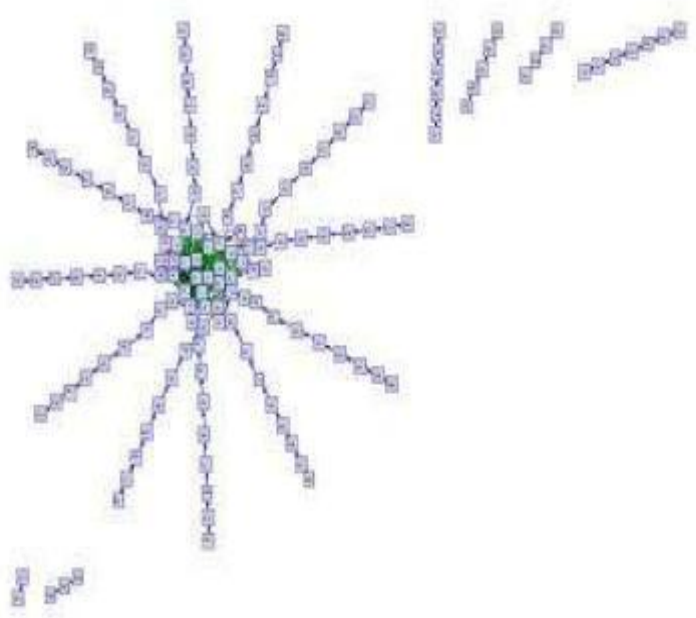


After setting just 12 (out of 800+) backdoor vars – problem almost solved.

Another example

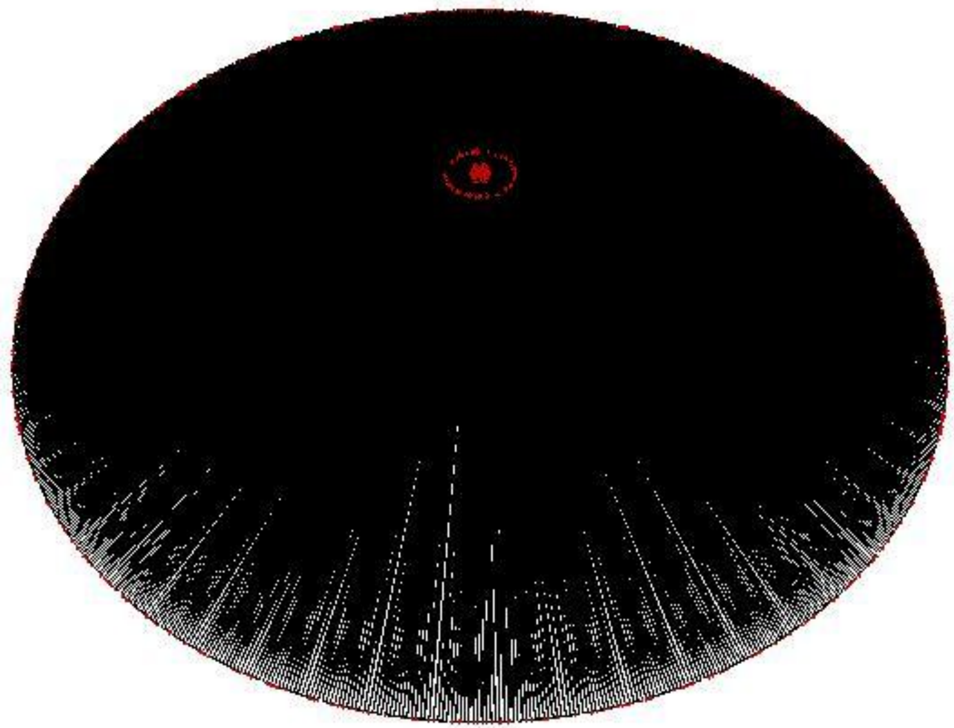


MAP-6-7.cnf infeasible planning instances. Strong backdoor of size 3.
392 vars, 2,578 clauses.

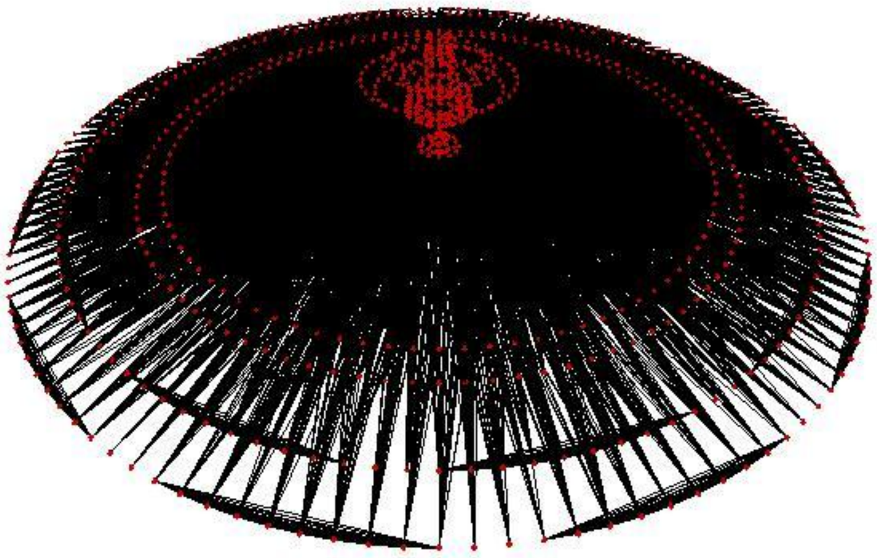


After setting 2 (out of 392) backdoor vars. ---
reducing problem complexity in just a few steps!

Last example.

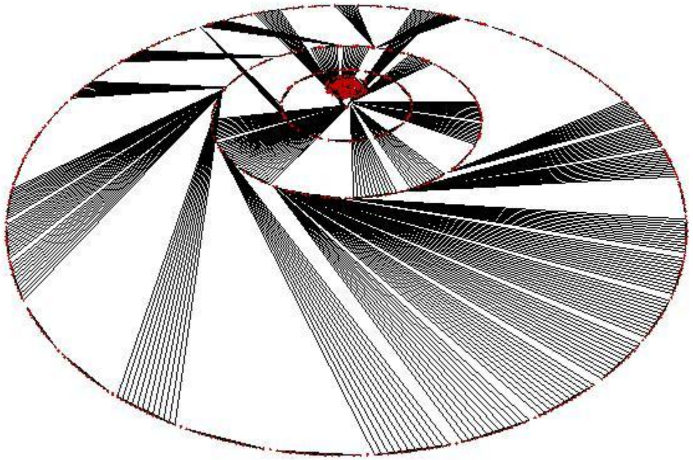


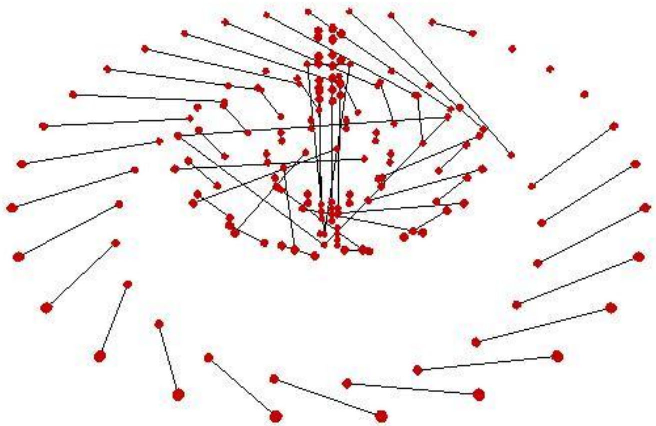
Inductive inference problem --- ii16a1.cnf. 1650 vars, 19,368 clauses.
Backdoor size 40.



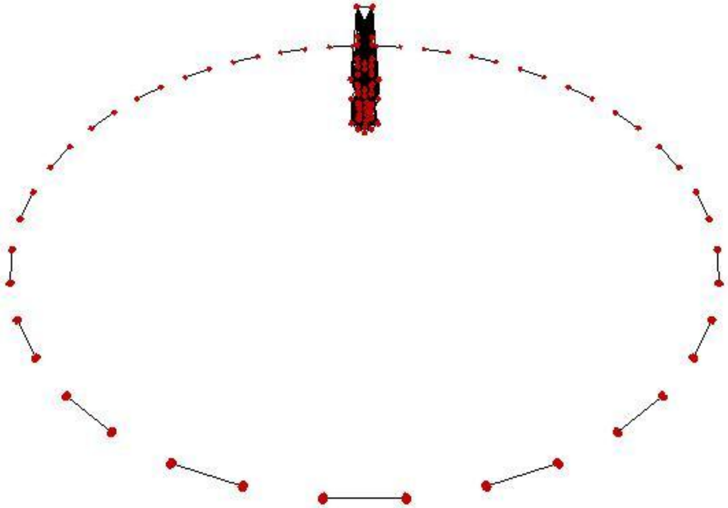
After setting 6 backdoor vars.

Some other intermediate stages:

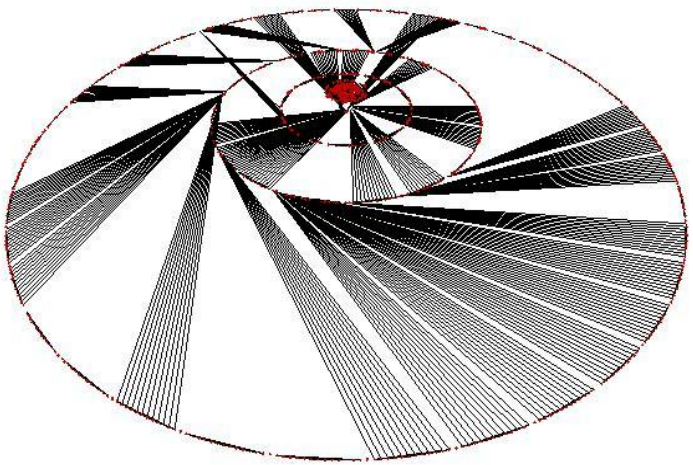


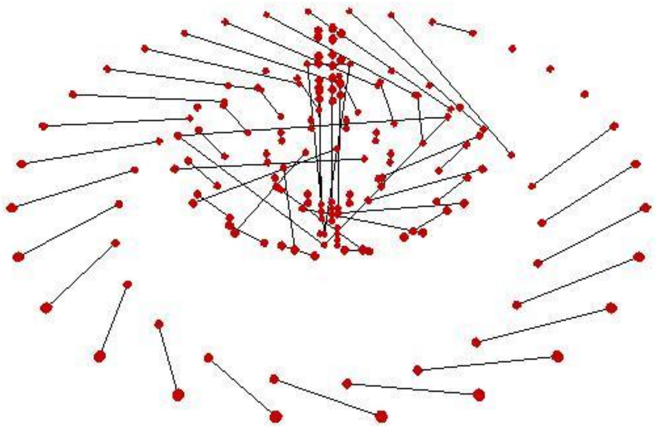


After setting 38 (out of 1600+) backdoor vars:

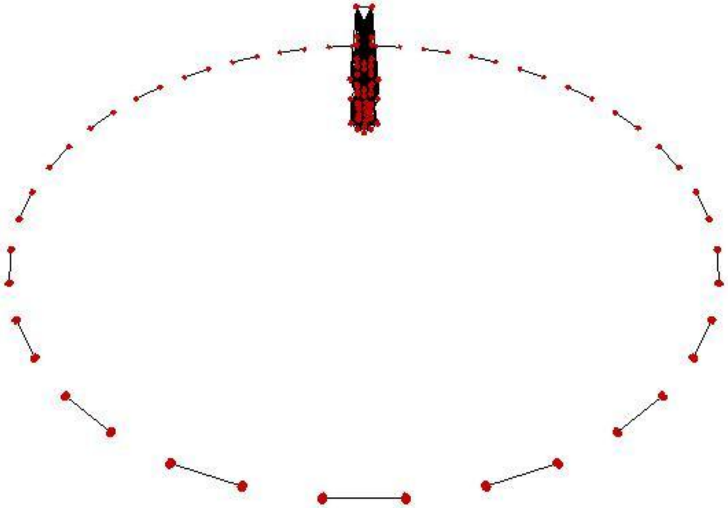


Some other intermediate stages:





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

$B(n)$	deterministic	randomized	heuristic
n/k	small $exp(n)$	smaller $exp(n)$	tiny $exp(n)$
$O(\log n)$	$\left(\frac{n}{\sqrt{\log n}}\right)^{O(\log n)}$	$\left(\frac{n}{\log n}\right)^{O(\log n)}$	$poly(n)$
$O(1)$	$poly(n)$	$poly(n)$	$poly(n)$

n = num. vars.
k is a constant

Current solvers

(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**: Satifisiability Modulo Theories (e.g., linear arithmetic, bit-vector operations, uninterpreted functions)