Advanced Satisfiability

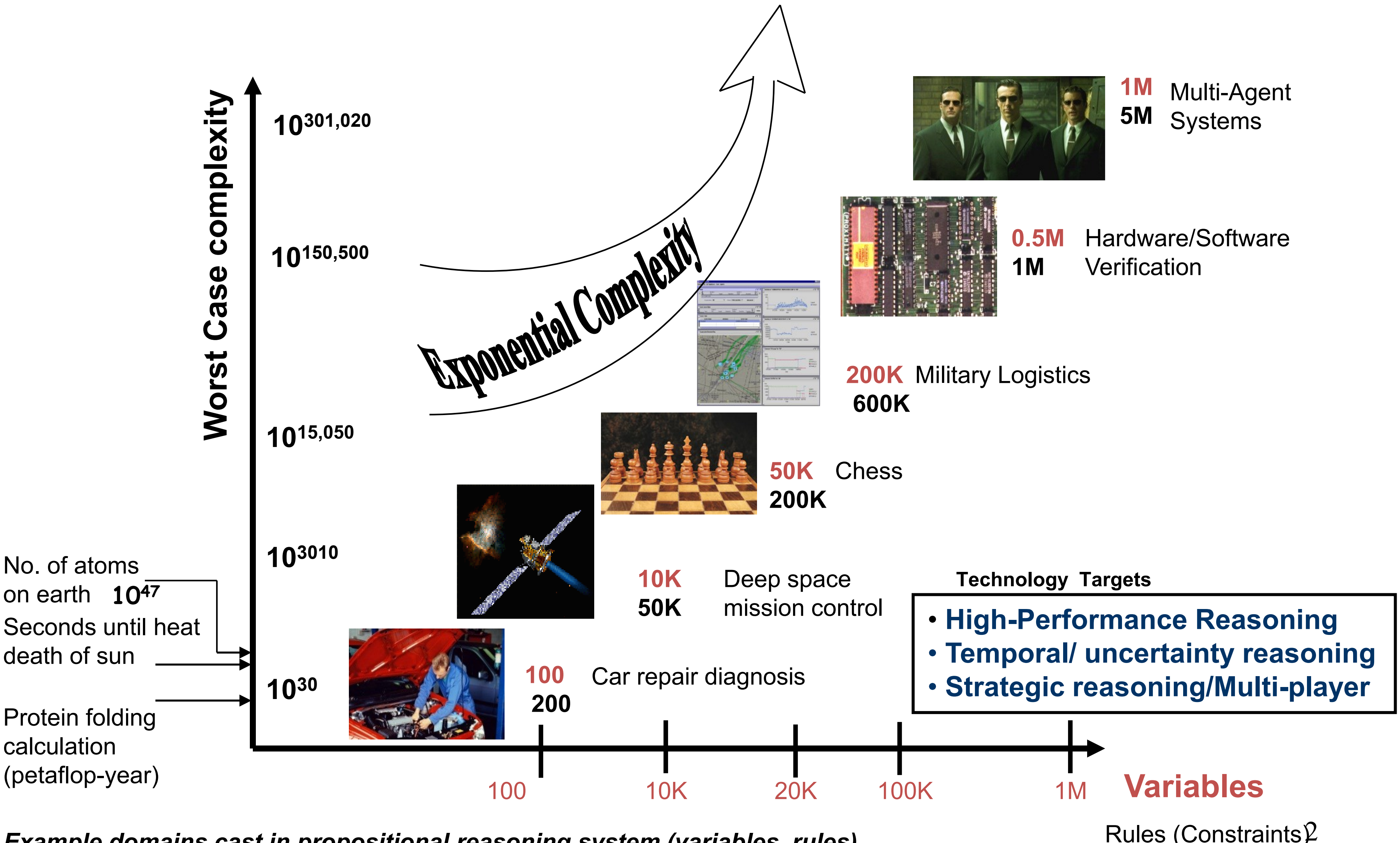
Mausam

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

Real-World Reasoning

Tackling inherent computational complexity

DARPA Research Program



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

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

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."]

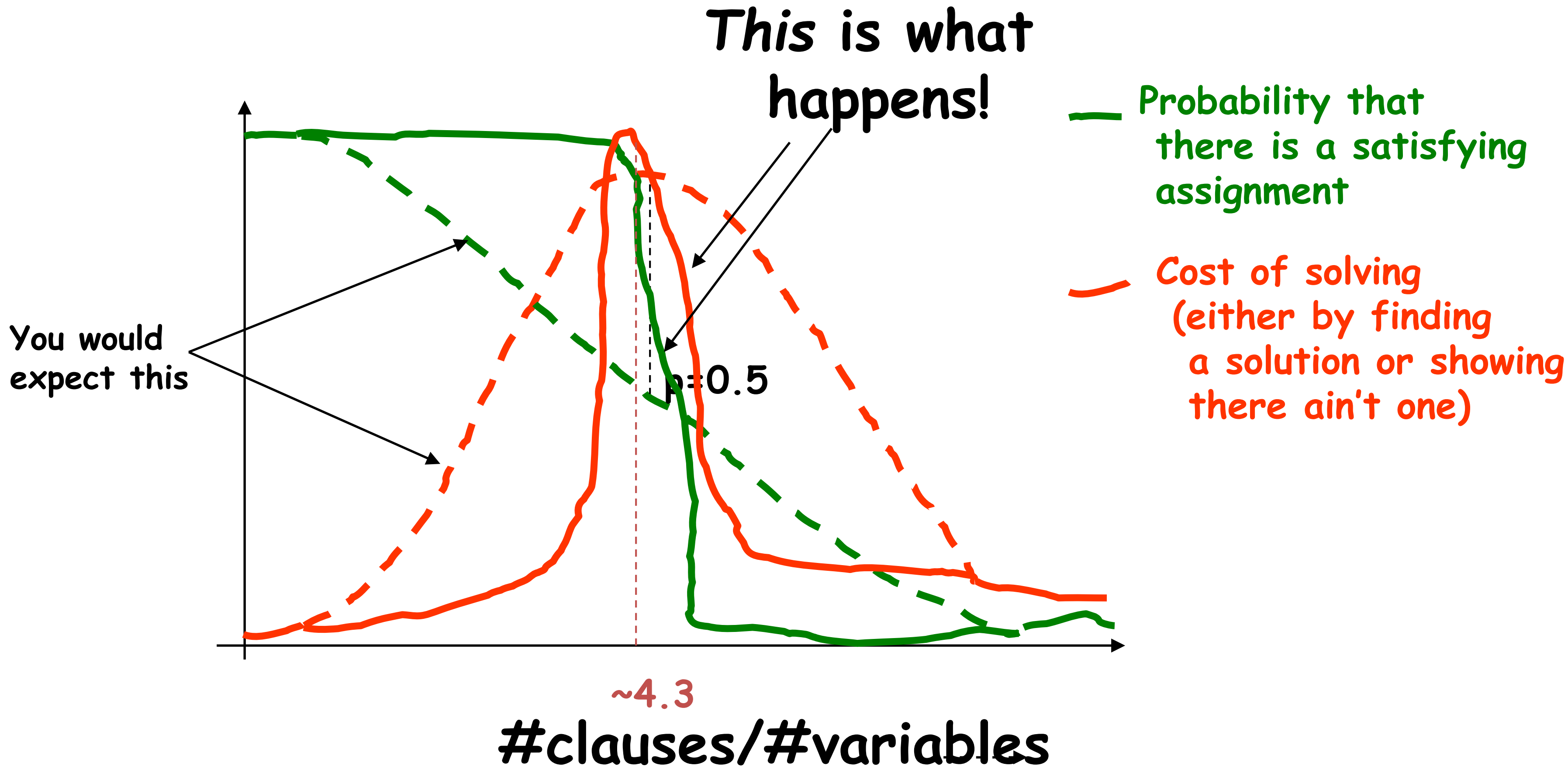
- Is this just Moore's Law? It helped, but not much…
- – 2x faster computers does *not* mean can solve 2x larger instances
- – search difficulty does not scale linearly with problem size!
- **Tools**: 50+ competitive SAT solvers available

Forces Driving Faster, Better SAT Solvers

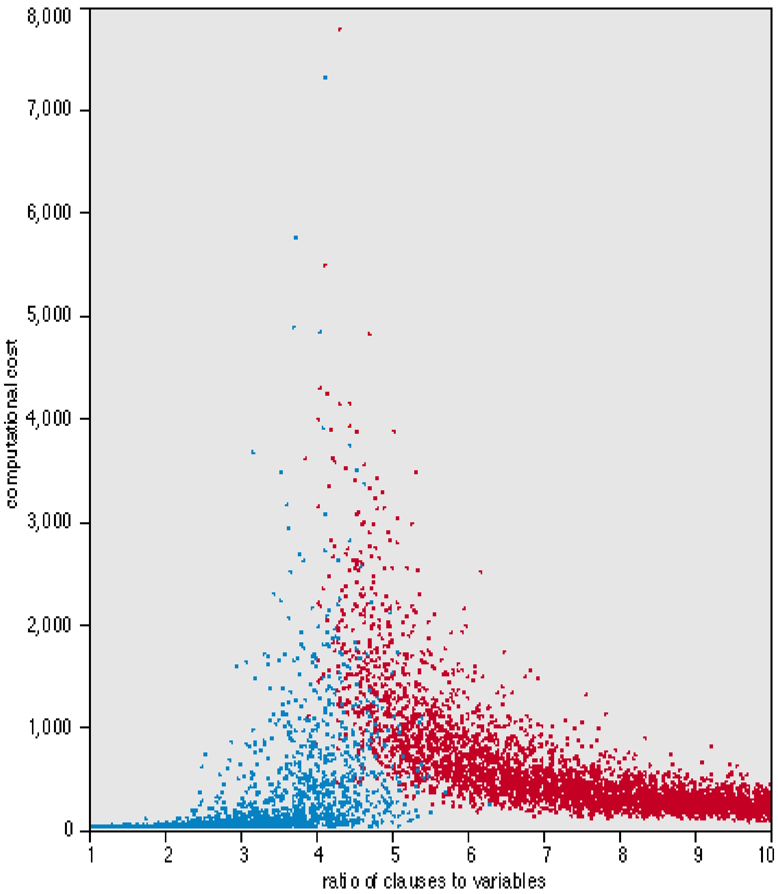
- **From academically interesting to practically relevant "Real" benchmarks**, with real interest in solving them

- 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 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*

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



Random 3-SAT

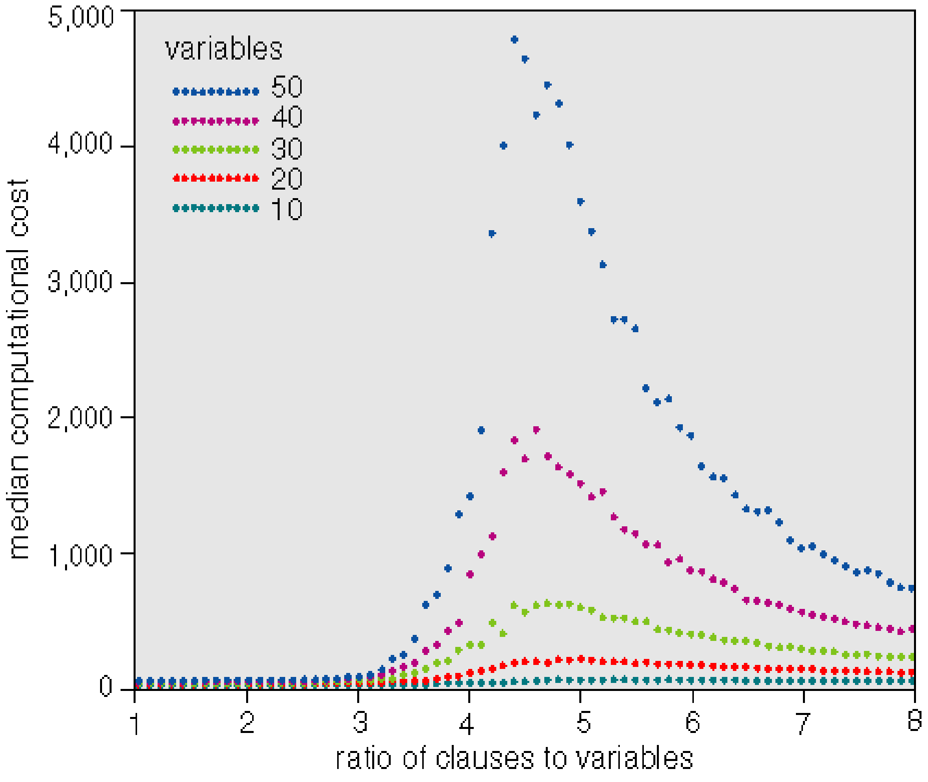


- Random 3-SAT
  - sample uniformly from space of all possible 3-clauses
  - *n* variables, *l* clauses
- Which are the hard instances?
  - around $l/n = 4.3$

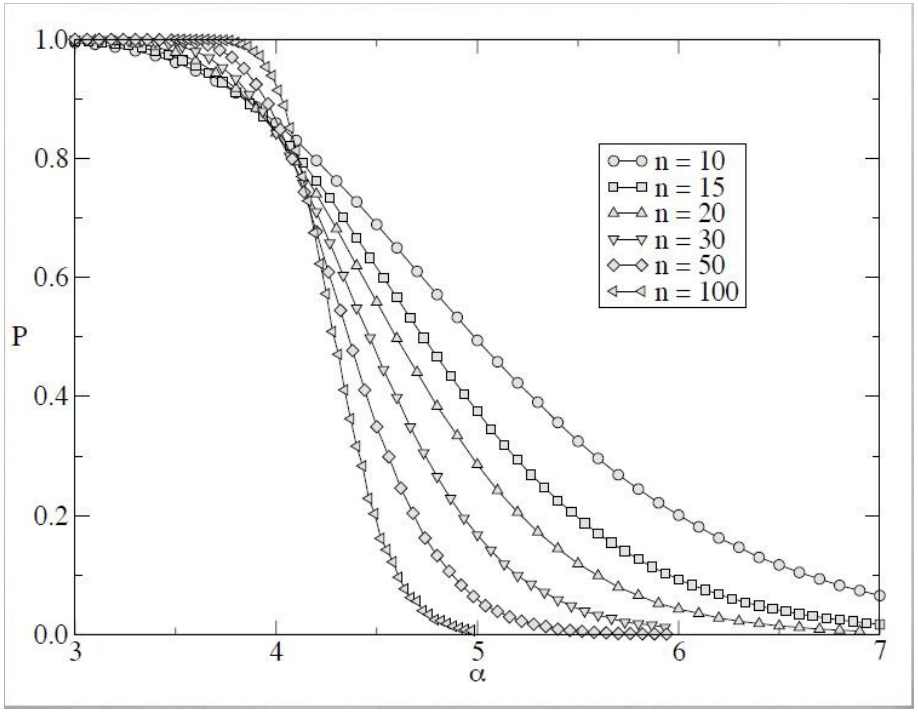
Random 3-SAT

- Varying problem size, *n*

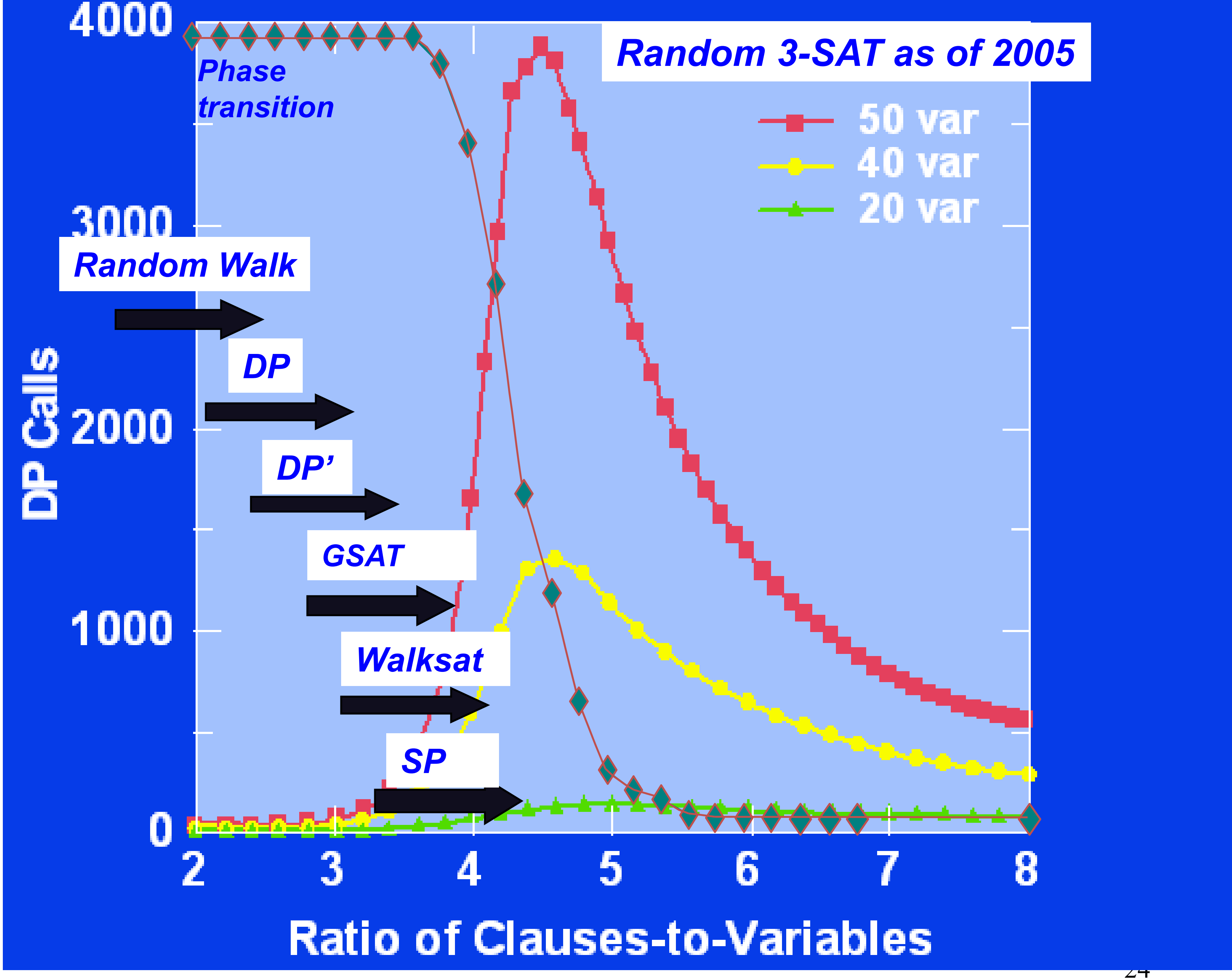
- Complexity peak appears to be largely invariant of algorithm



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

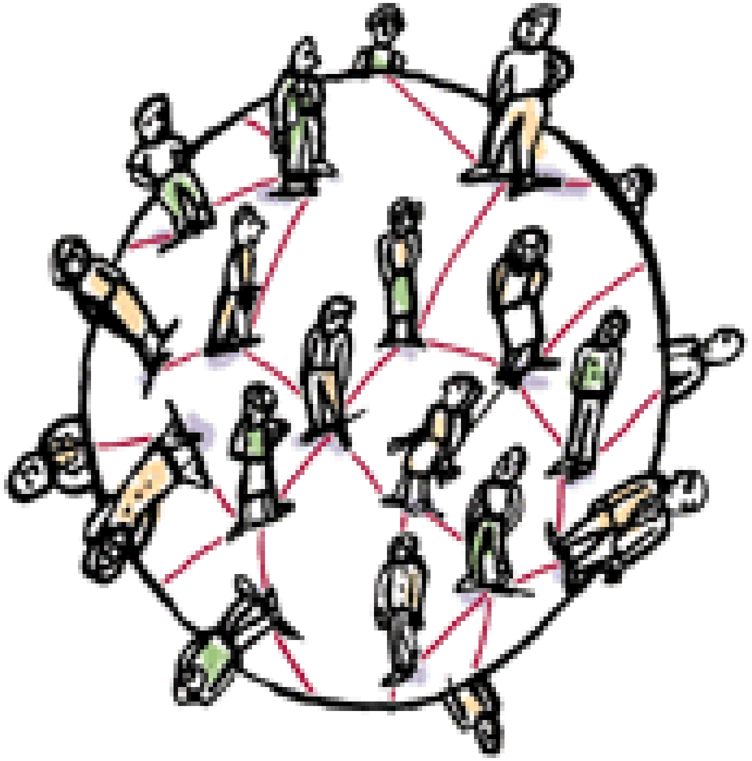


Mitchell, Selman, and Levesque '92

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.

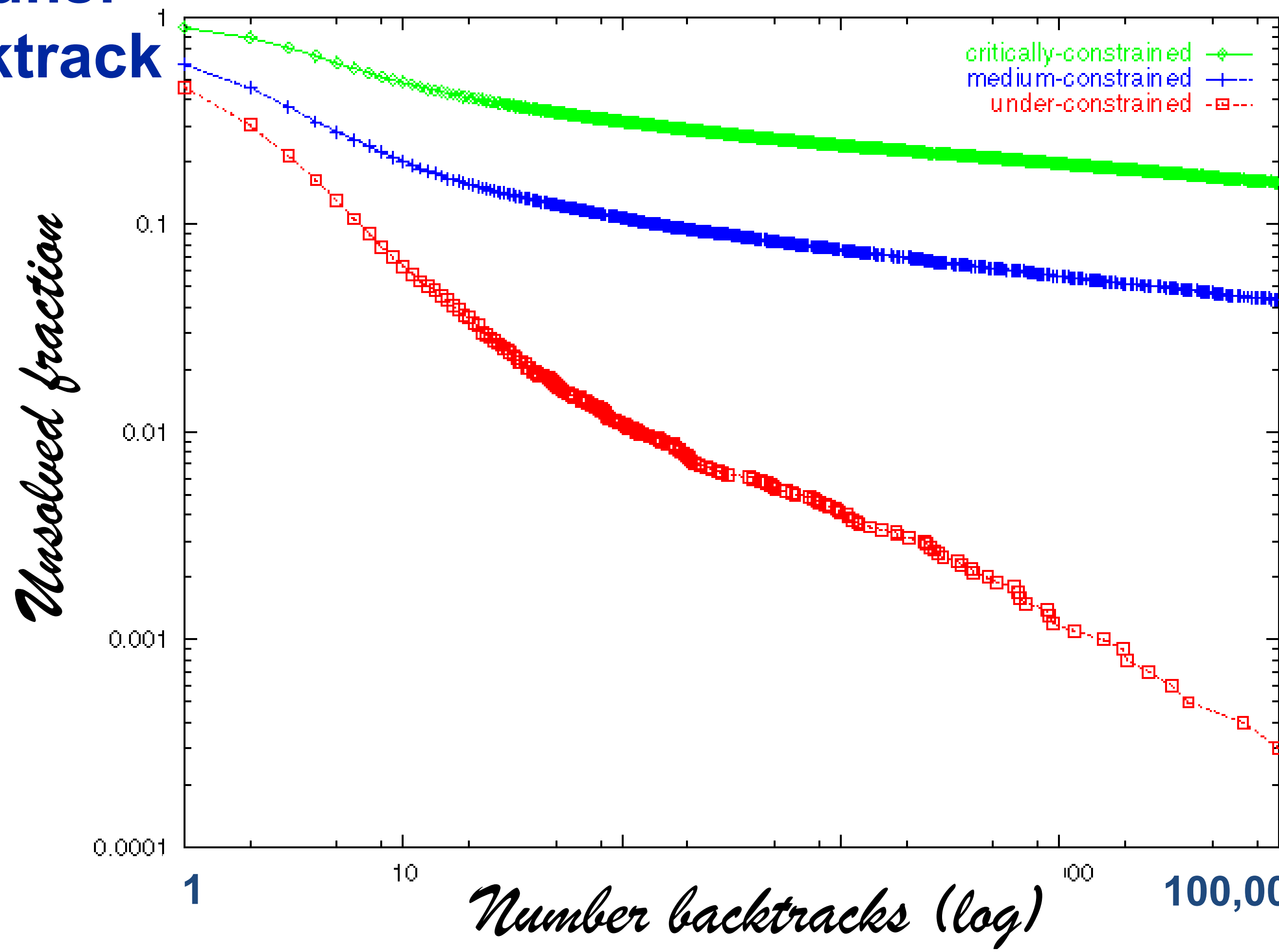
Run time distributions are often "heavy-tailed".

Orders of magnitude difference in run time on different runs.

(Gomes et al. 1998; 2000)

Heavy Tails on Structured Problems

50% runs:
1 backtrack



10% runs:
> 100,000
backtracks

1 **100,000**

Randomized Restarts

Solution: randomize the backtrack strategy

Add noise to the heuristic branching (variable choice) function

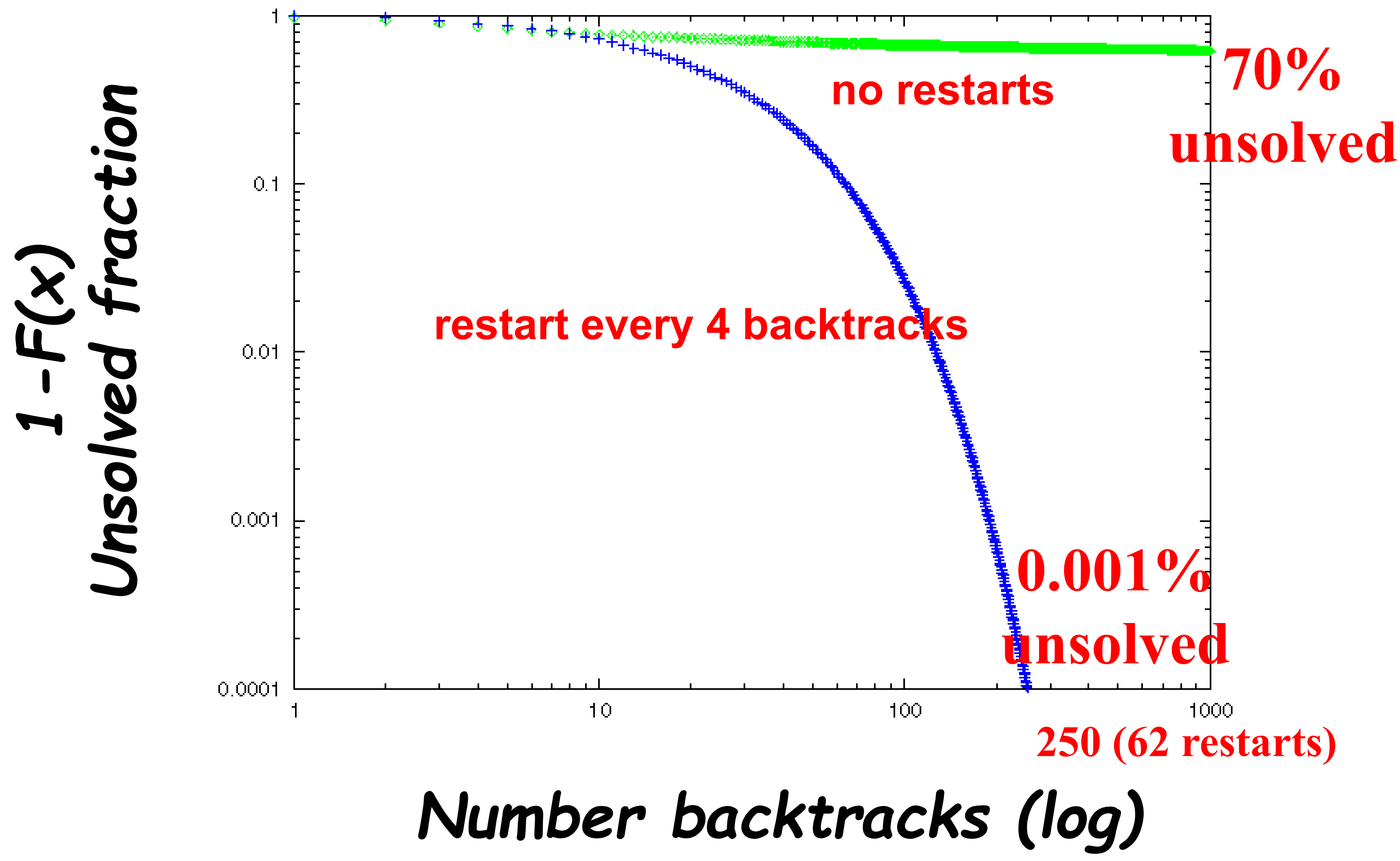
Cutoff and restart search after a fixed number of backtracks

Provably Eliminates heavy tails

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

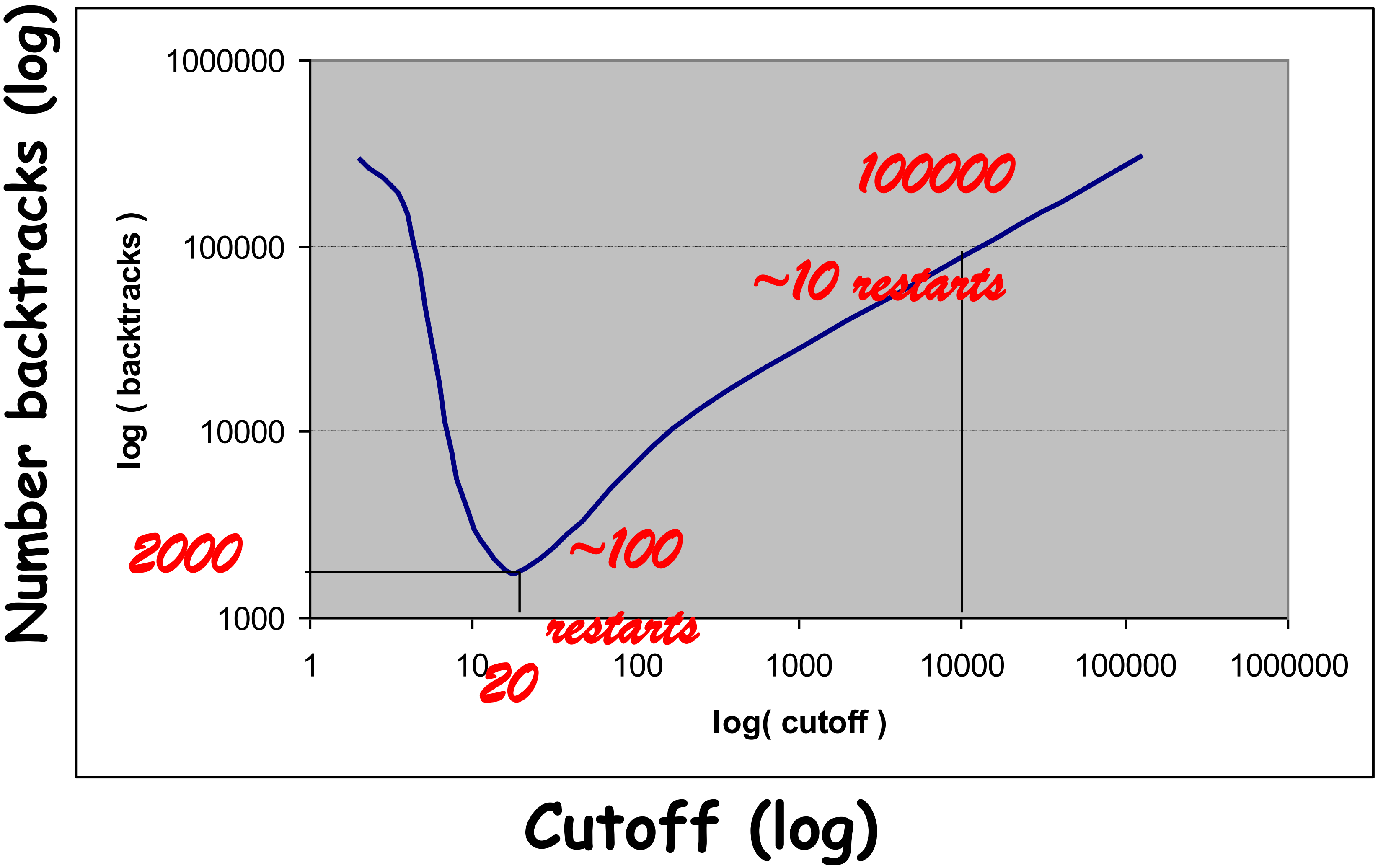
Exploited in many current SAT solvers combined with clause learning and non-chronological backtracking. (e.g., Chaff etc.)

Restarts



1
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



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;*

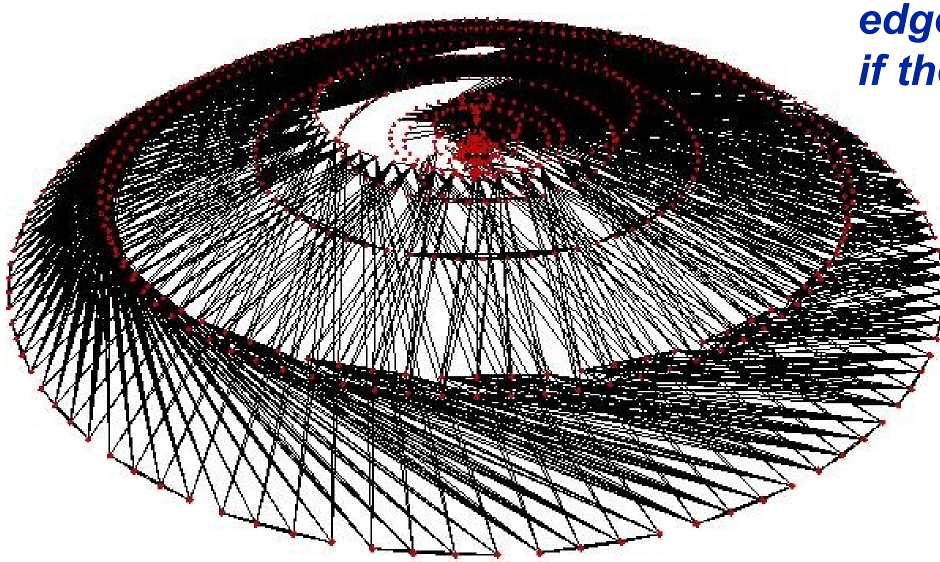
Backdoors can be surprisingly small

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)***

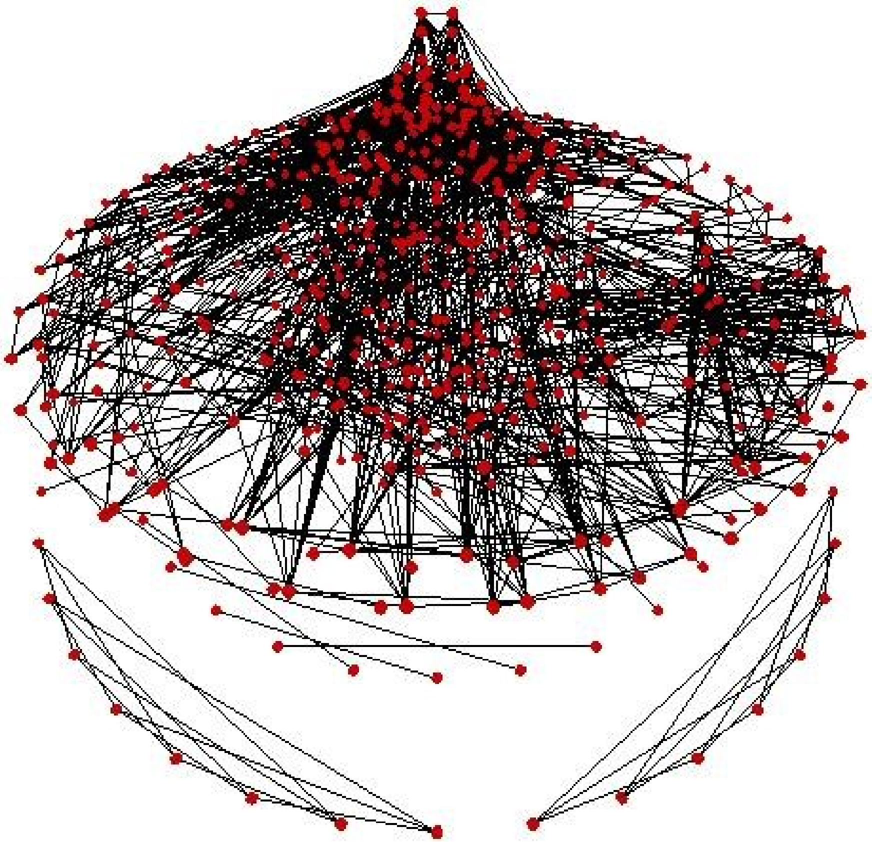
Backdoors capture critical problem resources (bottlenecks).

Backdoors --- “seeing is believing”

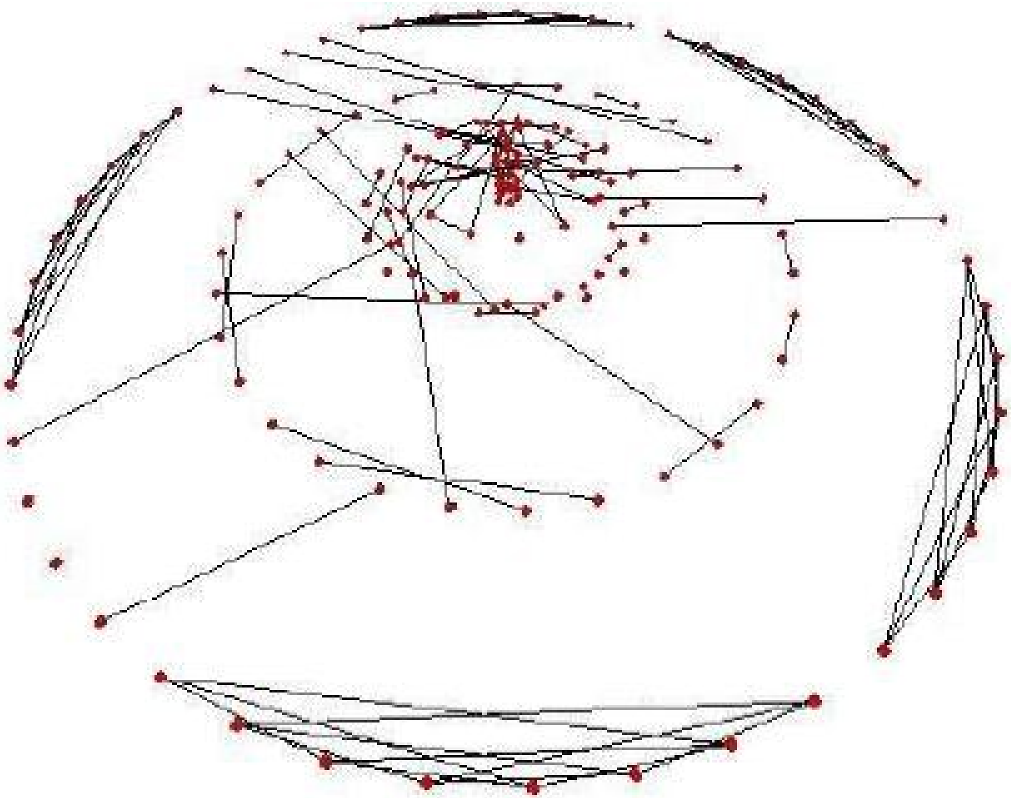


Constraint graph of reasoning problem.
One node per variable:
edge between two variables if they share a constraint.

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

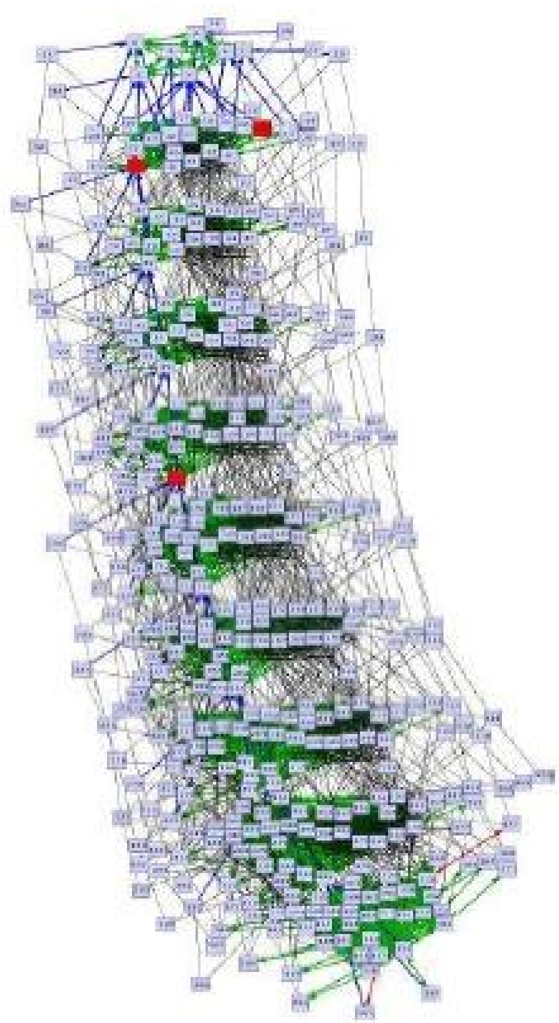


Logistics.b.cnf after setting 5 backdoor vars.

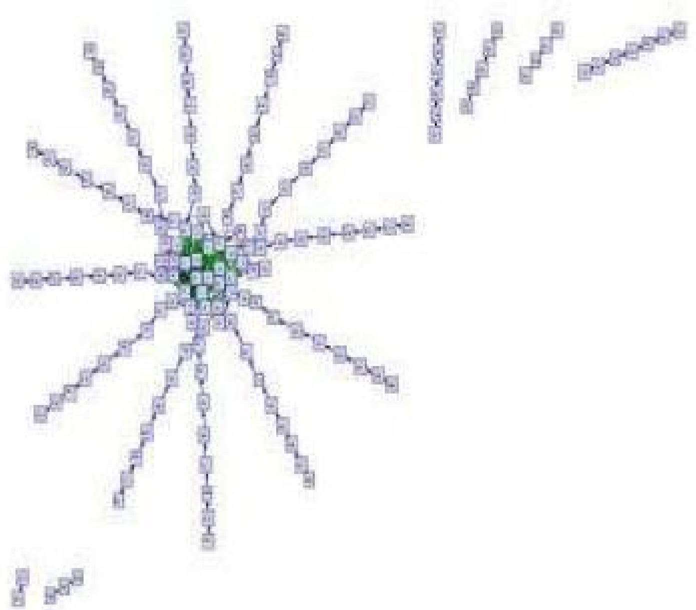


After setting just 12 (out of 800+) backdoor vars – problem almost solved.

Another example

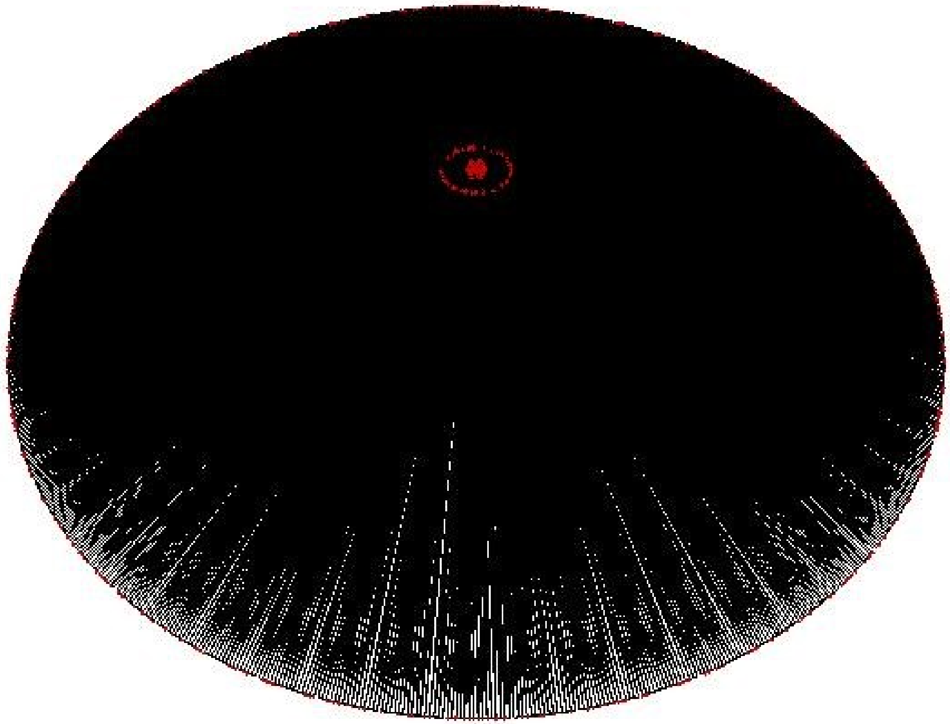


MAP-6-7.cnf infeasible planning instances. Strong backdoor of size 3.
392 vars, 2,578 clauses.

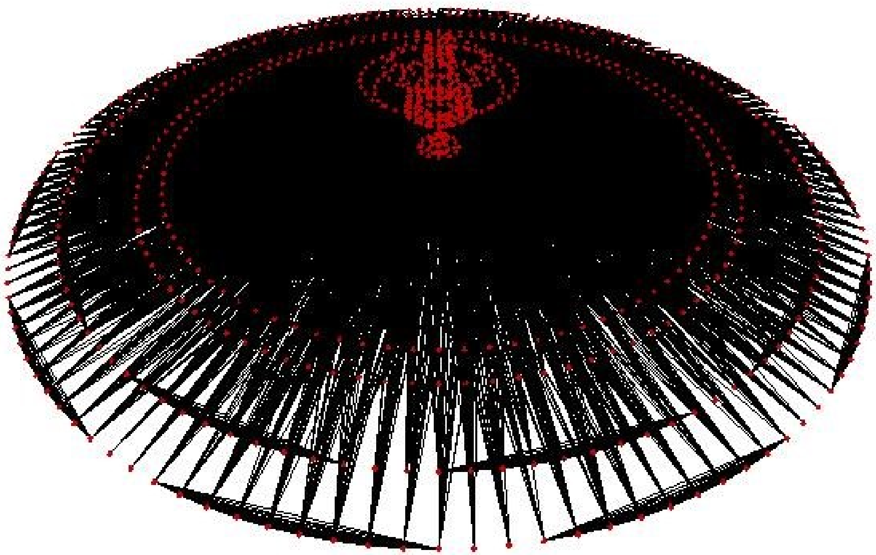


After setting 2 (out of 392) backdoor vars. ---
reducing problem complexity in just a few steps!

Last example.

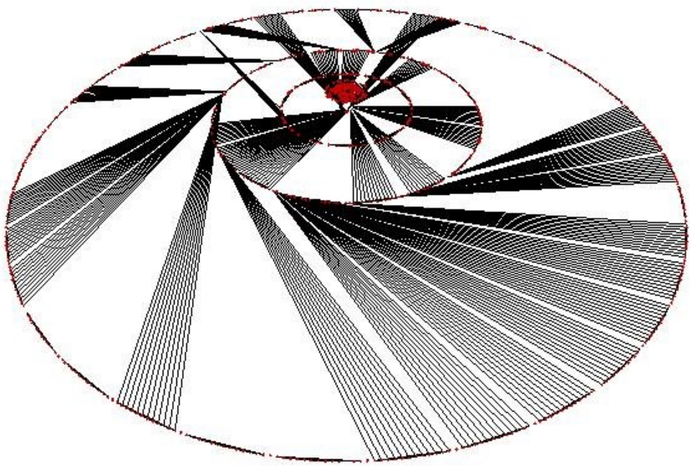


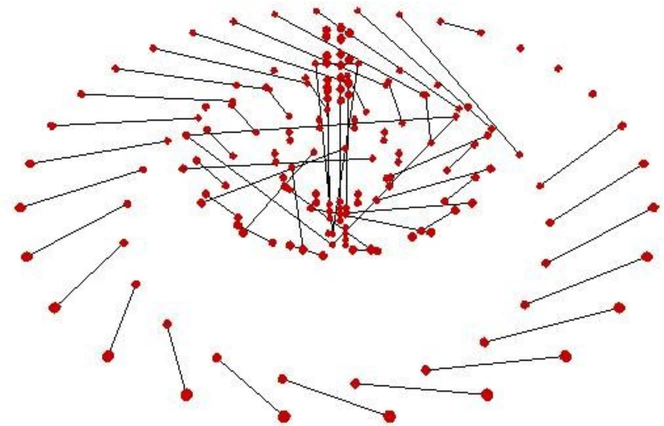
Inductive inference problem --- ii16a1.cnf. 1650 vars, 19,368 clauses.
Backdoor size 40.



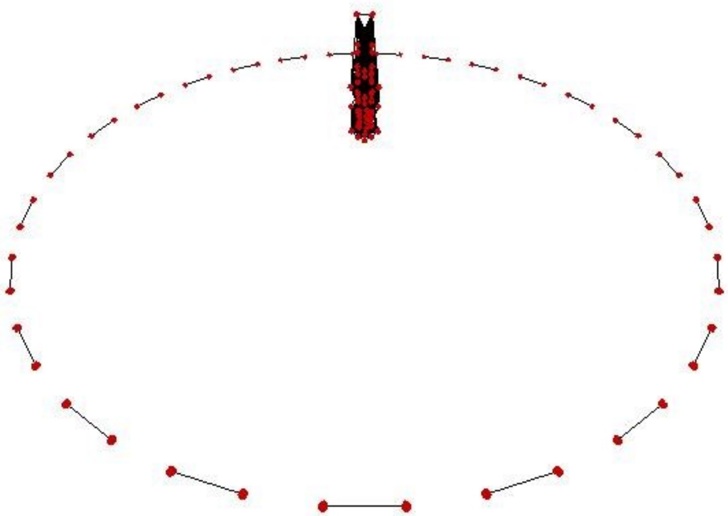
After setting 6 backdoor vars.

Some other intermediate stages:





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



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)