

# **Exploiting Heavy-tailed Phenomena in Computation**

---

**Carla P. Gomes**  
Cornell University

**Joint work with**

**Bart Selman**  
Cornell University

**American Association for the Advancement of  
Science  
2005  
Washington**

# Outline Talk

---

- Hard Computational Problems
- Solving Hard Computational Problems
- Formal Models:
  - Heavy-tailed Phenomena in Computation
  - Explaining and Exploiting Heavy-Tailed Phenomena in Computation:
- Conclusions

---

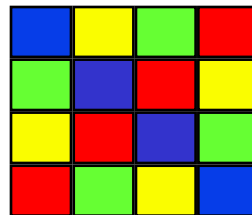
# **Hard Computational Problems**

# Latin Squares: An Abstraction for Real World Applications

---

Given an  $N \times N$  matrix, and given  $N$  colors, a Latin Square of order  $N$  is a colored matrix, such that:

- all cells are colored.
- each color occurs exactly once in each row.
- each color occurs exactly once in each column.



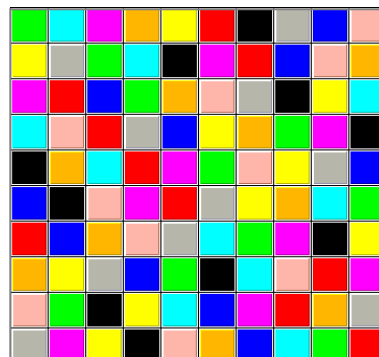
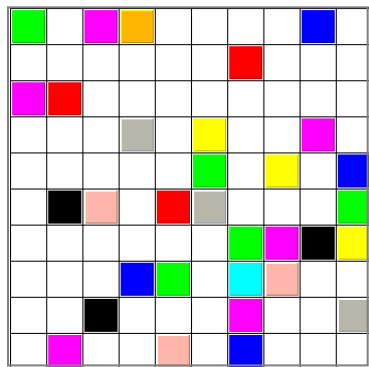
Latin Square  
(Order 4)

# Completing Latin Squares: A Hard Computational Problem

---

Given a partial assignment of colors (10 colors in this case), can the partial quasigroup (latin square) be completed so we obtain a full quasigroup?

Example:



Leads to interesting search problems when structure is perturbed.

**32% preassignment**  
**(Gomes & Selman 97)**

# Underlying Latin Square structure Characterizes many real world applications

teams

	teams					
	.1	2	3	4	5	
•1	X	•R1	•R4	•R2	•R5	•R3
•2	•R1	X	•R2	•R5	•R3	•R4
•3	•R4	•R2	X	•R3	•R1	•R5
•4	•R2	•R5	•R3	X	•R4	•R1
•5	•R5	•R3	•R1	•R4	X	•R2
•6	•R3	•R4	•R5	•R1	•R2	X

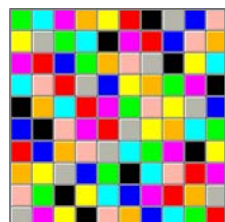
A	D	E	B	C
C	B	A	E	D
D	C	B	A	E
E	A	C	D	B
B	E	D	C	A

## Scheduling and timetabling

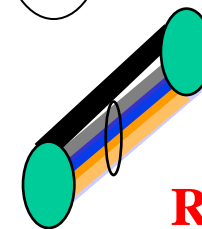
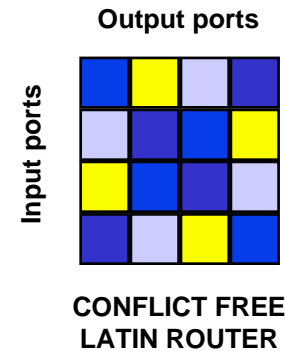
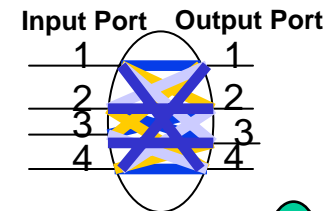
		6					1
7			6				5
8			1	3	2		
		5	4		8		
4		7	2		9		
	8		1	7			
	1	2	5			3	
6			7			8	
2					4		

5	3	6	8	2	7	9	4	1
1	7	2	9	6	4	3	5	8
8	9	4	1	5	3	2	6	7
7	1	5	3	4	9	8	2	6
6	4	3	7	8	2	1	9	5
9	2	8	5	1	6	7	3	4
4	8	1	2	9	5	6	7	3
3	6	9	4	7	1	5	8	2
2	5	7	6	3	8	4	1	9

## Sudoku



## Design of Scientific Experiments



## Routing in Fiber Optic Networks

Many more applications...

# Propositional satisfiability problem (SAT)

---

**SAT:** Given a formula in propositional calculus, is there an assignment to its variables making it true?

Example:

( **A OR NOT B OR NOT C** ) AND ( **B OR NOT C** ) AND ( **A OR C** )

*Solution: A ← True; B ← False; C ← False*

SAT: prototypical hard combinatorial search and reasoning problem.

First problem to be shown to be NP-complete. (Cook 1971)

SAT is also used as a language to express other problems!!!

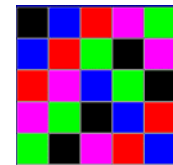
# SATISFIABILITY:

## A language to express other problems

### Latin Squares

---

- Variables:  $n^3$   $x_{ijk} \in \{0,1\}$   
 $x_{ijk}$  cell  $i,j$  has color  $k$ ;  $i,j,k=1,2, \dots,n$ .  
 Each variables represents a color assigned to a cell.



- Clauses:  $O(n^4)$   $\forall_{ij} (x_{ij1} \vee x_{ij2} \dots x_{ijn})$
- Some color must be assigned to each cell (clause of length n);

- No color is repeated in the same row (sets of negative binary clauses);  
 $\forall_{ik} (\neg x_{i1k} \vee \neg x_{i2k}) \wedge (\neg x_{i1k} \vee \neg x_{i3k}) \dots (\neg x_{i1k} \vee \neg x_{ink})$

- No color is repeated in the same column (sets of negative binary clauses);  
 $\forall_{jk} (\neg x_{1jk} \vee \neg x_{2jk}) \wedge (\neg x_{1jk} \vee \neg x_{3jk}) \dots (\neg x_{1jk} \vee \neg x_{njk})$

# **SATISFIABILITY:** **A language to express other problems**

---

**Applications:  
Hardware and Software  
Verification,  
Planning, Protocol Design,,  
etc.**



---

# **Solving Hard Combinatorial Problems**

# Solving Combinatorial Problems

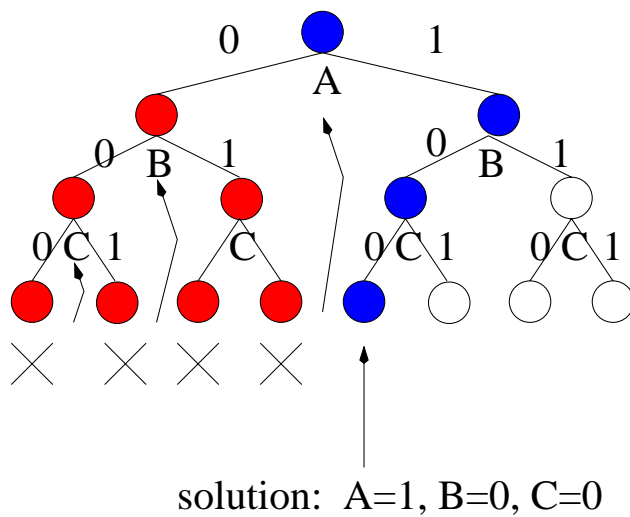
---

Many computational tasks, such as satisfiability, completing Latin squares, scheduling, software verification, etc. can be reduced to:

- Try all possible value assignments to the different variables and pick the best!

# Backtrack Search

( A OR NOT B OR NOT C ) AND ( B OR NOT C ) AND ( A OR C )



Problem:

combinatorial explosion!  
*SAT ---- Worst case*  
*→ N – number variables*  
*→ 2<sup>n</sup> possible assignments*

solution: C=0,B=0,A=1

X - dead end, backtracking must occur

● visited nodes ○ unvisited nodes

↖ backtracking to an ancestor with unvisited child nodes

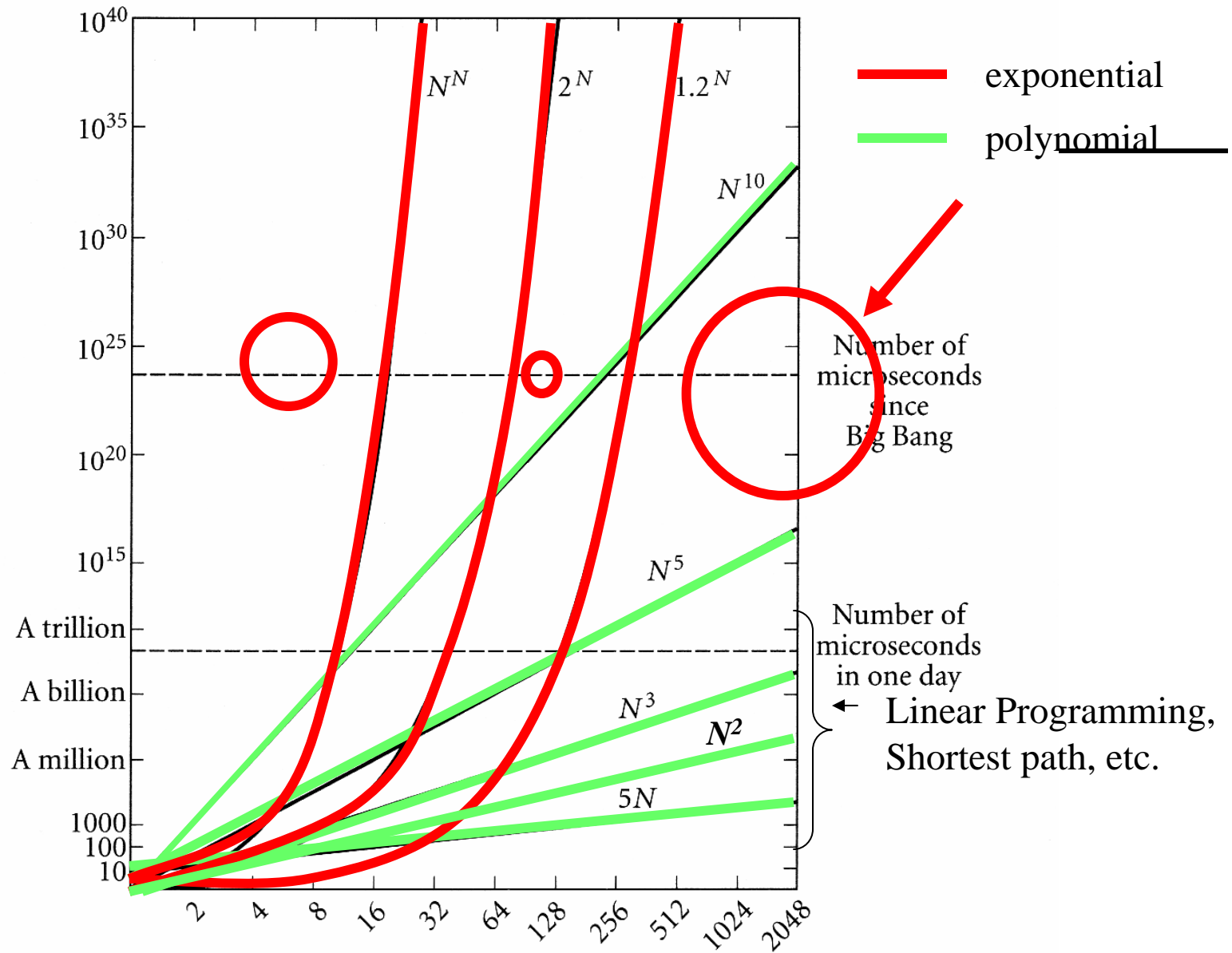
● visited nodes of solution path

**State-of-the-art complete solvers are based on backtrack search procedures;**

# Polynomial vs. exponential growth

(Harel 2000)

SATISFIABILITY





# P vs. NP

## \$1,000,000 Prize



CLAY MATHEMATICS INSTITUTE

Dedicated to increasing and disseminating mathematical knowledge

| Millennium Prize Problems | Clay Research Award | Research | Education | Annual Meeting  
| Popular Lectures | Publications | Forum | About CMI |

Navigate with the  
Figureeight Knot Complement  
at the upper left corner

**Millennium Prize Problems** - To celebrate mathematics in the new millennium, CMI identifies seven old and important mathematics questions that resisted all past attempts to solve them. CMI designates the \$7 million prize fund for their solution, with \$1 million allocated to each "Millennium Prize Problem."



Clay Research A  
[ 2003-11-17.. Ca  
Massachusetts ]

Date: November 1

CLAY RESEARCH AWARD WINNER  
FOR OUTSTANDING ACHIEVEMENT  
IN MATHEMATICS

CAMBRIDGE, MA - The Clay Mathematics Institute



CLAY MATHEMATICS INSTITUTE

Dedicated to increasing and disseminating mathematical knowledge

### P VS NP

The P versus NP Problem

[http://www.claymath.org/Millennium\\_Prize\\_Problems/P\\_vs\\_NP/](http://www.claymath.org/Millennium_Prize_Problems/P_vs_NP/)

---

# **Solving Real-World Hard Combinatorial Problems**

# Significant progress in Complete/Exact search methods for SATISFIABILITY

---

**Applications:  
Hardware and Software  
Verification,  
Planning, Protocol Design,,  
etc.**

Going from 50 variable, 200 constraints

to 1,000,000 variables and 5,000,000 constraints  
in



**But first, what is BIG?**

## “Real World” (begin)

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.

## "Real World" (continued)

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

**#vars. / #clauses**

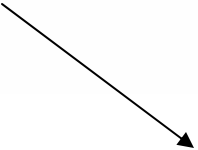
```
p cnf 51639 368352
-1 7 0   →   i.e., (not x_1 or x_7)
-1 6 0           (not x_1 or x_6)
-1 5 0           etc.
-1 -4 0
-1 3 0
-1 2 0
-1 -8 0
-9 15 0
-9 14 0
-9 13 0
-9 -12 0
-9 11 0
-9 10 0
-9 -16 0
-17 23 0
-17 22 0
```

# 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
...

```



**i.e., (x\_33 or x\_25 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

...

# Finally, 15,000 pages later

---

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

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

The Chaff SAT solver (Princeton) solves this instance in a few minutes.

# Gap between theory and practice

---

How can we explain this gap between theory and practice?

What makes this possible?

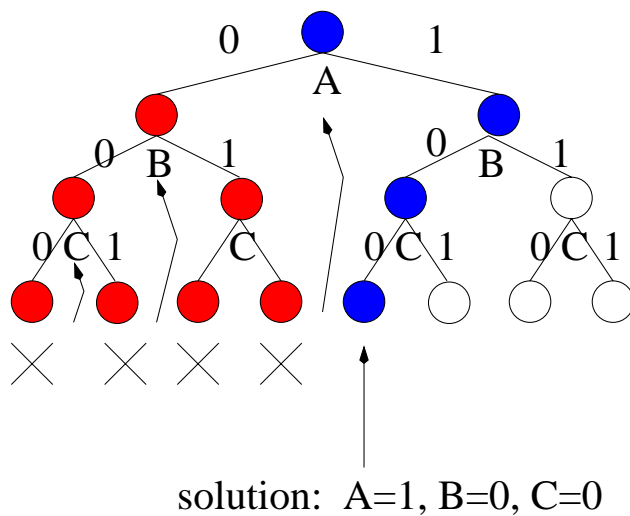
**This talk**  
**recent theoretical insights into the structure of combinatorial search spaces of real-world problems and practical implications that make searching such ultra-large spaces possible.**

---

# **Heavy-tailed Phenomena in Computation**

# Backtrack Search

( A OR NOT B OR NOT C ) AND ( B OR NOT C ) AND ( A OR C )



Problem:

combinatorial explosion!  
*SAT ---- Worst case*  
*→ N – number variables*  
*→ 2<sup>n</sup> possible assignments*

solution: C=0,B=0,A=1

X - dead end, backtracking must occur

● visited nodes ○ unvisited nodes

↖ backtracking to an ancestor with unvisited child nodes

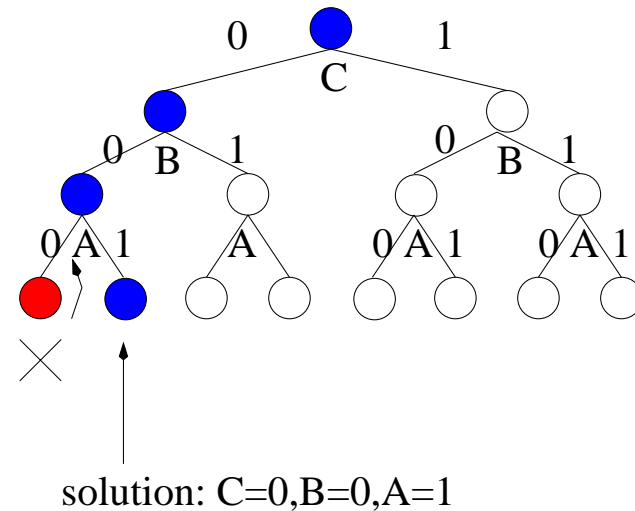
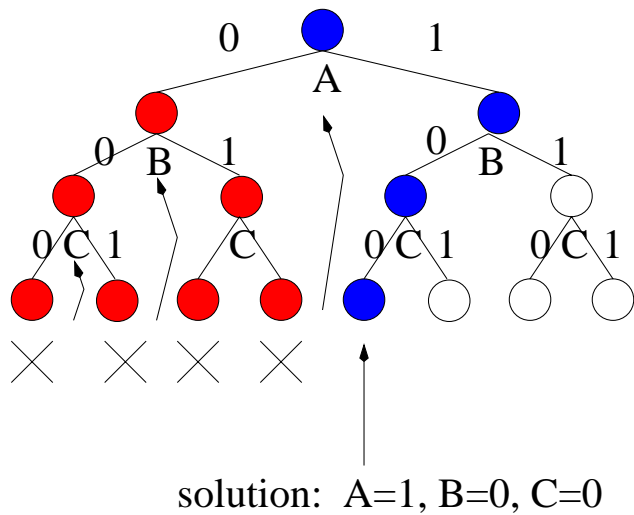
● visited nodes of solution path

**State-of-the-art complete solvers are based on backtrack search procedures;**

# Backtrack Search

## Two Different Executions

( a OR NOT b OR NOT c ) AND ( b OR NOT c ) AND ( a OR c )



✕ - dead end, backtracking must occur

↖ - backtracking to an ancestor with unvisited child nodes

● visited nodes ○ unvisited nodes

● visited nodes of solution path

# Randomized Backtrack Search

---

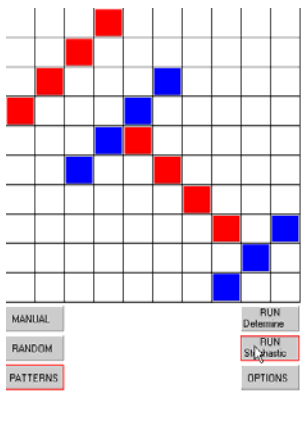
What if we use a good heuristic to select variables/values and introduce a *tiny* element of **randomness into the search heuristic** – e.g., by breaking ties randomly --- and run this (still complete) randomized search procedure on the same instance over and over again?

Study of **runtime distributions of a randomized backtrack search**  
on the same instance :

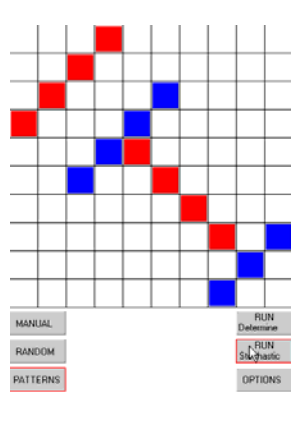
**Way of isolating the variance caused solely by the algorithm**

# Extreme Variance in Runtime of Randomized Backtrack Search

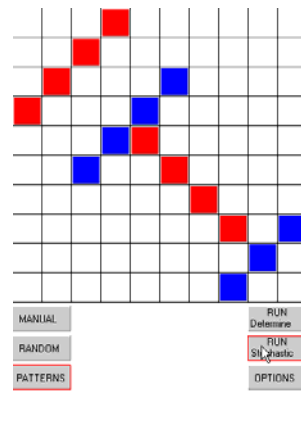
Easy instance – 15 % preassigned cells



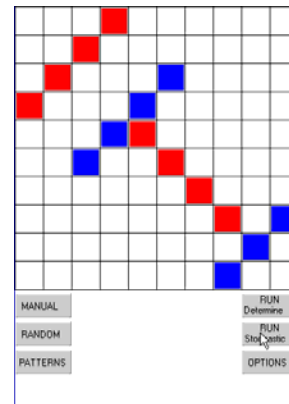
Time: 7



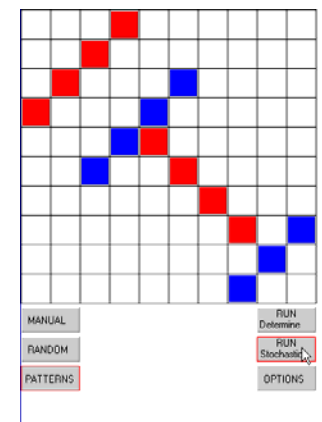
11



30



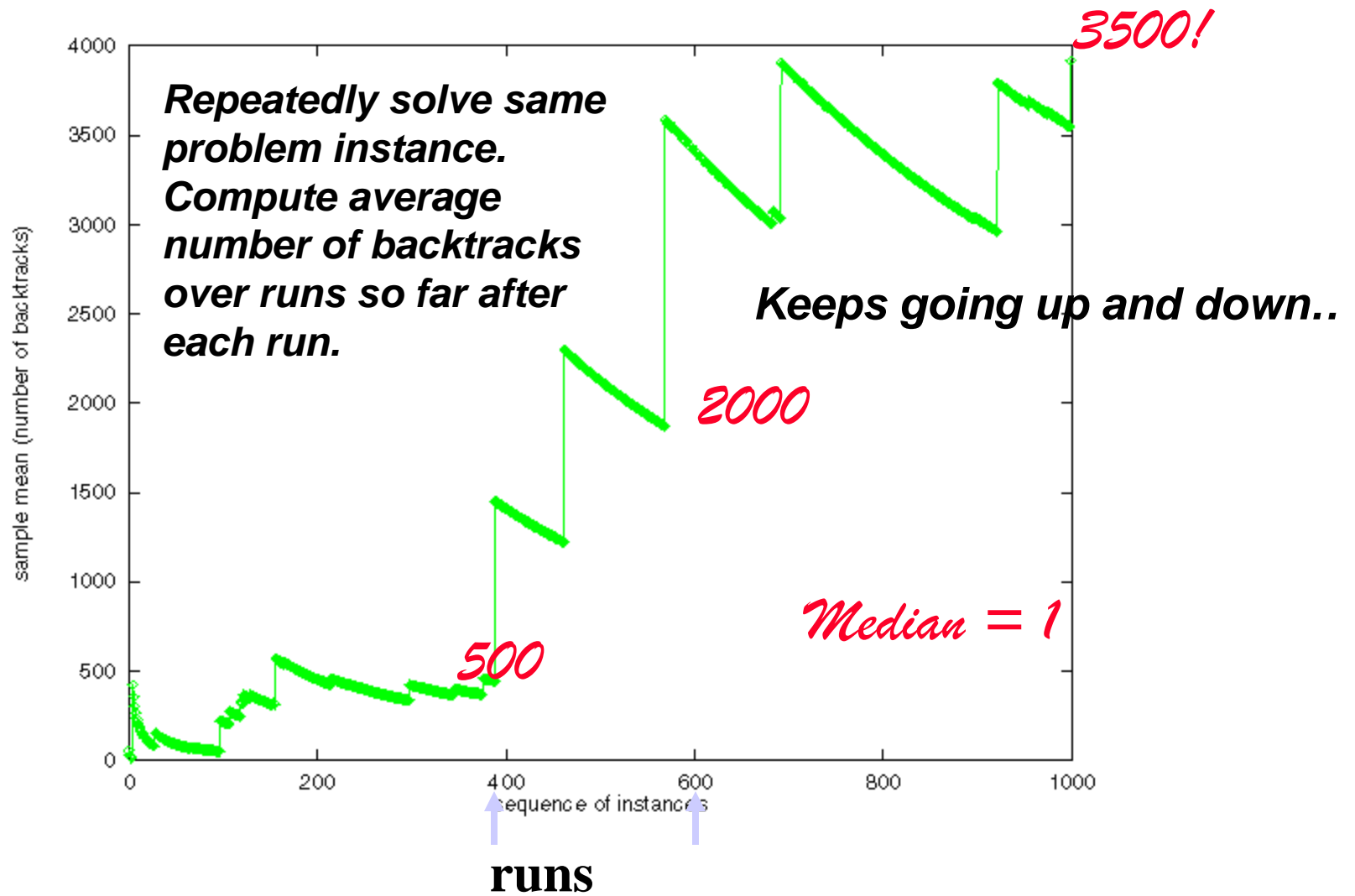
>2000



>2000

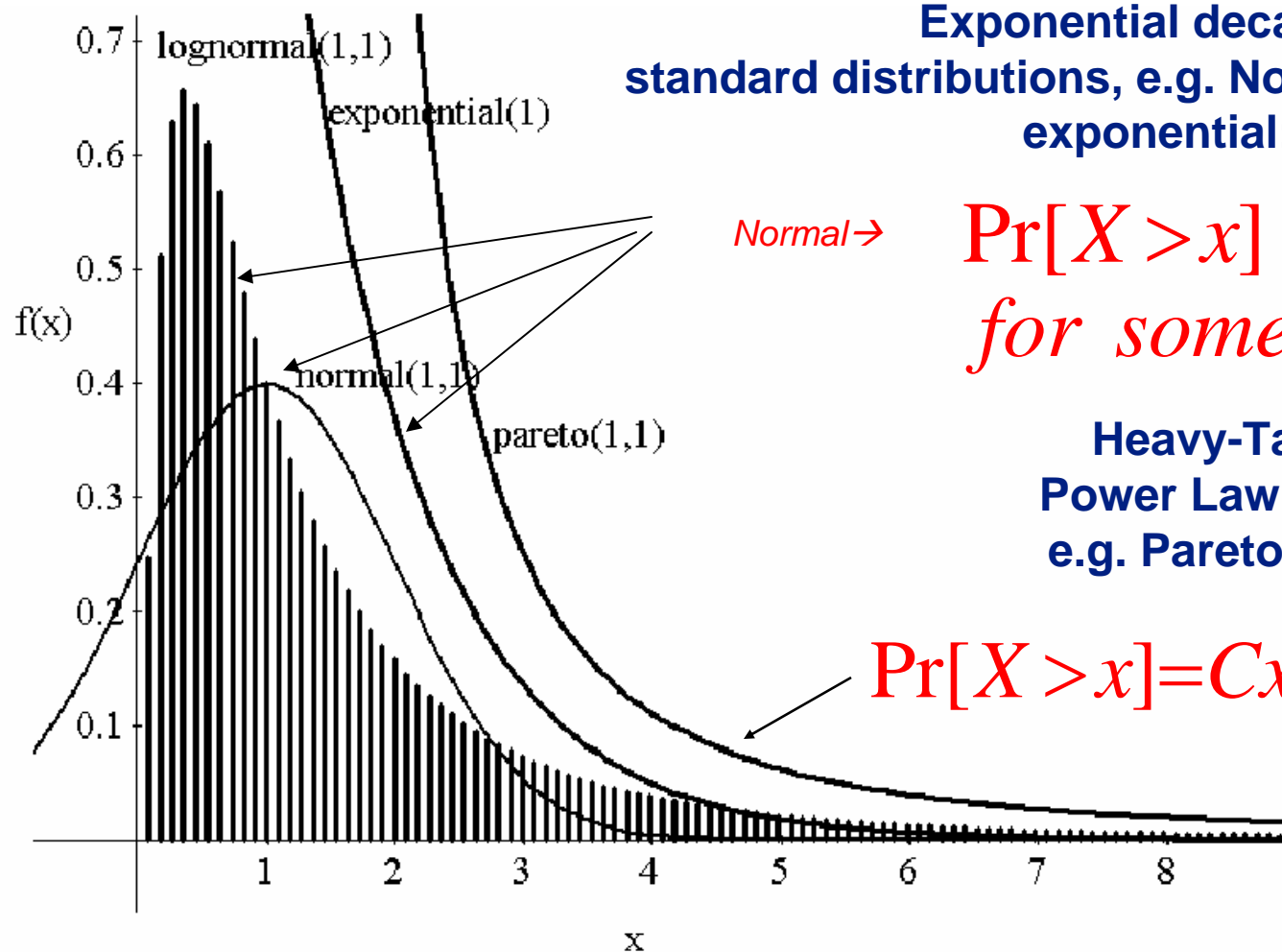
# Search cost of backtrack search method on Latin Square problem

average



Gomes Selman Crato 2000)

# Heavy-tailed distributions



Exponential decay for standard distributions, e.g. Normal, Logonormal, exponential:

Normal  $\rightarrow$   $\Pr[X > x] \approx Ce^{-x^2}$ ,  
for some  $C > 0$

Heavy-Tailed  
Power Law Decay  
e.g. Pareto-Levy:

$\Pr[X > x] = Cx^{-\alpha}, x > 0$

# Exploiting Heavy-Tailed Behavior

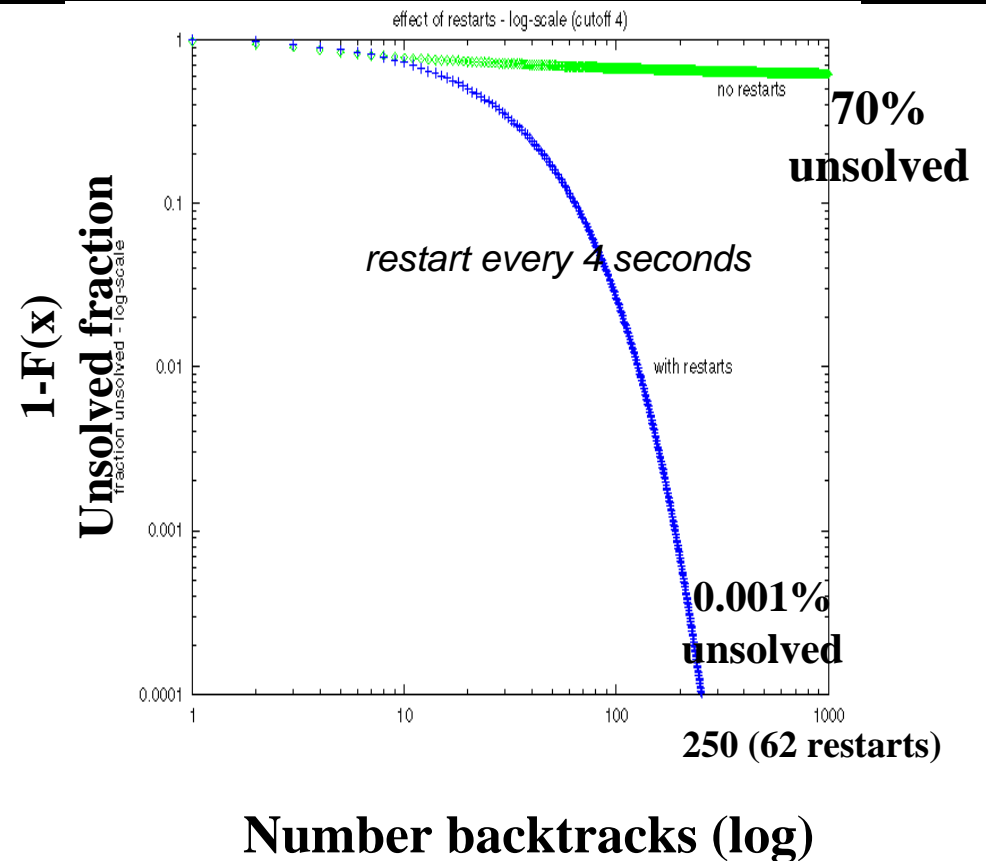
**Heavy Tailed behavior** has been observed in **several domains**:

- Latin Squares, • Graph Coloring, • Planning, • Scheduling, • Circuit synthesis, • Decoding, etc.

Consequence for algorithm design:

Use **restarts** or **parallel / interleaved runs** to combat the extreme variance performance.

**Restarts provably eliminate heavy-tailed behavior (Gomes Selman Crato 2000)**



---

# **Explaining Heavy-Tailed Phenomena: Formal Models**

# Formal Models: Heavy-tailed Distributions

---

- Explain very long runs of complete solvers;
- But also imply the existence of a wide range of solution times, often from very short runs to very long

**How to explain  
short runs?**

**Backdoors**



# Backdoors: Intuitions

---

Real World Problems are characterized  
by **Hidden Tractable Substructure**

## **BACKDOORS**

Subset of the “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**

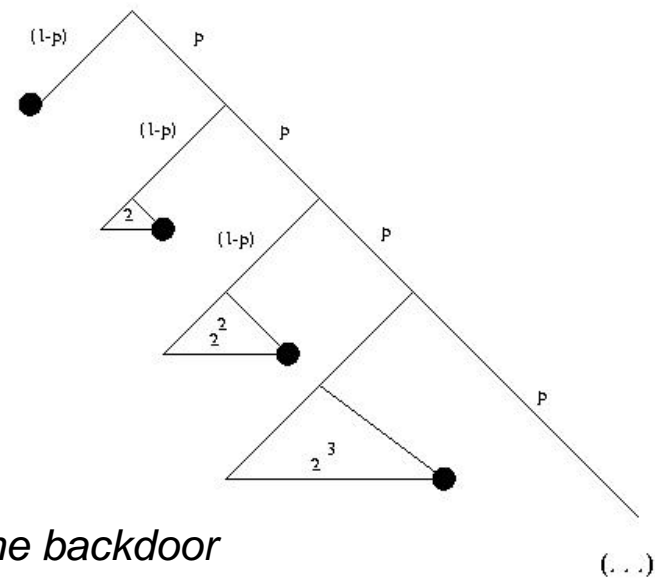
# Explaining short runs: Backdoors to tractability

1 backdoor

Formally:

we define notion of “subsolver”  
(handles tractable substructure of  
problem instance in polynomial  $t$

backdoors and strong backdoors



$p$  – probability of not finding the backdoor

● successful leaf

$T$  - the number of leaf nodes visited up to and including  
the successful node;  $b$  - branching factor

$$P[T = b^i] = (1-p)p^i \quad i \geq 0$$

(Chen, Gomes, Selman 01)

# Three Regimes

---

Regime 1:

finite expected time, finite variance

$$p \leq \frac{1}{b^2}$$

Regime 2:

finite expected time, infinite variance

$$\frac{1}{b^2} < p < \frac{1}{b}$$

Regime 3:

infinite expected time, infinite variance

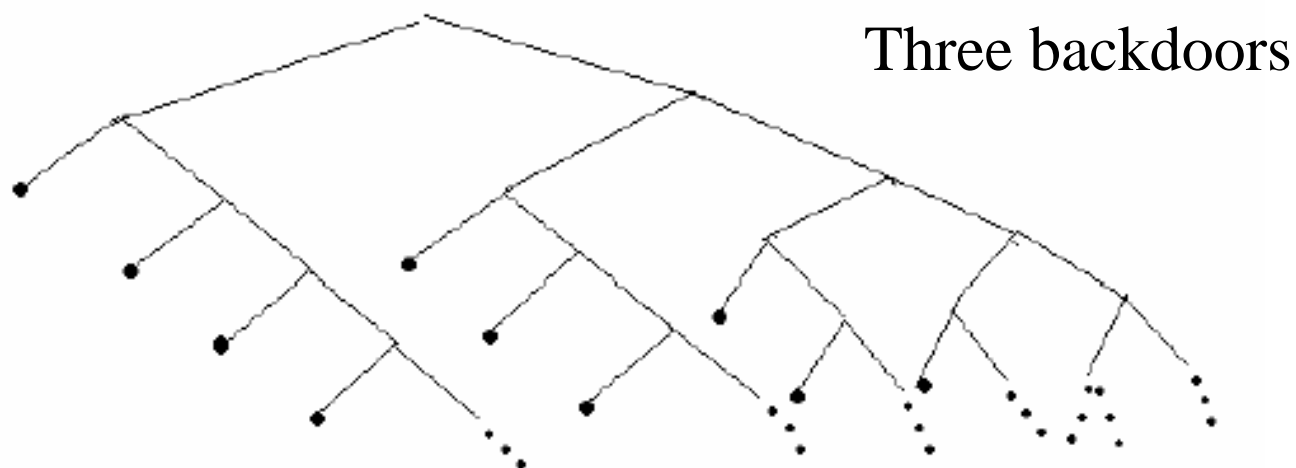
$$p \geq \frac{1}{b}$$

*p* – probability of not finding the backdoor

(Chen, Gomes, Selman 2001)

# Backdoors: formal model explaining heavy-tailed search behavior.

---



**Theorem 3.** (Heavy-tail lower bound) If  $B \in o(N/\log N)$  and the probability of heuristic failure  $(1 - 1/h)$  is less than  $1/b^\alpha$  for  $\alpha \in (0, 2)$ , then the tail probability of the search cost on  $V(B)$  is lower bounded by a heavy-tail.

Can formally relate **size of backdoor** and **strength of heuristics** (captured by its failure probability to identify backdoor variables) to occurrence of heavy tails in backtrack search.

(Williams, Gomes, Selman, 2003)

## Backdoors in real-world problems can be surprisingly small

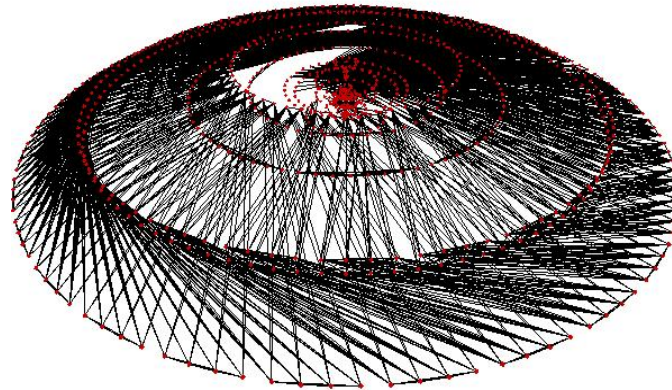
---

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

(Gomes, Selman, 2004)

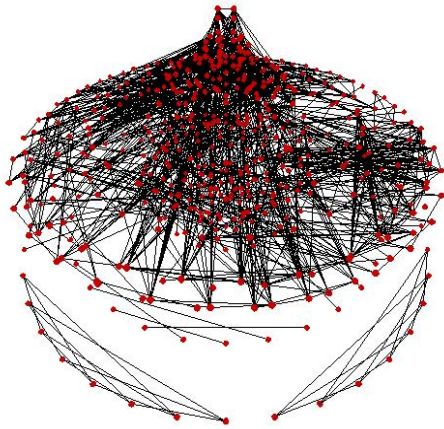
Recent results – syntactic domains for planning with provably small backdoor sets

# Backdoors: Visualization

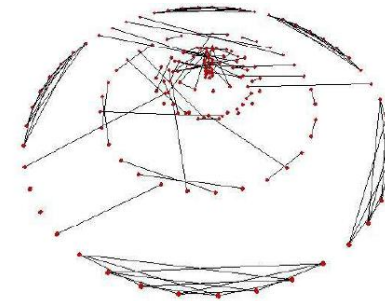


Initial Constraint Graph

*Logistics planning problem formula  
843 vars, 7,301 constraints – 16 backdoor variables  
(visualization by Anand Kapur)*



After setting 5 backdoor vars



After setting 12 backdoor vars

---

# **Exploiting Backdoors**

# Exploiting Backdoors: Algorithms that explicitly look for backdoors

---

We cover three kinds of strategies for dealing with backdoors:

- A **deterministic** *algorithm*

- A **complete randomized** *algorithm*

*Provably better performance over the deterministic one*

- A **heuristicly guided complete randomized algorithm**

*Assumes existence of a good heuristic for choosing variables to branch on*

*We believe this is close to what happens in practice*

# Runtime Table for Algorithms

---

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

Upper bound on  
backdoor size  
given  $n$  variables

(Williams, Gomes, Selman, 2003)

When the backdoor is relatively small  
and we use a “decent” heuristic,  
the algorithm is guaranteed to run in  
polynomial time –

# Conclusions

---

During the past few years, we have obtained a much better **understanding** of the **nature and structure** of computationally hard problems with the **identification of heavy-tailed behavior**;

**“Backdoor” sets**.: encapsulate the combinatorics of a problem instance, as dealt with in practice. Backdoors can be **surprisingly small in practice**.

- Provide insight into the scalability of current solvers;
- Search heuristics + randomization are provably efficient.

**Current SAT solvers use restart strategies**  
**De facto looking for Backdoors!!!!**

# Conclusions

---

Research area with rich interactions between:

- computer science, • operations research,
- mathematics, and • statistical physics

and between

- theory, • experiments, and • applications.

Very exciting and  
promising research area 😊!

---

**The End**

**[www.cs.cornell.edu/gomes](http://www.cs.cornell.edu/gomes)**