Leveraging Belief Propagation, Backtrack Search, and Statistics for **Model Counting**

Lukas Kroc, Ashish Sabharwal, Bart Selman
Cornell University

May 23, 2008
CPAIOR-08 Conference, Paris

Talk Outline

- Model counting / solution counting
  - Problem, motivation, background
- Lower bound estimates
  - Fast, using belief propagation for guidance
  - Probabilistic correctness guarantees
  - Experiments
- Upper bound estimates
  - Fast, using multiple runs of backtrack search solvers
  - Statistical correctness guarantees
  - Experiments

Model/Solution Counting

[model ≡ solution ≡ satisfying assignment]

Model Counting (#CSP): Given a constraint satisfaction problem P, ***how many* solutions** does P have?

Techniques generic for CSPs, but presented and evaluated for SAT

E.g. propositional formula F = (a or b) and (not (a and (b or c)))

Boolean variables: a, b, c

Total 2^3 possible 0-1 truth assignments

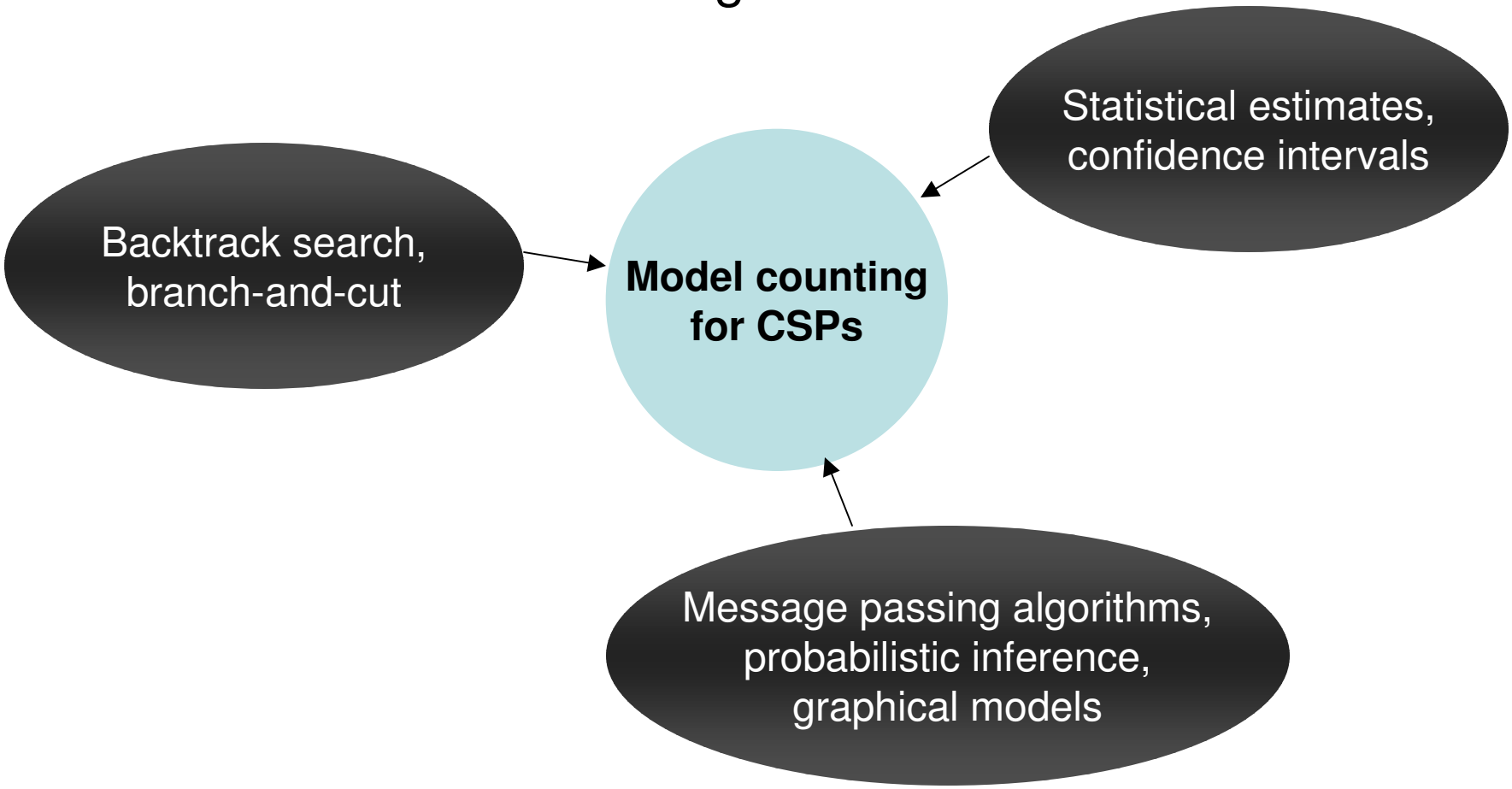
F has exactly 3 **satisfying assignments** (a,b,c) :

(1,0,0), (0,1,0), (0,1,1)

- Must continue searching after one solution is found
- With N variables, can have anywhere from 0 to 2^N solutions
- Will denote the model count by #F

This Work …

brings together several techniques for fast solution count estimates with correctness guarantees

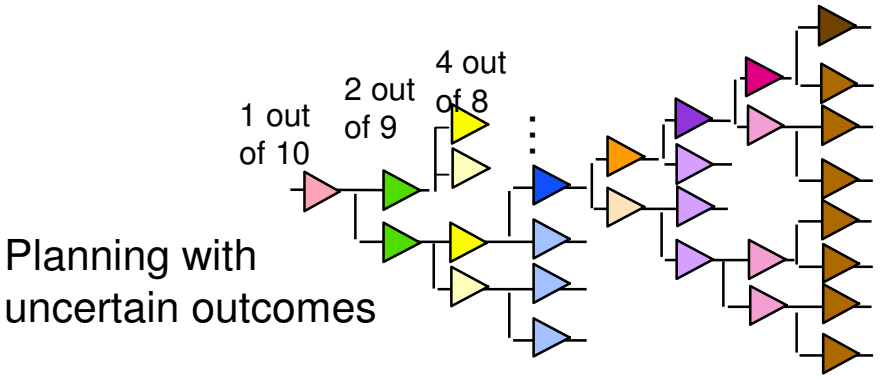


Why Model Counting and Sampling?

Efficient model counting techniques will extend the success of constraint reasoning to a whole new range of applications

- Probabilistic reasoning / uncertainty / Bayesian inference
  e.g. Markov logic networks [Richardson-Domingos-06]
- Multi-agent / adversarial reasoning (bounded)
- New avenues in planning and model checking

[Roth-96, Littman-etal-01, Sang-etal-04, Darwiche-05, Domingos-06, …]



Planning with uncertain outcomes

Computational Complexity Hierarchy

EXP-complete:
games like Go, …

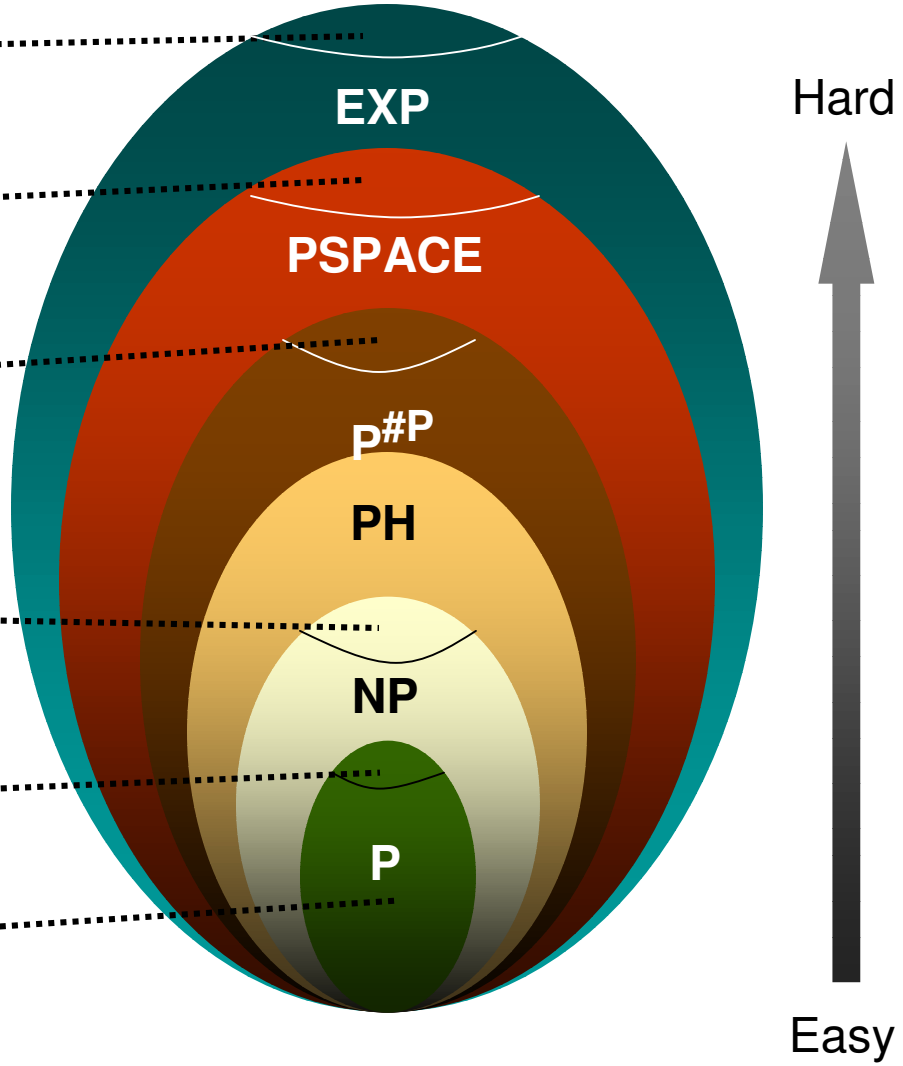
PSPACE-complete:
QBF, adversarial
planning, chess (bounded), …

#P-complete/hard:
#SAT, sampling,
probabilistic inference, …

NP-complete:
SAT, scheduling,
graph coloring, puzzles, …

P-complete:
circuit-value, …

In P:
sorting, shortest path, …

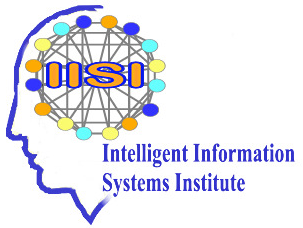


Note: widely believed hierarchy; know P≠EXP for sure

The Challenge of Model Counting

- In theory
  - Model counting is #P-complete
    (believed to be much harder than NP-complete problems)
  - Even for 2CNF-SAT and Horn-SAT!

- Practical issues
  - Often finding even a single solution is quite difficult!
  - Typically have huge search spaces
    - E.g. $2^{1000} \approx \mathbf{10^{300}}$ truth assignments for a 1000 variable formula
  - Solutions often sprinkled unevenly throughout this space
    - E.g. with $\mathbf{10^{60}}$ **solutions**, the chance of hitting a solution at random is $\mathbf{10^{-240}}$



Main message: *If you can solve the problem, you can often count solutions very well!*

- Solve using Belief Propagation
  - Earlier work: “if you can sample using local search…”
- Solve using standard backtrack search
  - Earlier work: lower bounds, “xor” constraints

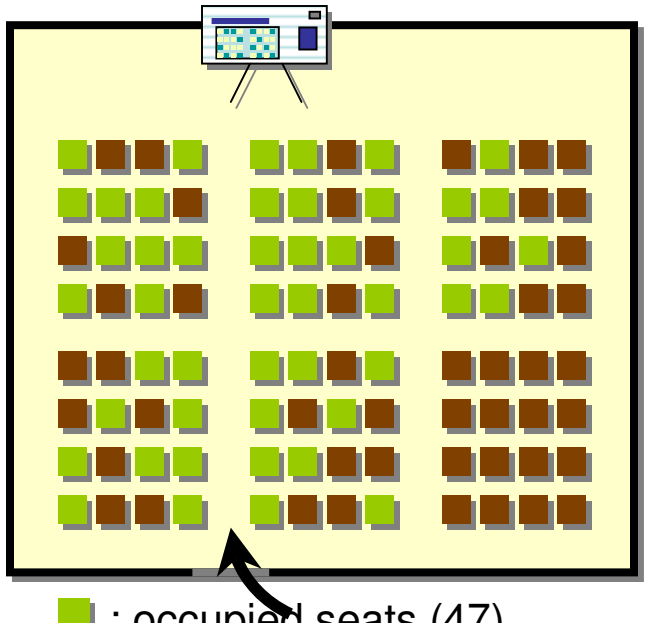
How Might One Count?

How many people are present in the hall?

Problem characteristics:

- Space naturally divided into rows, columns, sections, …
- Many seats empty
- Uneven distribution of people (e.g. more near door, aisles, front, etc.)

How Might One Count?



: occupied seats (47)

: empty seats (49)

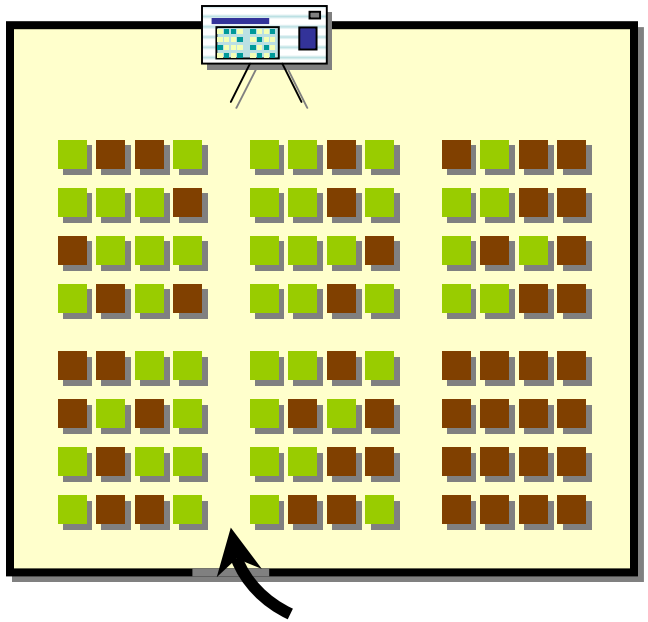
Previous approaches:

1. Brute force
2. Branch-and-bound
3. Estimation by sampling
4. Knowledge compilation (BDD, DNNF)

Contribution from our group: randomized strategies using

- random XOR/parity constraints [AAAI-06/07, SAT-07]
- solution samples from local search [IJCAI-07]
- **belief propagation + statistical upper bounds** [CPAIOR-08]

#1: Brute-Force Counting



Idea:

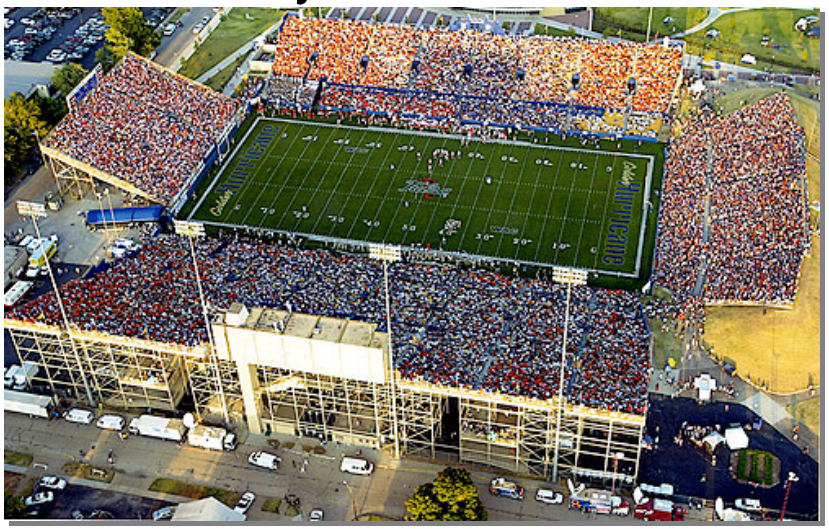
- Go through every seat
- If occupied, increment counter

Advantage:

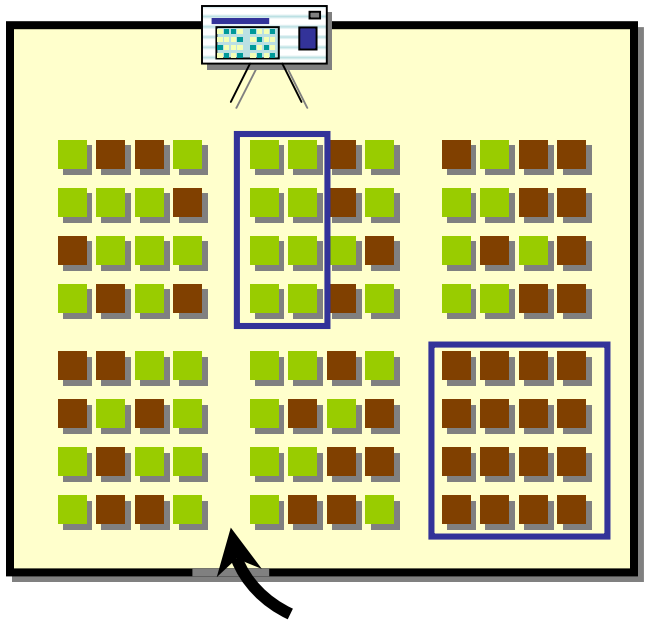
- Simplicity, accuracy

Drawback:

- Scalability



#2: Branch-and-Bound (DPLL-style)



Framework used in DPLL-based systematic exact counters
e.g. Relsat [Bayardo-etal-00], Cachet [Sang-etal-04]

Idea:

- Split space into sections
  e.g. front/back, left/right/ctr, …
- Use smart detection of full/empty sections
- Add up all partial counts

Advantage:

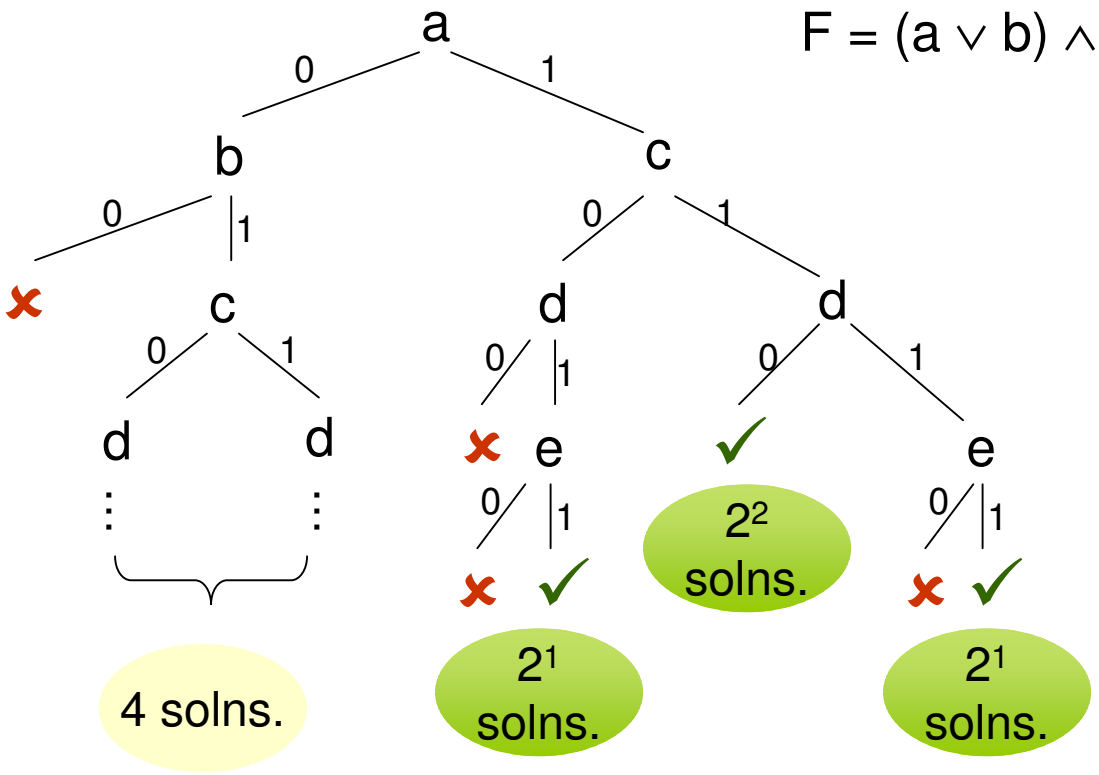
- Relatively faster, exact

Drawback:

- Still "accounts for" every single person present: need extremely fine granularity
- Scalability

#2: Branch-and-Bound (DPLL-style)

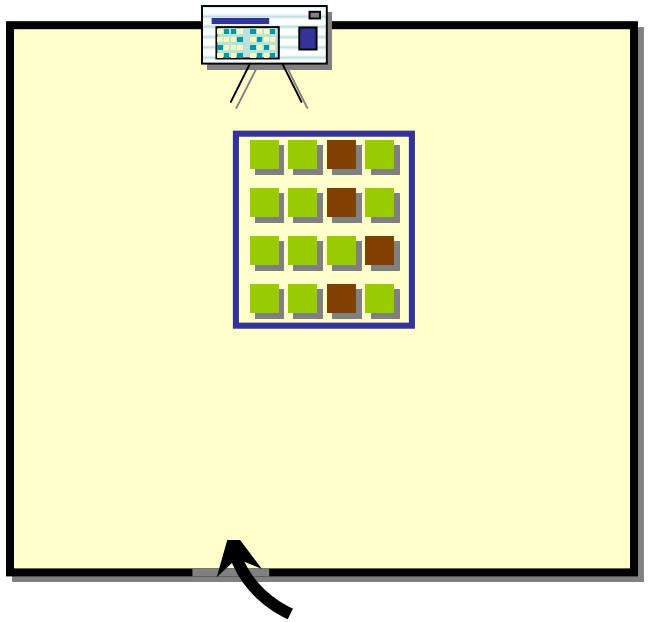
- For an N variable formula, if the residual formula is satisfiable after fixing d variables, count 2^{N-d} as the model count for this branch and backtrack.



$F = (a \vee b) \wedge (c \vee d) \wedge (\neg d \vee e)$

Total 12 solutions

#3: Estimation By Sampling -- Naïve



Idea:

- Randomly select a region
- Count within this region
- Scale up appropriately

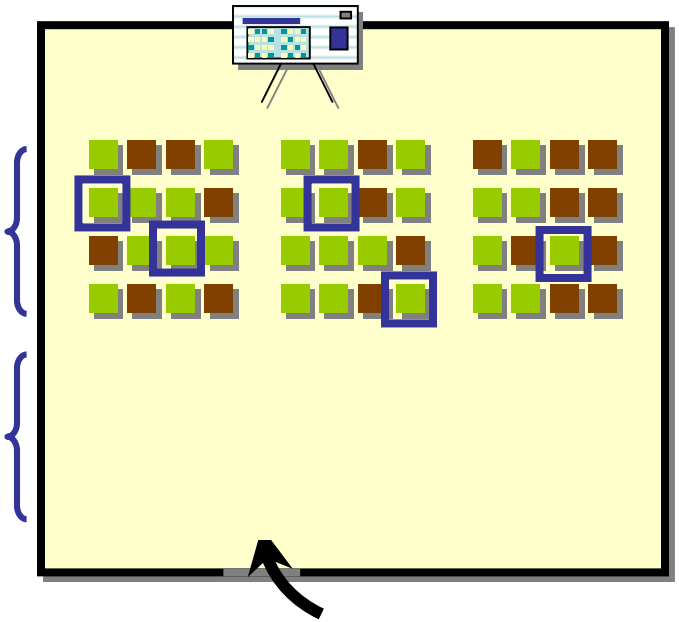
Advantage:

- Quite fast

Drawback:

- Robustness: can easily under- or over-estimate
- Scalability in sparse spaces: e.g. 10^{60} solutions out of 10^{300} means need region much larger than 10^{240} to "hit" any solutions

#3: Estimation By Sampling -- Smarter



Framework used in approximate counters like ApproxCount [Wei-Selman-05]

Idea:

- Randomly sample *k occupied* seats
- Compute fraction in front & back
- Recursively count only front
- Scale with appropriate multiplier

Advantage:

- Quite fast

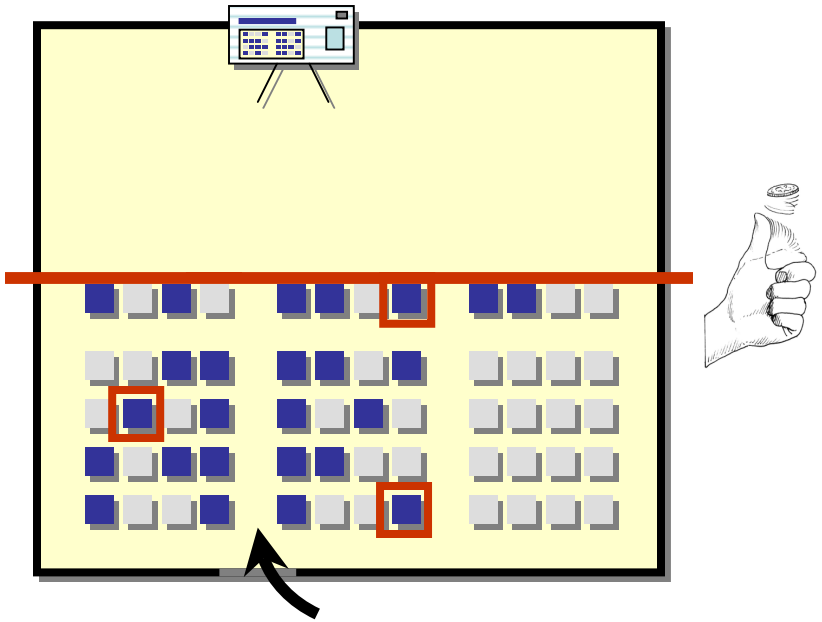
Drawback:

- Relies on uniform sampling of occupied seats -- not any easier than counting itself!
- Robustness: often under- or over-estimates; no guarantees

Talk Outline

- Lower bound estimates
  - Fast, using belief propagation for guidance
  - Probabilistic correctness guarantees
  - Experiments
- Upper bound estimates
  - Fast, using multiple runs of backtrack search solvers
  - Statistical correctness guarantees
  - Experiments

Estimation with Guarantees



Idea:

- Identify a "balanced" row split or column split (roughly equal number of people on each side)
  - Use sampling for estimate
- Pick one side *at random*
- Count on that side recursively
- Multiply result by 2

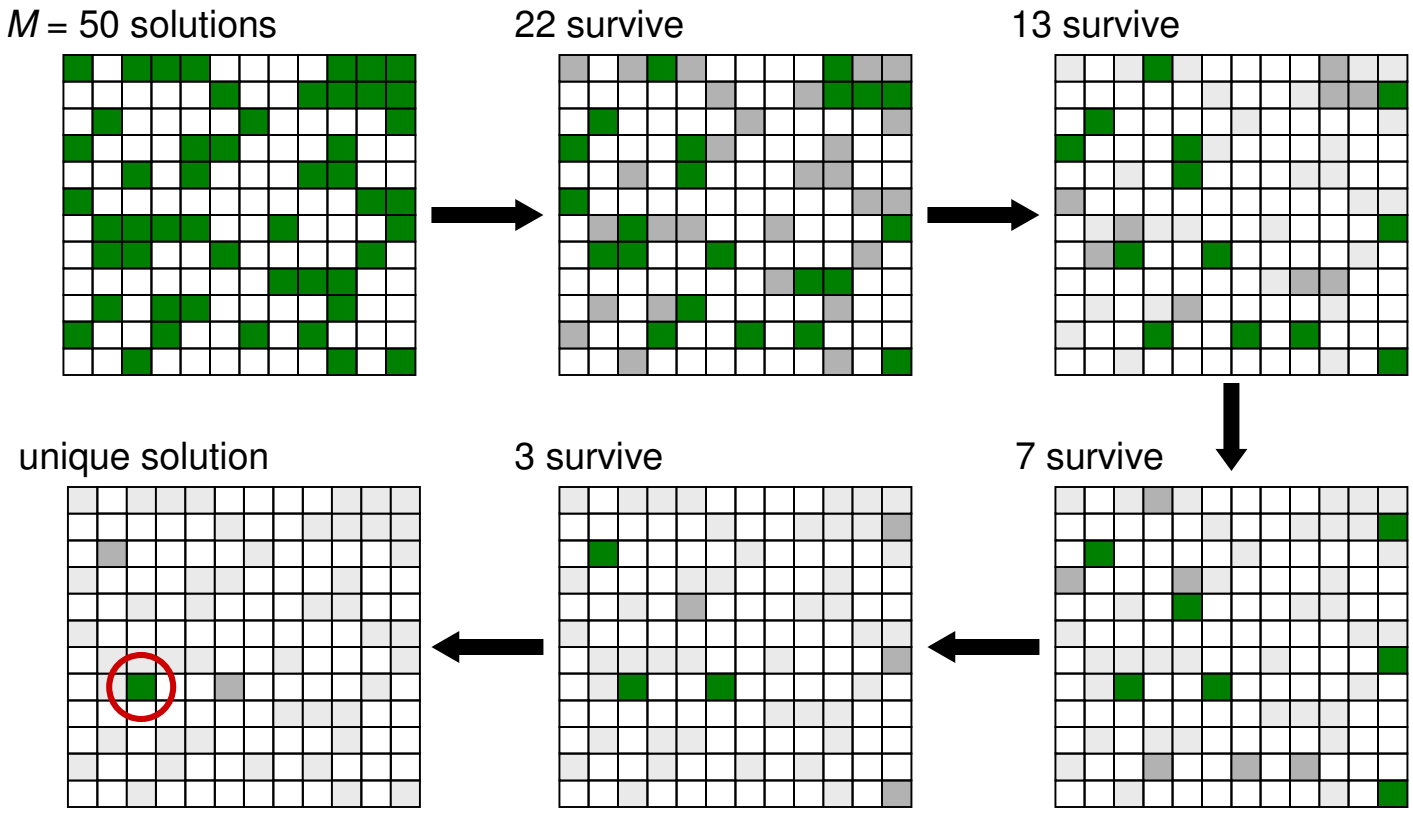
"decimation"

This provably yields the true count on average!

- Even when an unbalanced row/column is picked accidentally for the split, e.g. even when samples are biased or insufficiently many
- Provides probabilistic correctness guarantees on the estimated count
- Surprisingly good in practice, using SampleSat as the sampler

The Desired Effect of Decimation

If each setting cut the solution space roughly in half, would get down to a unique solution in roughly $\log_2$ #F steps!



Lower Bounds by Decimation

- Idea borrowed from SampleCount:
  - Select a **balanced variable** (appears same number of times positively and negatively in solutions)
  - Fix it to a **random** va
  - Recursively count th
  - **Scale up** the result

Difference from SampleCount: Choose most balanced variables efficiently using **Belief Propagation**

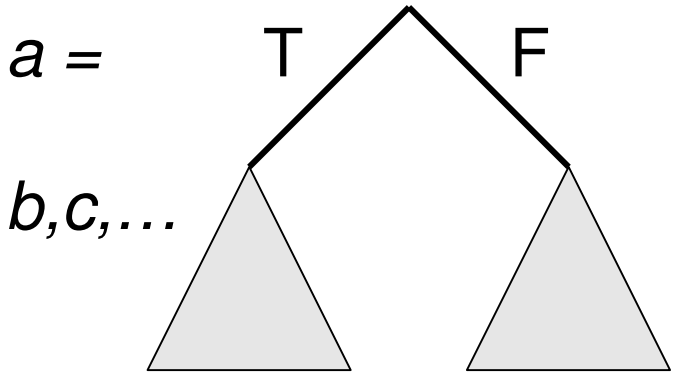
- Resulting count, $\#F_{dec}$, is thus a **random variable**
  - Key proposition: $\mathbf{E}[\#F_{dec}] = \#F$ (next slide)
  - Variance limits how well we can estimate $\mathbf{E}[\#F_{dec}]$
    - Balanced variables ⇒ smaller variance ⇒ tighter bounds
    - But method is robust to inaccuracies

Why Does This Work?

Proposition: $\mathbf{E}[\#F_{dec}] = \#F$

where $\#F_{dec}$ is a count estimate
and $\#F$ is true solution count of F

M=full count, p=(unknown) fraction in left subtree



Chance:	50%		50%
Count:	pM		(1-p)M
Scaling:	2		2
Average:	½.pM.2	+	½.(1-p)M.2 = M

What if there aren't any Balanced Vars?

- I.e. for every variable, p is far from 1/2
  - Or perhaps p is not estimated well

- **E**[#F_{dec}] still equals #F
  - Thus, the method is robust

- Can we do better if we have confidence in the estimate of p?
  - Yes! Can *reduce variance* by using biased coin: Pr[heads] = q ≠1/2
  - Scale up by 1/q or 1/(1-q)

  - Average = q.pM.(1/q) + (1-q).(1-p)M.(1/(1-q)) = M

decimated count = M
when q = p

From $\mathbf{E}[\#F_{dec}]$ to Lower Bound on #F

- Simple but useful application of Markov's Inequality

$$\Pr[\#F_{dec} \geq c\ \mathbf{E}[\#F]] \leq 1/c$$

$$\text{i.e. } \Pr[\#F_{dec} \geq c\ \#F] \leq 1/c$$

Would work even if $\mathbf{E}[\#F_{dec}] \leq \#F$

Would *not* work if $\mathbf{E}[\#F_{dec}] \geq \#F$

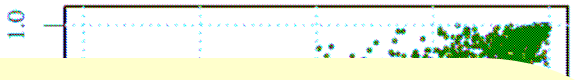
- will run into this soon!

Theorem: $\#F_{lower} = (\#F_{dec} / c)$ is a correct lower bound on #F with probability at least (1-1/c)

Adding Belief Propagation

- Identifying balanced variables by sampling is slow
  - Requires finding ~50 solutions per variable

- Replace sampling by **Belief Propagation** (BPCount)
  - Message-passing algorithm that (tries to) compute **marginal probabilities** of variables in graphical models (still a #P-complete problem!)
  - Balanced variables = those with marginals 0.5
  - **Advantage:** quick
  - **Drawbacks:** hard to make it work for many problems. Not accurate.

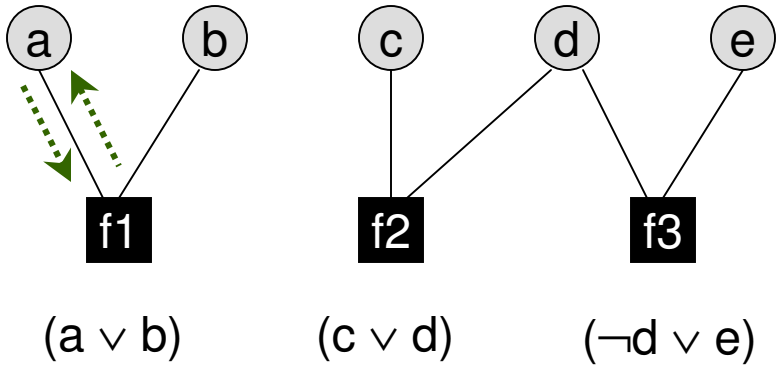


- **Quick:** big plus! We need it many times.
- **Hard to make work:** Ok, can use a "damped" version
- **Not accurate:** Ok, the underlying method is robust

Aside: Belief Propagation for CSPs

- A general iterative message-passing algorithm to compute marginal probabilities over "graphical models"
- Based on fixed-point computations of recursive equations
  - For CSPs, marginal probabilities = solution densities
  - Convert F into a two-layer Bayesian network

Iterative message passing



variables of F
⇒ variable nodes

constraints of F
⇒ function nodes

Experiments: BPCount

Intelligent Information Systems Institute

Always very fast!

Sometimes better

Sometimes not

Instance	# of vars	True Count (if known)	SampleCount (99% confidence) LWR-bound	SampleCount (99% confidence) Time	**BPCount** (99% confidence) LWR-bound	**BPCount** (99% confidence) Time
CIRCUIT SYNTH.						
2bitmax_6	252	2.1×10^{29}	$\geq 2.4 \times 10^{28}$	20 hrs	$\geq \mathbf{2.8 \times 10^{28}}$	5 hrs
RANDOM k-CNF						
wff-3-3.5	150	1.4×10^{14}	$\geq 1.6 \times 10^{13}$	3 min	$\geq \mathbf{1.6 \times 10^{11}}$	5 sec
wff-3-1.5	100	1.8×10^{21}	$\geq 1.6 \times 10^{20}$	3 min	$\geq \mathbf{1.0 \times 10^{20}}$	1 sec
wff-4-5.0	100	—	$\geq 8.0 \times 10^{15}$	2 min	$\geq \mathbf{2.0 \times 10^{15}}$	2 sec
LATIN SQUARE						
ls8-norm	301	5.4×10^{11}	$\geq 2.1 \times 10^{10}$	18 min	$\geq \mathbf{1.9 \times 10^{10}}$	12 sec
ls9-norm	456	3.8×10^{17}	$\geq 1.4 \times 10^{17}$	32 min	$\geq \mathbf{1.0 \times 10^{16}}$	30 sec
ls10-norm	657	7.6×10^{24}	$\geq 2.7 \times 10^{23}$	54 min	$\geq \mathbf{1.0 \times 10^{23}}$	32 sec
ls11-norm	910	5.4×10^{33}	$\geq 1.3 \times 10^{30}$	50 min	$\geq \mathbf{6.4 \times 10^{28}}$	1 min
ls12-norm	1221	—	$\geq 4.0 \times 10^{37}$	58 min	$\geq \mathbf{2.0 \times 10^{41}}$	70 sec
ls13-norm	1596	—	$\geq 3.0 \times 10^{49}$	47 min	$\geq \mathbf{4.0 \times 10^{53}}$	6 min
ls14-norm	2041	—	$\geq 2.0 \times 10^{60}$	31 min	$\geq \mathbf{1.0 \times 10^{67}}$	4 min
LANGFORD PROBS.						
lang-2-12	576	1.0×10^{5}	$\geq 4.5 \times 10^{3}$	32 min	$\geq \mathbf{3.8 \times 10^{3}}$	50 sec
lang-2-15	1024	3.0×10^{7}	$\geq 1.0 \times 10^{6}$	60 min	$\geq \mathbf{8.5 \times 10^{5}}$	1 min
lang-2-16	1024	3.2×10^{8}	$\geq 1.0 \times 10^{6}$	55 min	$\geq \mathbf{2.9 \times 10^{6}}$	1 min
lang-2-19	1444	2.1×10^{11}	$\geq 2.5 \times 10^{8}$	63 min	$\geq \mathbf{4.7 \times 10^{7}}$	3 min
lang-2-20	1600	2.6×10^{12}	$\geq 3.8 \times 10^{9}$	51 min	$\geq \mathbf{2.1 \times 10^{8}}$	10 min
lang-2-23	2116	3.7×10^{15}	$\geq 1.6 \times 10^{13}$	55 min	$\geq \mathbf{1.8 \times 10^{12}}$	15 min
lang-2-24	2304	—	$\geq 1.1 \times 10^{15}$	80 min	$\geq \mathbf{8.9 \times 10^{13}}$	18 min

CORNELL UNIVERSITY

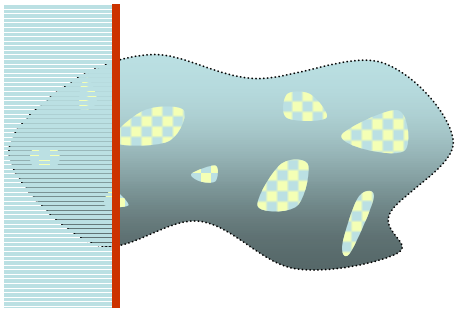
Talk Outline

- Upper bound estimates
  - Fast, using multiple runs of backtrack search solvers
  - Statistical correctness guarantees
  - Experiments

Why is Upper Bounding "Harder"?

Solution spaces are often relatively sparse

compared to the raw search space



E.g. consider F with 1000 variables and 2^{200} solutions

- Suppose we have explored 1/16 of the search space

- Likely number of solutions encountered = $2^{200}/16 = 2^{196}$
- Fairly good lower bound: $\#F \geq 2^{196}$

- Likely number of non-solutions encountered = $(2^{1000}-2^{200})/16 > 2^{995}$
- Very bad naïve upper bound: $\#F \leq 2^{1000} - 2^{995} \approx 2^{999}$

DPLL Search as a Random Process

- DPLL-style backtrack search involves:
  - Choose a variable to fix
  - Select its truth value (True or False)
  - If unsatisfied constraint, revise some previous decision (backtrack)

- Some of the decisions are randomized ⇒ no. of vars, d, fixed on a path to a solution is a **random variable**

- Certain properties of the search process make it possible to infer the **solution count** from *d*
  - MiniSat solver [Eén-Sörensson '05] can be used!
  - Powerful solver can be used to assess solution counts ⇒ **MiniCount**

Upper Bound using MiniCount

- Desired property: the solver does not specify *which truth value* a selected variable gets
  - truth value can be assigned at random
  - similar probabilistic reasoning as for BPCount can be used to get:

due to backtracks!

Proposition: $\mathbf{E}[\#F_{MiniCount}] \geq \#F$

where $\#F_{MiniCount} = 2^d$ is a count estimate
and $\#F$ = true solution count of F

- Inequality because can't correct for infeasible branches not visited
- No hope of retrieving lower bound using #FMiniCount

  Recall: E[#Fdec] equals #F $\Rightarrow$ probabilistic lower bounds

Goal: Estimate $\mathbf{E}[\#F_{\text{MiniCount}}]$

- Naïve estimation
  - Run process multiple times; obtain samples of $\#F_{\text{MiniCount}}$
  - Compute *sample average*

Not very robust

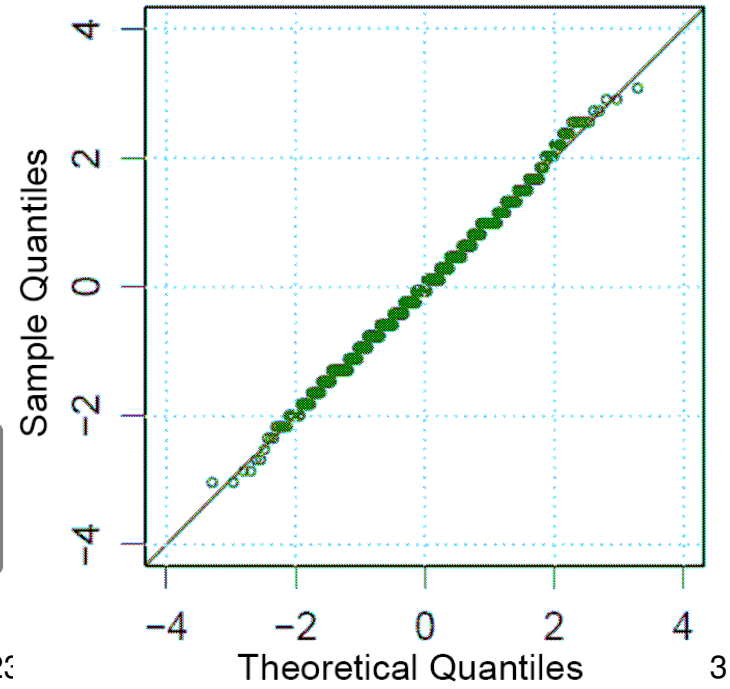
- $\#F_{\text{MiniCount}}$ might have high variance
  (fixed variables typically not balanced)
- Too slow: need many samples
- *Can we do better?*

$\Rightarrow$ Use ***statistics*** to bound the expectation

Bounding $\mathbf{E}[\#F_{\text{MiniCount}}]$

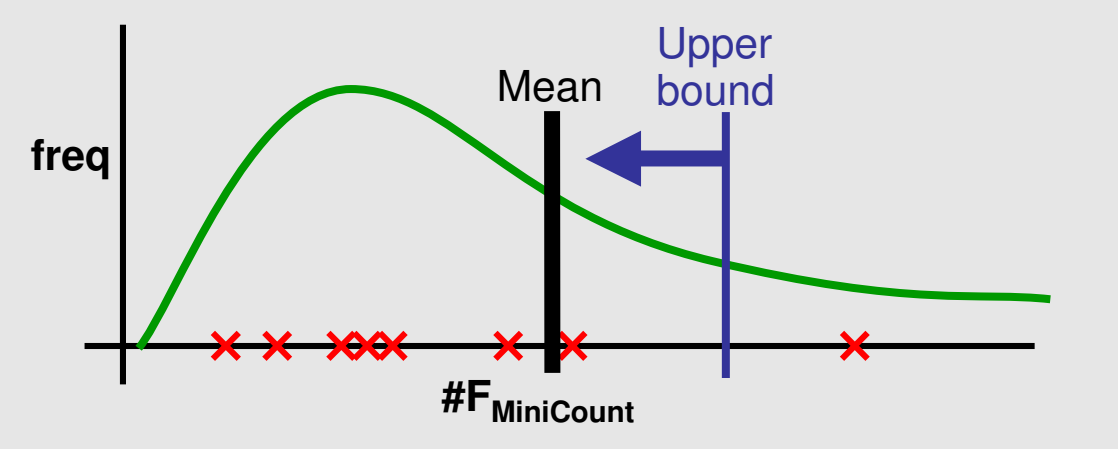
- Will use **confidence interval** for $\mathbf{E}[\#F_{\text{MiniCount}}]$
  - Need samples of the random variable **(OK)**
  - Need to know the shape of the distribution **(?)**

- Key empirical observation: *For many problems, the distribution of $\#F_{MiniCount}$ is approximately log-normal*
  - Can be checked using normality tests (like Shapiro-Wilk)
  - Can be assessed graphically (QQ-plots)



In either case, log-normality cannot be guaranteed, only "not disproved"

MiniCount in a Nutshell



- Generate independent samples from $\#F_{MiniCount}$ by independent runs of MiniCount
- Assess whether the samples come from a log-normal distribution, or a sufficiently similar one
- Calculate the (one-sided) confidence interval for $\mathbf{E}[\#F_{MiniCount}]$ and use it as the upper bound for #F

Sample average might easily underestimate the mean, due to the heavy tail of the distribution

MiniCount Results

Very fast!

Applicable for most problems

Sometimes not well applicable

Simple average often good, but not a bound

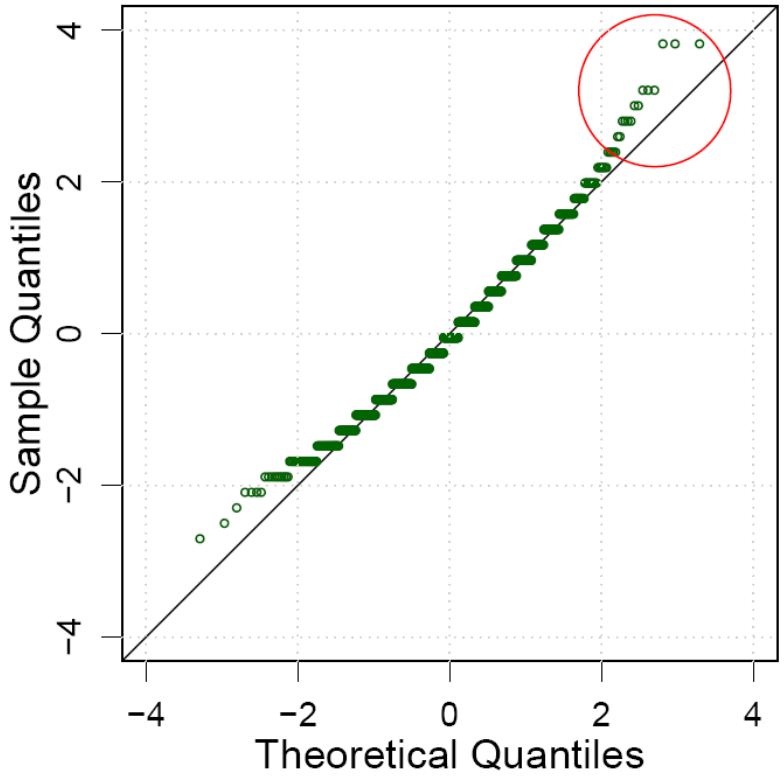
Instance	# of vars	True Count (if known)	MiniCount S-W Test	MiniCount (99% confidence) Average	MiniCount (99% confidence) UPR-bound	MiniCount Time
CIRCUIT SYNTH.						
2bitmax_6	252	2.1×10^{29}	✓	3.5×10^{30}	$\leq 4.3 \times 10^{32}$	2 sec
RANDOM k-CNF						
wff-3-3.5	150	1.4×10^{14}	✓	4.3×10^{14}	$\leq 6.7 \times 10^{15}$	2 sec
wff-3-1.5	100	1.8×10^{21}	✓	1.7×10^{21}	$\leq 4.8 \times 10^{22}$	2 sec
wff-4-5.0	100	—	✓	2.8×10^{16}	$\leq 5.7 \times 10^{28}$	2 sec
LATIN SQUARE						
ls8-norm	301	5.4×10^{11}	✓	6.4×10^{12}	$\leq 1.8 \times 10^{14}$	2 sec
ls9-norm	456	3.8×10^{17}	✓	6.9×10^{18}	$\leq 2.1 \times 10^{21}$	3 sec
ls10-norm	657	7.6×10^{24}	✓	4.3×10^{26}	$\leq 7.0 \times 10^{30}$	7 sec
ls11-norm	910	5.4×10^{33}	✓	1.7×10^{34}	$\leq 5.6 \times 10^{40}$	35 sec
ls12-norm	1221	—	✓	9.1×10^{44}	$\leq 3.6 \times 10^{52}$	4 min
ls13-norm	1596	—	✓	1.0×10^{54}	$\leq 8.6 \times 10^{69}$	17 min
ls14-norm	2041	—	✓	3.2×10^{63}	$\leq 1.3 \times 10^{86}$	2.5 hrs
LANGFORD PROBS.						
lang-2-12	576	1.0×10^{5}	×	5.2×10^{6}	$\leq 1.0 \times 10^{7}$	23 sec
lang-2-15	1024	3.0×10^{7}	✓	1.0×10^{8}	$\leq 9.0 \times 10^{8}$	8 sec
lang-2-16	1024	3.2×10^{8}	×	1.1×10^{10}	$\leq 1.1 \times 10^{10}$	7.5 sec
lang-2-19	1444	2.1×10^{11}	×	1.4×10^{10}	$\leq 6.7 \times 10^{12}$	30 sec
lang-2-20	1600	2.6×10^{12}	✓	1.4×10^{12}	$\leq 9.4 \times 10^{12}$	3 min
lang-2-23	2116	3.7×10^{15}	×	3.5×10^{12}	$\leq 1.4 \times 10^{13}$	20 min
lang-2-24	2304	—	×	2.7×10^{13}	$\leq 1.9 \times 10^{16}$	25 min

CORNELL UNIVERSITY

What Happens When *Not* Log-Normal?

Luckily, often easy to detect in practice

Langford problems:



Summary

- Fast lower bounds with probabilistic guarantees
  - Using an earlier randomized framework
  - Adding the power of Belief Propagation
    - Key improvement: speed!

- Fast upper bounds with statistical guarantees
  - Upper bounds turn out to be more difficult than lower bounds
  - First efficient way to tackle the problem, albeit not with any probabilistic guarantees