Model Counting: A New Strategy for Obtaining Good Bounds

Carla P. Gomes, Ashish Sabharwal, Bart Selman
Cornell University

AAAI Conference, 2006
Boston, MA

What is Model/Solution Counting?

- F : a Boolean formula
  - e.g. 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)

- **#SAT**: *How many* satisfying assignments does F have?
  - Generalizes SAT: Is F satisfiable at all?
  - With n variables, can have anywhere from 0 to 2^n satisfying assignments

Why Model Counting?

- Success of SAT solvers has had a tremendous impact
  - E.g. verification, planning, model checking, scheduling, …
  - Can easily model a variety of problems of interest as a Boolean formula, and use an off-the-shelf SAT solver
  - Rapidly growing technology:
    scales to 1,000,000+ variables and 5,000,000+ constraints

- Efficient model counting techniques will extend this to a whole new range of applications
  - Probabilistic reasoning
  - Multi-agent / adversarial reasoning (bounded)

    [Roth ‘96, Littman et. al. ‘01, Sang et. al. ‘04, Darwiche ‘05, Domingos ‘06]

The Challenge of Model Counting

- In theory
  - Model counting or #SAT is #P-complete
    (believed to be much harder than NP-complete problems)

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

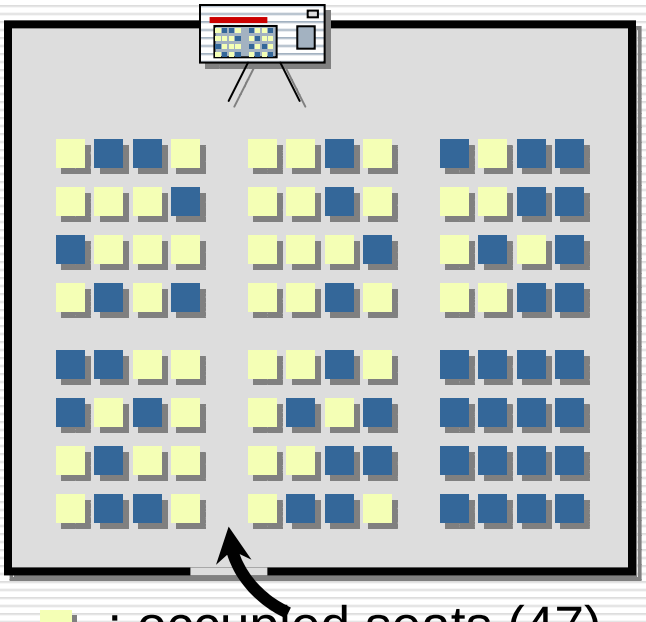
How Might One Count?

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)

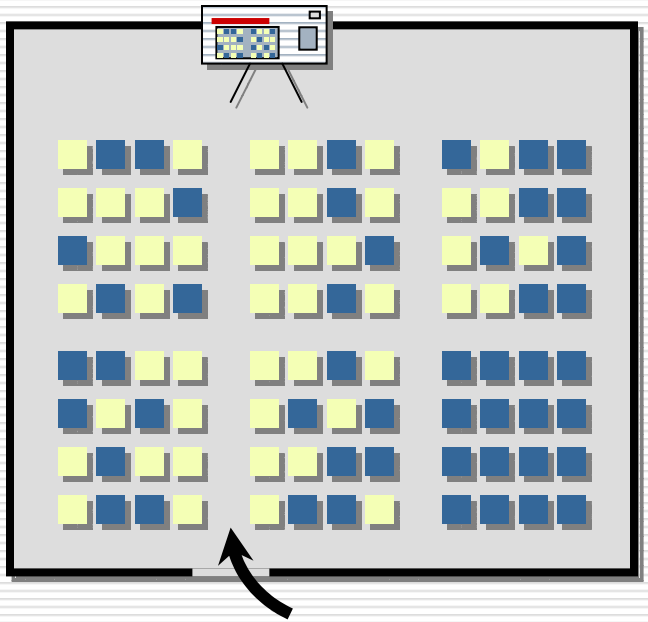
Previous approaches:

1. Brute force
2. Branch-and-bound
3. Estimation by sampling

This work:

A clever randomized strategy
using random XOR/parity constraints

#1: Brute-Force Counting



Idea:

- Go through every seat
- If occupied, increment counter

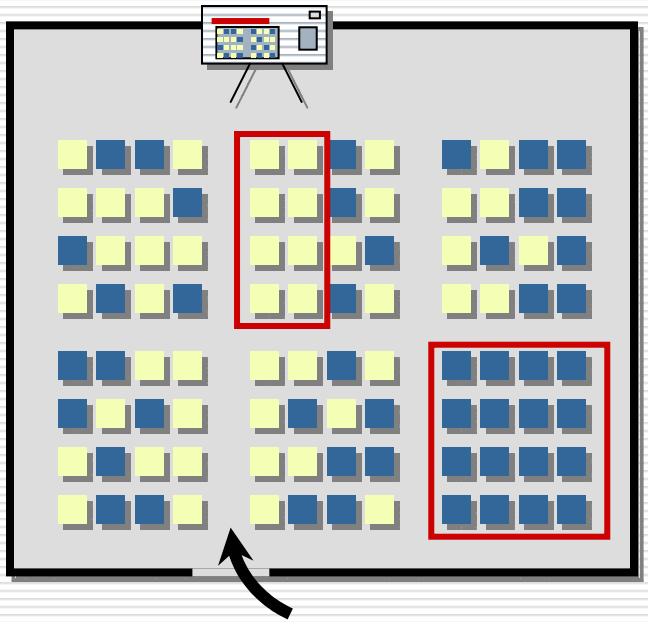
Advantage:

- Simplicity

Drawback:

- Scalability

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



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

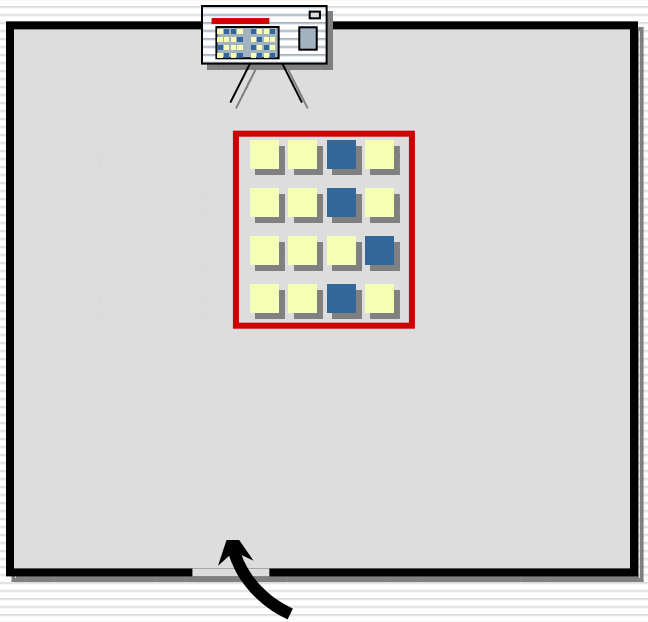
Advantage:

- Relatively faster

Drawback:

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

#3: Estimation By Sampling -- Naïve



Idea:

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

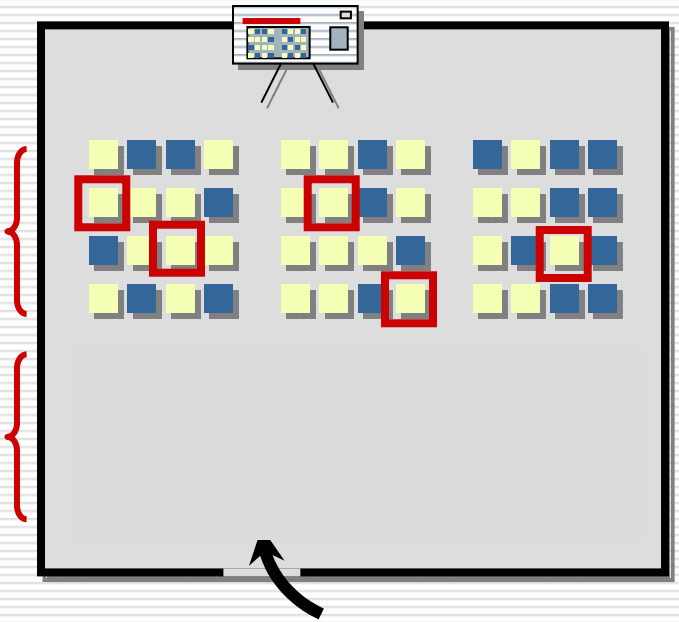
Advantage:

- 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

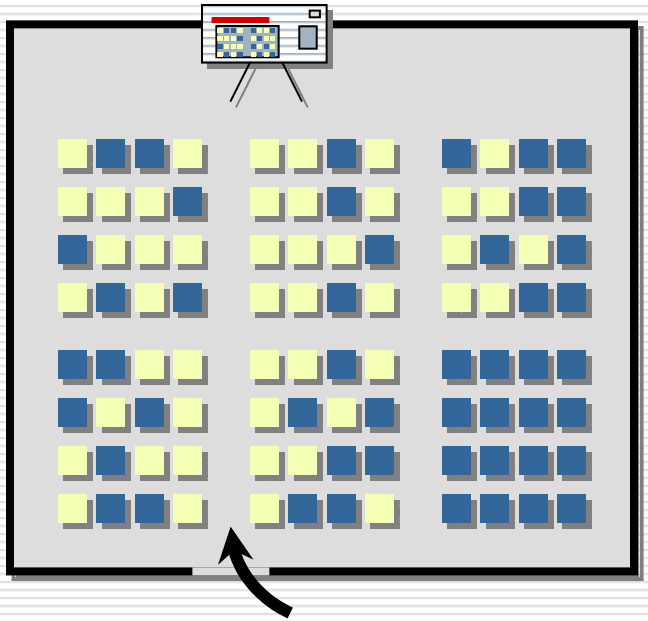
Advantage:

- Quite fast

Drawback:

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

Let's Try Something Different …



A Coin-Flipping Strategy

(Intuition)

Idea:

Everyone starts with a hand up

- Everyone tosses a coin
- If heads, keep hand up,
  if tails, bring hand down
- Repeat till only one hand is up

Return $2^{\#(\text{rounds})}$

Does this work?

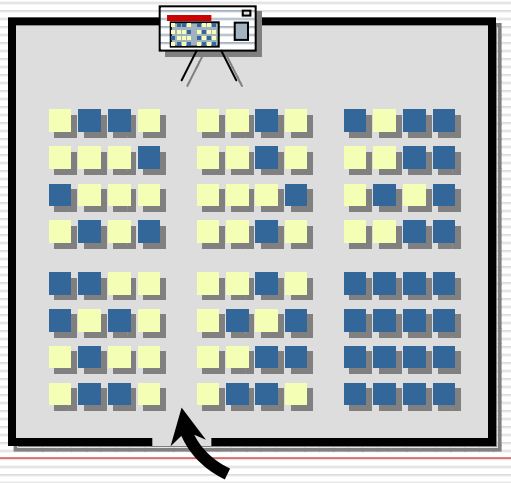
- On average, Yes!
- With M people present, need roughly $\log_2 M$ rounds for a unique hand to survive

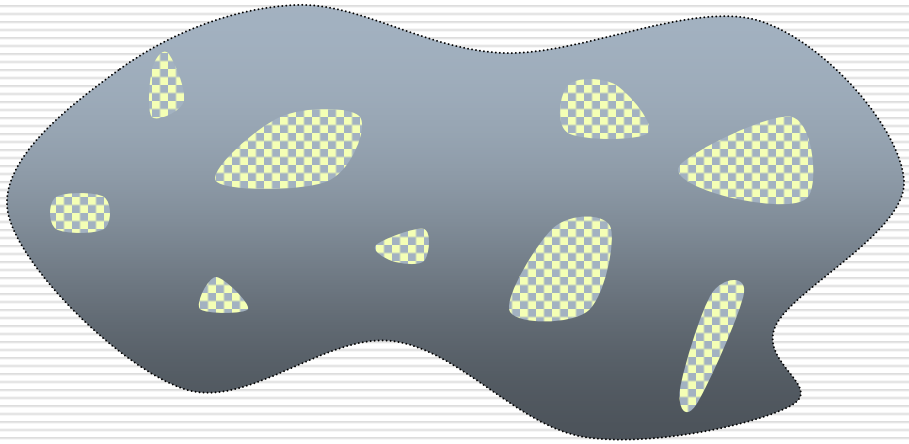
From Counting People to #SAT

Given a formula F over n variables,

- Auditorium : search space for F
- Seats : 2^n truth assignments
- Occupied seats : satisfying assignments

Bring hand down : add additional constraint eliminating that satisfying assignment





Making the Intuitive Idea Concrete

- How can we make each solution "flip" a coin?
  - Recall: solutions are implicitly "hidden" in the formula
  - Don't know anything about the solution space structure
- What if we don't hit a unique solution?
- How do we transform the average behavior into a robust method with provable correctness guarantees?

Somewhat surprisingly, all these issues can be resolved!

XOR Constraints to the Rescue

- Use XOR/parity constraints
  - $E.g.\quad a \oplus b \oplus c \oplus d = 1$
    (satisfied if an odd number of variables set to True)
  - Translates into a small set of CNF clauses
  - Used earlier in randomized reductions in Theo. CS
    [Valiant-Vazirani '86]

- Which XOR constraint X to use? Choose at random!

  Two crucial properties:
  - For every truth assignment A,
    Pr [A satisfies X] = 0.5

    Gives average behavior, some guarantees
  - For every two truth assignments A and B,
    "A satisfies X" and "B satisfies X" are independent

    Gives stronger guarantees

Obtaining Correctness Guarantees

- For formula F with M models/solutions, should ideally add $\log_2 M$ XOR constraints
- Instead, suppose we add $s = \log_2 M + 2$ constraints

slack factor

Fix a solution A.

Pr [A survives s XOR constraints] = $1/2^s = 1/(4M)$

$\Rightarrow$ Exp [number of surviving solutions] = $M / (4M) = 1/4$

$\Rightarrow$ Pr [some solution survives] $\leq 1/4$ (by Markov's Ineq)

Pr [F is satisfiable after s XOR constraints] $\leq 1/4$

Thm: If F is still satisfiable after s random XOR constraints, then F has $\geq 2^{s-2}$ solutions with prob. $\geq 3/4$

Boosting Correctness Guarantees

Simply repeat the whole process!

Say, we iterate 4 times independently with s constraints.

Pr [F is satisfiable in every iteration] $\leq 1/4^4 < 0.004$

Thm: If F is satisfiable after s random XOR constraints in each of 4 iterations, then F has at least 2^{s-2} solutions with prob. ≥ 0.996.

MBound Algorithm (simplified; by concrete usage example) :

Add k random XOR constrains and check for satisfiability using an off-the-shelf SAT solver. Repeat 4 times.

If satisfiable in all 4 cases, report 2^{k-2} as a lower bound on the model count with 99.6% confidence.

Key Features of MBound

- Can use any state-of-the-art SAT solver off the shelf
- Random XOR constraints independent of both the problem domain and the SAT solver used
- Adding XORs further constrains the problem
  - Can model count formulas that couldn't even be solved!
  - An effective way of "streamlining" [Gomes-Sellmann '04]
    → XOR streamlining
- Very high provable correctness guarantees on reported bounds on the model count
  - May be boosted simply by repetition

Making it Work in Practice

- Purely random XOR constraints are generally large
  - Not ideal for current SAT solvers
- In practice, we use relatively short XORs
  - Issue: Higher variation
  - Good news: lower bound correctness guarantees still hold
  - Better news: can get surprisingly good results in practice with extremely short XORs!

Experimental Results

Problem Instance	**Mbound** (99% confidence)		Relsat (exact counter)		ApproxCount (approx. counter)	
	Models	Time	Models	Time	Models	Time
Ramsey 1	≥ **1.2×10^{30}**	**2 hrs**	≥ 7.1×10^{8}	12 hrs	≈ 1.8×10^{19}	4 hrs
Ramsey 2	≥ **1.8×10^{19}**	**2 hrs**	≥ 1.9×10^{5}	12 hrs	≈ 7.7×10^{12}	5 hrs
Schur 1	≥ **2.8×10^{14}**	**2 hrs**	---	12 hrs	≈ 2.3×10^{11}	7 hrs
Schur 2 **	≥ **6.7×10^{7}**	**5 hrs**	---	12 hrs	---	12 hrs
ClqColor 1	≥ **2.1×10^{40}**	**3 min**	≥ 2.8×10^{26}	12 hrs	---	12 hrs
ClqColor 2	≥ **2.2×10^{46}**	**9 min**	≥ 2.3×10^{20}	12 hrs	---	12 hrs

** Instance cannot be solved by *any* state-of-the-art SAT solver

Summary and Future Directions

- Introduced XOR streamlining for model counting
  - can use any state-of-the-art SAT solver off the shelf
  - provides significantly better counts on challenging instances, including some that can't even be solved
  - Hybrid strategy: use exact counter after adding XORs
  - Upper bounds (extended theory using large XORs)

- Future Work
  - Uniform solution sampling from combinatorial spaces
  - Insights into solution space structure
  - From counting to probabilistic reasoning

Extra Slides

How Good are the Bounds?

- In theory, with enough computational resources, can provably get as close to the exact counts as desired.

- In practice, limited to relatively short XORs. However, can still get quite close to the exact counts!

Instance	Number of vars	Exact count	MBound	
			xor size	lowerbound
bitmax	252	21.0×10^{28}	9	$\geq 9.2 \times 10^{28}$
log_a	1719	26.0×10^{15}	36	$\geq 1.1 \times 10^{15}$
php 1	200	6.7×10^{11}	17	$\geq 1.3 \times 10^{11}$
php 2	300	20.0×10^{15}	20	$\geq 1.1 \times 10^{15}$