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. $F = (a \text{ or } b) \text{ and } (\text{not } (a \text{ and } (b \text{ or } c)))$
  - 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?

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)

**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

Framework used in DPLL-based systematic exact counters 

- e.g. Relsat [Bayardo-et-al ‘00], Cachet [Sang et. al. ‘04]
#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

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

Framework used in approximate counters like ApproxCount [Wei-Selman ‘05]
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. \( a \oplus b \oplus c \oplus d = 1 \)
  (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 \( A \),
  \( \Pr [ A \text{ satisfies } X ] = 0.5 \)
- For every two truth assignments \( A \) and \( B \),
  “\( A \) satisfies \( X \)” and “\( B \) satisfies \( X \)” are independent

Gives average behavior, some guarantees

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

Fix a solution $A$.

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

$\Rightarrow \exp [ \text{ number of surviving solutions } ] = M / (4M) = 1/4$

$\Rightarrow \Pr [ \text{some solution survives } ] \leq 1/4$ (by Markov’s Ineq)

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

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.

\[
\Pr \left[ F \text{ is satisfiable in every iteration } \right] \leq 1/4^4 < 0.004
\]

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. \( \geq 0.996 \).

**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

<table>
<thead>
<tr>
<th>Problem Instance</th>
<th>Mbound (99% confidence)</th>
<th>Relsat (exact counter)</th>
<th>ApproxCount (approx. counter)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Models</strong></td>
<td><strong>Time</strong></td>
<td><strong>Models</strong></td>
<td><strong>Time</strong></td>
</tr>
<tr>
<td>Ramsey 1</td>
<td>( \geq 1.2 \times 10^{30} )</td>
<td>2 hrs</td>
<td>( \geq 7.1 \times 10^{8} )</td>
</tr>
<tr>
<td>Ramsey 2</td>
<td>( \geq 1.8 \times 10^{19} )</td>
<td>2 hrs</td>
<td>( \geq 1.9 \times 10^{5} )</td>
</tr>
<tr>
<td>Schur 1</td>
<td>( \geq 2.8 \times 10^{14} )</td>
<td>2 hrs</td>
<td>---</td>
</tr>
<tr>
<td>Schur 2 **</td>
<td>( \geq 6.7 \times 10^{7} )</td>
<td>5 hrs</td>
<td>---</td>
</tr>
<tr>
<td>ClqColor 1</td>
<td>( \geq 2.1 \times 10^{40} )</td>
<td>3 min</td>
<td>( \geq 2.8 \times 10^{26} )</td>
</tr>
<tr>
<td>ClqColor 2</td>
<td>( \geq 2.2 \times 10^{46} )</td>
<td>9 min</td>
<td>( \geq 2.3 \times 10^{20} )</td>
</tr>
</tbody>
</table>

** 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!

<table>
<thead>
<tr>
<th>Instance</th>
<th>Number of vars</th>
<th>Exact count</th>
<th>xor size</th>
<th>lowerbound</th>
</tr>
</thead>
<tbody>
<tr>
<td>bitmask</td>
<td>252</td>
<td>$21.0 \times 10^{28}$</td>
<td>9</td>
<td>$\geq 9.2 \times 10^{28}$</td>
</tr>
<tr>
<td>log_a</td>
<td>1719</td>
<td>$26.0 \times 10^{15}$</td>
<td>36</td>
<td>$\geq 1.1 \times 10^{15}$</td>
</tr>
<tr>
<td>php 1</td>
<td>200</td>
<td>$6.7 \times 10^{11}$</td>
<td>17</td>
<td>$\geq 1.3 \times 10^{11}$</td>
</tr>
<tr>
<td>php 2</td>
<td>300</td>
<td>$20.0 \times 10^{15}$</td>
<td>20</td>
<td>$\geq 1.1 \times 10^{15}$</td>
</tr>
</tbody>
</table>