Model Counting for Probabilistic Reasoning

Beyond NP Workshop

Stefano Ermon
CS Department, Stanford
Combinatorial Search and Optimization

Progress in combinatorial search since the 1990s (SAT, SMT, MIP, CP, …): from 100 variables, 200 constraints (early 90s) to 1,000,000 variables and 5,000,000 constraints in 25 years

**SAT:** Given a formula $F$, does it have a satisfying assignment?

$$(x_1 \lor x_2 \lor \neg x_3) \land (\neg x_2 \lor \neg x_1) \land (\neg x_1 \lor x_3)$$

$x_1 = \text{True}$

$x_2 = \text{False}$

$x_3 = \text{True}$

Symbolic representation + combinatorial reasoning technology (e.g., SAT solvers) used in an enormous number of applications
Applications

- chip design
- protein folding
- timetabling
- logistics
- scheduling

Program synthesis

Package dependencies

Game playing

TimeTable

03:37 23:22

03:37
Key paradigm in AI:

Separate models from algorithms

What is the “right” modeling language?
Knowledge Representation

- Model is used to represent our **domain knowledge**
- Knowledge that is **deterministic**
  - “If there is rain, there are clouds”:
    Clouds OR \(\neg\) (Rain)
- Knowledge that includes **uncertainty**
  - “If there are clouds, there is a chance for rain”
- **Probabilistic** knowledge
  - “If there are clouds, the rain has probability 0.2”
    Probability \((\text{Rain}=\text{True} \mid \text{Clouds}=\text{True})=0.2\)

Probabilistic/statistical modeling useful in many domains: handles uncertainty, noise, ambiguities, model misspecifications, etc. **Whole new range of applications!**
Applications of Probabilistic Reasoning

- Bioinformatics
- Social sciences
- Robotics
- Ecology
- Personal assistants
- Machine Translation
- Semantic labeling
- Image classification

Translate into Russian “the spirit is willing, but the flesh is weak”

.. but, how do we represent probabilistic knowledge?
Graphical models

For any configuration (or state), defined by an assignment of values to the random variables, we can compute the weight/probability of that configuration.

Example: \( \Pr [\text{Rain}=T, \text{Sprinkler}=T, \text{Wet}=T] \propto 0.01 \times 0.2 \times 0.99 \)

Idea: knowledge encoded as soft dependencies/constraints among the variables (essentially equivalent to weighted SAT)
Typical query: What is the probability of an event? For example, 

\[
\Pr[\text{Wet}=\text{T}] = \sum_{x=\{\text{T},\text{F}\}} \sum_{y=\{\text{T},\text{F}\}} \Pr[\text{Rain}=x, \text{Sprinkler}=y, \text{Wet}=\text{T}]
\]

Involves \textbf{(weighted) model counting}:

- \textbf{Unweighted} model counting (hard constraints): 
  \[
  \Pr[\text{Wet}=\text{T}] = \frac{\text{(# SAT assignments with Wet=True)}}{\text{(# of SAT assignments)}}
  \]

- \textbf{Weighted} model counting (soft constraints): 
  \[
  \Pr[\text{Wet}=\text{T}] = \frac{\text{(weight of assignments with Wet=True)}}{\text{(weight of assignments)}}
  \]
Model/Solution Counting

Deterministic reasoning:

**SAT:** Given a formula $F$, does it have a satisfying assignment?

\[
(x_1 \lor x_2 \lor \neg x_3) \\
\land (\neg x_2 \lor \neg x_1) \\
\land (\neg x_1 \lor x_3)
\]

$x_1 = \text{True}$  $x_2 = \text{False}$  $x_3 = \text{True}$

Probabilistic reasoning:

**Counting (#-SAT):** *How many* satisfying assignments (=models) does a formula $F$ have?

\[
(x_1 \lor x_2 \lor \neg x_3) \\
\land (\neg x_2 \lor \neg x_1) \\
\land (\neg x_1 \lor x_3)
\]

\{x_1 = \text{True, x}_2 = \text{False, x}_3 = \text{True}\} \\
\cdots \\
\{x_1 = \text{False, x}_2 = \text{False, x}_3 = \text{False}\}
Outline

• Introduction and Motivation
  – Knowledge representation and reasoning
  – Probabilistic modeling and inference
  – Model counting and sampling

• Algorithmic approaches
  – Unweighted model counting
  – Weighted model counting

• Conclusions
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, …
The Challenge of Model Counting

- **In theory**
  - Counting how many satisfying assignments at least as hard as deciding if there exists at least one
  - Model counting is \#P-complete (believed to be 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?

Analogy: 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.)
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

: occupied seats (47) = satisfying assignments
: empty seats (49)
#1: Brute-Force Counting

Idea:
- Go through every seat
- If occupied, increment counter

Advantage:
- Simplicity, accuracy

Drawback:
- Scalability

: occupied seats (47)
: empty seats (49)
#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, exact

Drawback:
- Still “accounts for” every single person present: need extremely fine granularity
- Scalability

Framework used in DPLL-based systematic exact counters e.g. Cachet [Sang-et]

Approximate model counting?

See also compilation approaches [Darwiche et. al]
#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
Let’s Try Something Different …

A Distributed 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 no 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
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

- How do we transform the average behavior into a robust method with provable correctness guarantees?

Somewhat surprisingly, all these issues can be resolved!
Random parity constraints

• XOR/parity constraints:
  
  – Example: \( a \oplus b \oplus c \oplus d = 1 \) satisfied if an odd number of \( a, b, c, d \) are set to 1

Each solution satisfies this random constraint with probability \( \frac{1}{2} \)

Randomly generated parity constraint \( X \)

\[
x_1 \oplus x_3 \oplus x_4 \oplus x_7 \oplus x_{10} = 1
\]
Using XORs for Counting

Given a formula $F$

1. Add some XOR constraints to $F$ to get $F'$ (this eliminates some solutions of $F$)
2. Check whether $F'$ is satisfiable
3. Conclude “something” about the model count of $F$

Key difference from previous methods:
- The formula changes
- The search method stays the same (SAT solver)
The Desired Effect

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

$M = 50$ solutions

22 survive

13 survive

unique solution

3 survive

7 survive
What about weighted counting?

For any configuration (or state), defined by an assignment of values to the random variables, we can compute the weight/probability of that configuration.

Example: \( \Pr [\text{Rain}=T, \text{Sprinkler}=T, \text{Wet}=T] \propto 0.01 \times 0.2 \times 0.99 \)
Using XORs for Weighted Counting

Given a **weighted** formula $F$

1. Add some XOR constraints to $F$ to get $F'$ (this eliminates some solutions of $F$)
2. Check whether $F'$ is satisfiable  Find MAX-SAT assignment
3. Conclude “something” about the **weighted** model count of $F$

Key difference from previous methods:
- The formula changes
- The search method stays the same (**MAX-SAT, ILP, CP solvers**)
Main Theorem (stated informally):

With probability at least 1 - δ (e.g., 99.9%), WISH (Weighted-Sums-Hashing) computes a sum defined over $2^n$ configurations (probabilistic inference, #P-hard) with a relative error that can be made as small as desired, and it requires solving $\Theta(n \log n)$ optimization instances (NP-equivalent problems).
Implementations and experimental results

- Many implementations based on this idea (originated from theoretical work due to [Stockmeyer-83, Valiant-Vazirani-86]):
  - Mbound, XorSample [Gomes et al-2007]
  - WISH, PAWS [Ermon et al-2013]
  - ApproxMC, UniWit, UniGen [Chakraborty et al-2014]
  - Achilioptas et al at UAI-15 (error correcting codes)
  - Belle et al. at UAI-15 (SMT solvers)

- Fast because they leverage good SAT/MAX-SAT solvers!

- How hard are the “streamlined” formulas (with extra parity constraints)?
Sparse/ Low-density parity constraints

The role of \textit{sparse (low-density) parity constraints}

\begin{align*}
X &= 1 \rightarrow \text{length 1, large variance} \\
X \oplus Y &= 0 \rightarrow \text{length 2, variance?} \\
X \oplus Y \oplus Q &= 0 \rightarrow \text{length 3, variance?} \\
&\quad \vdots \\
X \oplus Y \oplus … \oplus Z &= 0 \rightarrow \text{length } n/2, \text{ small variance}
\end{align*}

Increasingly complex constraints 😞

Increasingly low variance 😊

The shorter the constraints are, the easier they are to reason about.
The longer the constraints are, the more accurate the counts are

Can short constraints actually be used?
Random coin flipping

- **Distributed coin flipping mechanism**
- Ideally, each configuration flips a coin **independently**
Pairwise Independent Hashing

- Issue: we cannot simulate so many **independent coin flips** (one for each possible variable assignment)
- Solution: each configuration flips a coin **pairwise independently**

- Any two coin flips are independent
- Three or more might not be independent

**Still works!** Pairwise independence guarantees that configurations do not cluster too much in single bucket.

Can be simulated using **random parity constraints**: simply add each variable with probability $\frac{1}{2}$.

``Long” parity constraints are difficult to deal with!
Average Universal Hashing

• New class of **average universal hash functions** (coin flips generation mechanism) [ICML-14]

• Pairs of coin flips are NOT guaranteed to be independent anymore

• Key idea: Look at large enough sets. If we look at all pairs, **on average** they are “independent enough”. Main result:
  1. These coin flips are good enough for probabilistic inference (applies to all previous techniques based on hashing; **same theoretical guarantees**)
  2. Can be implemented with **short** parity constraints
Main Theorem (stated informally): [AAAI-16, Sunday]

For large enough $n$ (= number of variables),
- Parity constraints of length $\log(n)$ are sufficient.
- Parity constraints of length $\log(n)$ are also necessary.

Proof borrows ideas from the theory of low-density parity check codes.

Short constraints are much easier to deal with in practice.

Can even use other constraints! [under submission]
Outline

• Introduction and Motivation
  – Knowledge representation and reasoning
  – Probabilistic modeling and inference
  – Model counting and sampling

• Algorithmic approaches
  – Unweighted model counting
  – Weighted model counting

• Conclusions
Conclusions

• SAT solvers had a major impact on AI and other CS fields
• However, there are many interesting problems in AI and Machine Learning are beyond NP
• **Model counting** is the prototypical problem for **probabilistic reasoning**
  – Key computational problem with a long history: early work dates back to the 50s.
  – Recent approaches: try to “reduce” to problems in NP so that we can leverage SAT/ solvers.
  – Provides nice theoretical guarantees, as opposed to traditional approaches (MCMC, variational)