Probabilistic Inference and Accuracy Guarantees

Stefano Ermon

CS Department, Stanford

Combinatorial Search and Optimization

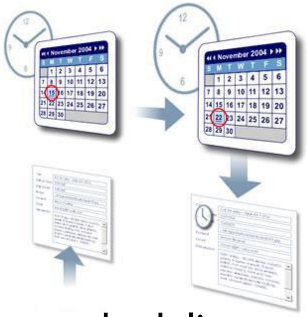
Progress in combinatorial search since the 1990s (SAT, SMT, MIP, CP, …): from 100 variables, 200 constraints (early 90s) to 1,000,000 variables and 5,000,000 constraints in 25 years

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

$$\begin{aligned} &(x_1 \vee x_2 \vee \neg x_3) \\ \wedge\ &(\neg x_2 \vee \neg x_1) \\ \wedge\ &(\neg x_1 \vee x_3) \end{aligned} \quad \longrightarrow \quad \begin{aligned} x_1&=\text{True} \\ x_2&=\text{False} \\ x_3&=\text{True} \end{aligned}$$

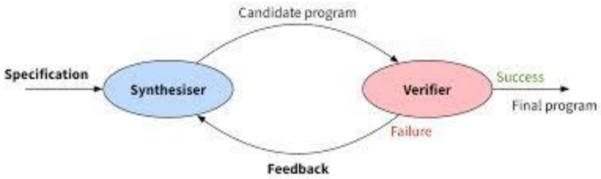
Symbolic representation + combinatorial reasoning technology (e.g., SAT solvers) used in an enormous number of applications

Applications



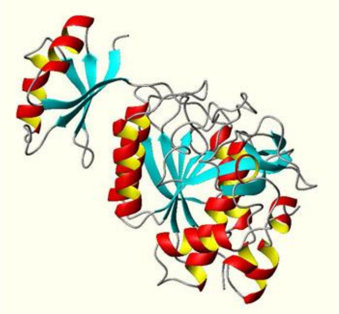
scheduling

logistics



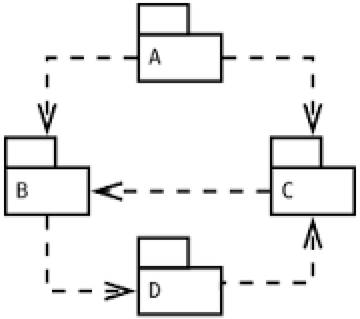
Program synthesis

chip design

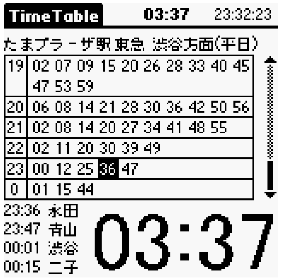


protein folding

network design



Package dependencies



TimeTable 03:37 23:32:23

たまプラーザ駅 東急 渋谷方面(平日)

19	02 07 09 15 20 26 28 33 40 45 47 53 59
20	06 08 14 21 28 30 36 42 50 56
21	02 08 14 20 27 34 41 48 55
22	02 11 20 30 39 49
23	00 12 25 36 47
0	01 15 44

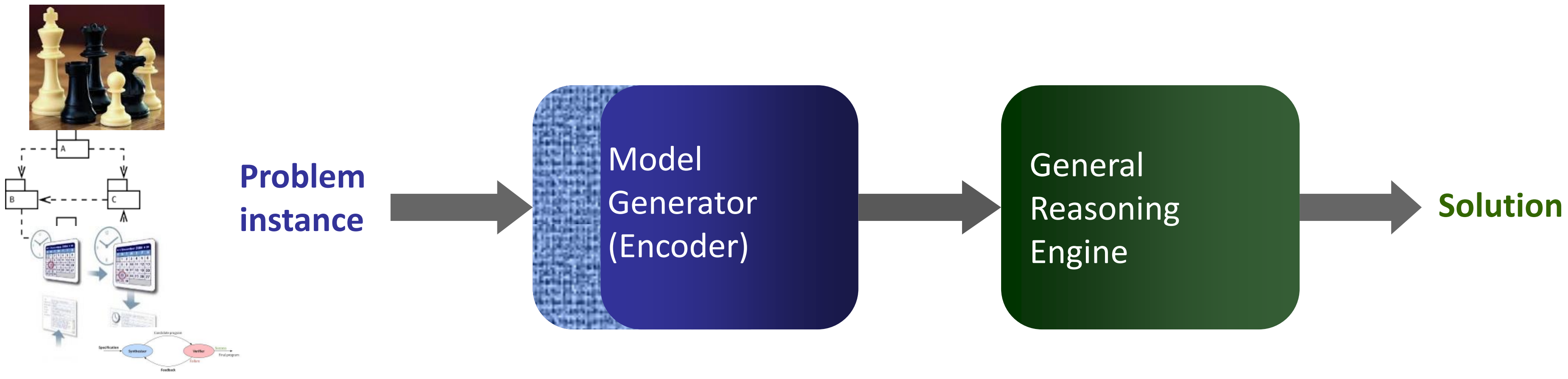
23:36 永田
23:47 青山
00:01 渋谷
00:15 二子

03:37

timetabling

Game playing

Problem Solving in AI



Problem instance

Model Generator (Encoder)

General Reasoning Engine

Solution

Domain-specific instances

General modeling language and algorithms

applicable to all domains
that can be expressed in the
modeling language

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”:

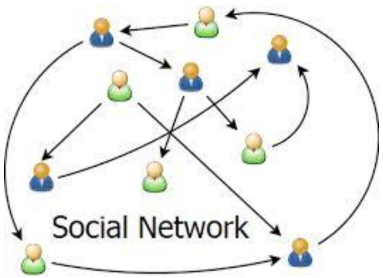
    Clouds OR ¬(Rain)
- Knowledge that includes **uncertainty**
  - “If there are clouds, there is a chance for rain”
- **Probabilistic** knowledge
  - “If there are clouds, the rain has probability 0.2”

    Probability (Rain=True | Clouds=True)=0.2

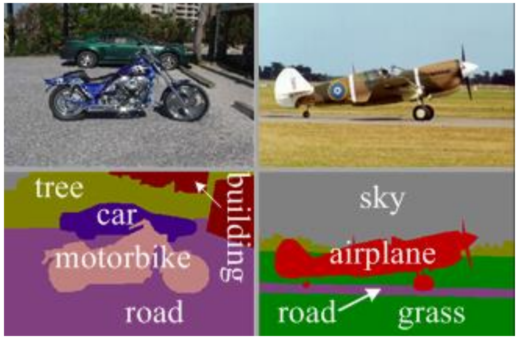
Probabilistic/statistical modeling useful in many domains: handles uncertainty, noise, ambiguities, model misspecifications, etc. Whole new range of applications!

Applications of Probabilistic Reasoning

bioinformatics

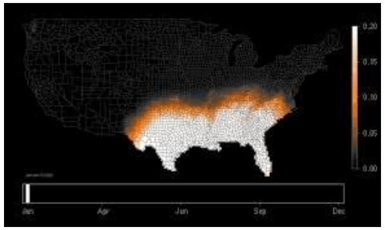


Social sciences

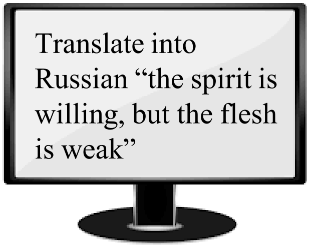


Semantic labeling

robotics



ecology



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

Machine Translation

What can I help you with?

Personal assistants

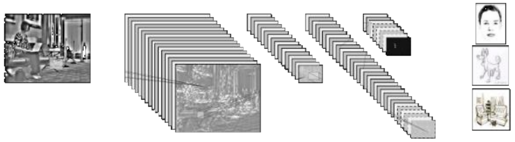


Image classification

.. but, how do we represent probabilistic knowledge?

Graphical models

“Rain OR Sprinkler => Wet Grass”

WET	
True	False
…	…
0.99	0.01
…	…

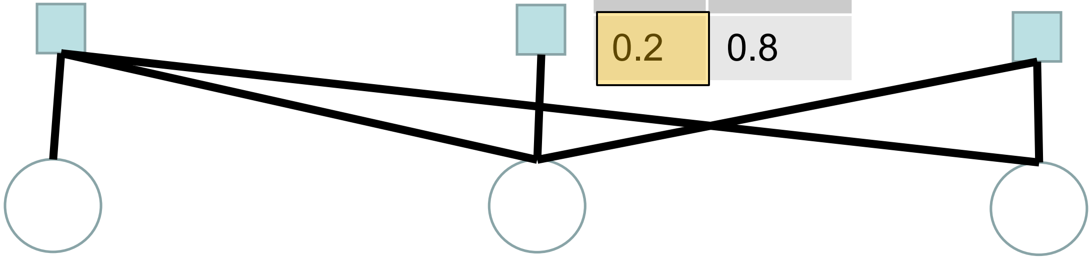
“Rain”

RAIN	
True	False
0.2	0.8

Factor

“Rain => ¬ Sprinkler”

		SPRINKLER	
		True	False
RAIN	True	F	T
	False	T	T



WET GRASS

RAIN

SPRINKLER

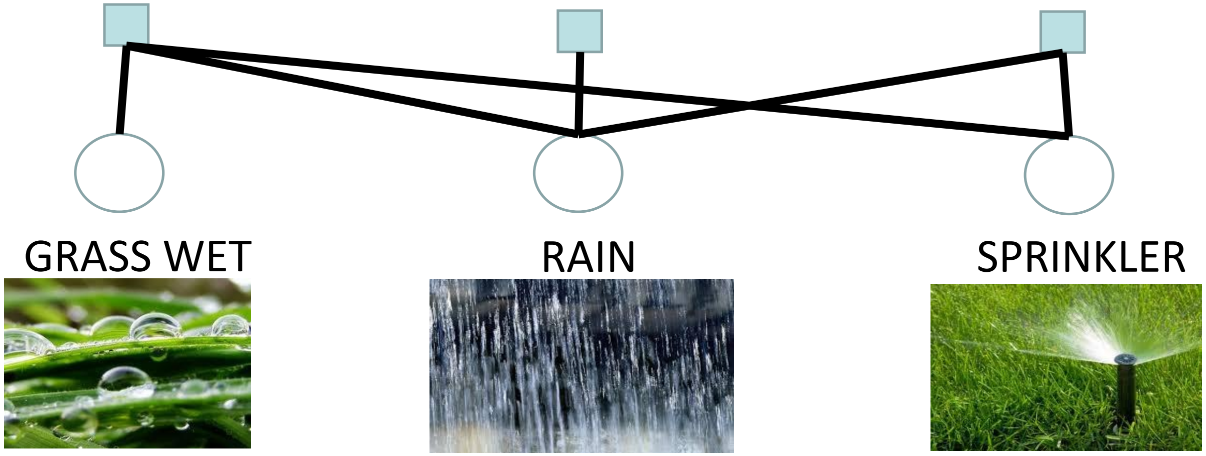
For any **configuration (**or **state)**, defined by an **assignment** of values to the random variables, we can compute the weight/probability of that configuration.

Example: Pr [Rain=T, Sprinkler=T, Wet=T] ∝ $\boxed{0.01} * \boxed{0.2} * \boxed{0.99}$

Idea: knowledge encoded as **soft dependencies/constraints** among the variables (essentially equivalent to weighted SAT)

How to do reasoning?

Probabilistic Reasoning



GRASS WET

RAIN

SPRINKLER

Typical query: What is the probability of an event? For example,

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

Involves (**weighted) model counting**:

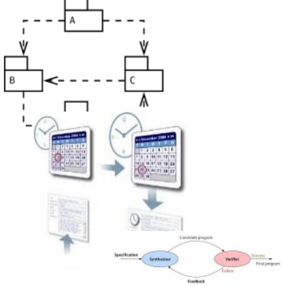
- **Unweighted** model counting (hard constraints):

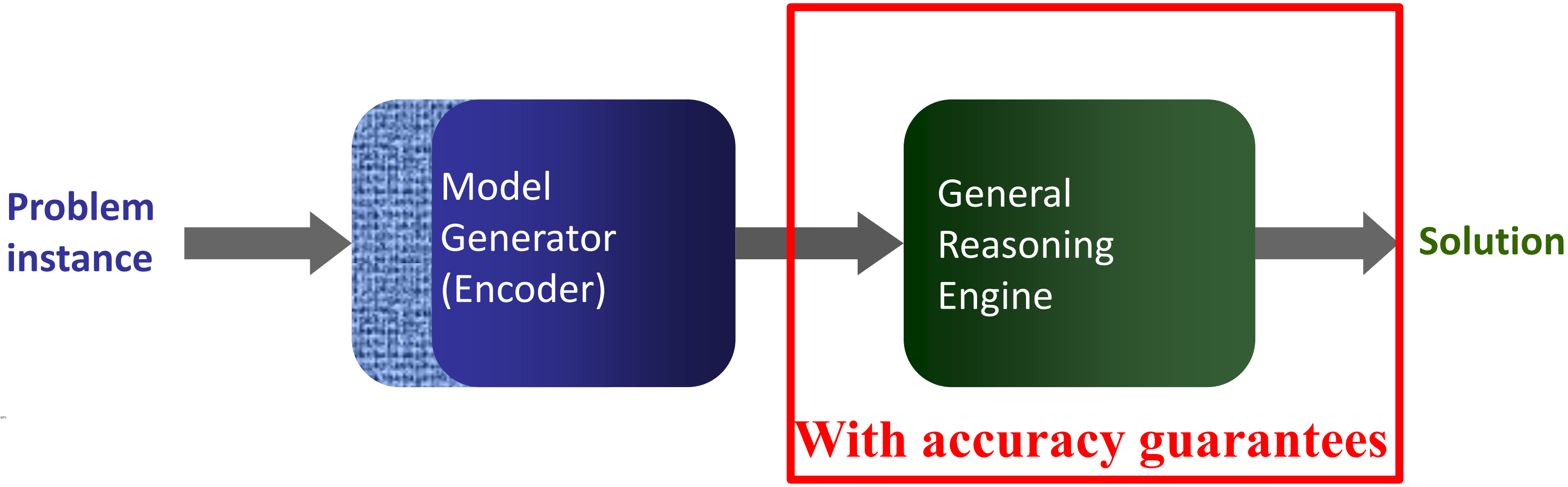
  Pr[Wet=T] = (# SAT assignments with Wet=True) / (# of SAT assignments)

- **Weighted** model counting (soft constraints):

  Pr[Wet=T] = (weight of assignments with Wet=True) / (weight of assignments)

Problem Solving in AI





For deterministic knowledge bases, **soundness** of the reasoning engine is crucial

- Lots of work on verifying “proofs”

For probabilistic reasoning, more emphasis on **scalability** than **accuracy guarantees**

- (Markov Chain) Monte Carlo sampling
- Variational methods
- Small errors might be OK, probability 0.546780 vs. 0.546781.
- 0.01 vs. 0.96 typically NOT OK

Model/Solution Counting

Deterministic reasoning:

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

$$\begin{aligned} &(x_1 \lor x_2 \lor \neg x_3) \\ \land\, &(\neg\, x_2 \lor \neg x_1) \\ \land\, &(\neg\, x_1 \lor x_3) \end{aligned} \longrightarrow \begin{aligned} &x_1=\text{True} \\ &x_2=\text{False} \\ &x_3=\text{True} \end{aligned}$$

Probabilistic reasoning:

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

$$\begin{aligned} &(x_1 \lor x_2 \lor \neg x_3) \\ \land\, &(\neg\, x_2 \lor \neg x_1) \\ \land\, &(\neg\, x_1 \lor x_3) \end{aligned} \longrightarrow \begin{gathered} \{x_1=\text{True},\ x_2=\text{False},\ x_3=\text{True}\} \\ \dots \\ \{x_1=\text{False},\ x_2=\text{False},\ x_3=\text{False}\} \end{gathered}$$

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?

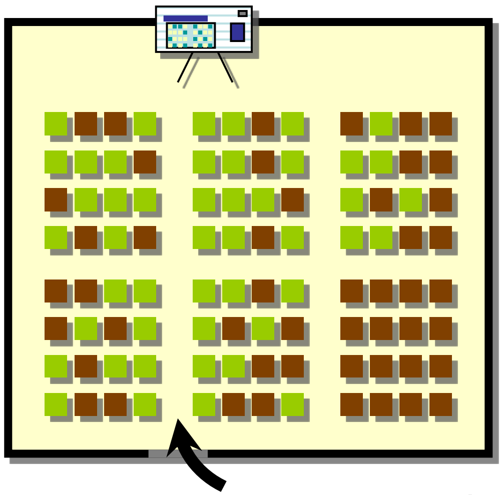
Problem characteristics:

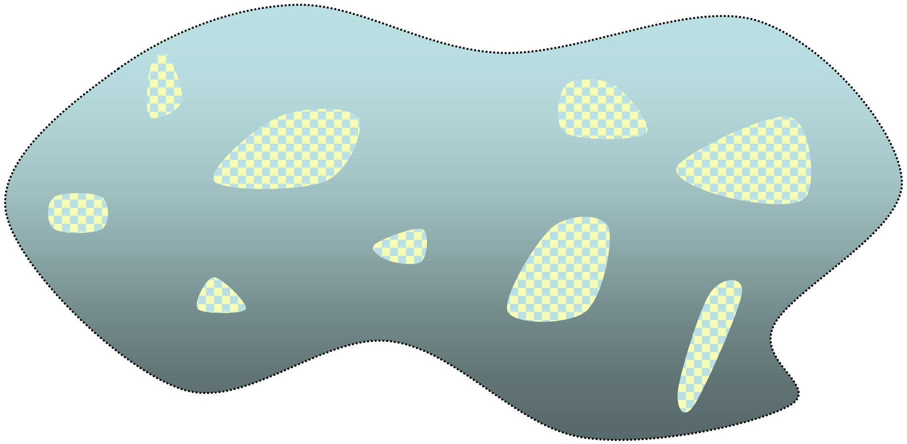
- Space naturally divided into rows, columns, sections, …
- Many seats empty
- Uneven distribution of people (e.g. more near door, aisles, front, etc.)

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

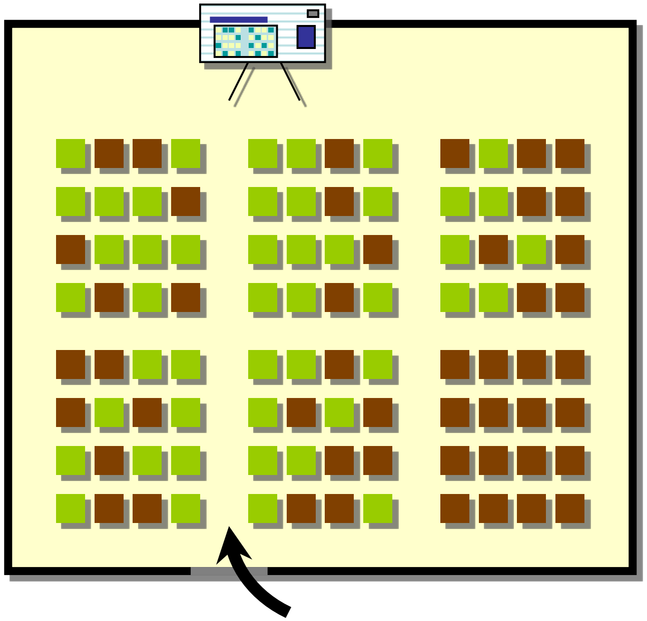




🟩 : occupied seats (47) = satisfying assignments

🟫 : empty seats (49)

#1: Brute-Force Counting



🟩 : occupied seats (47)

🟫 : empty seats (49)

Idea:

- Go through every seat
- If occupied, increment counter

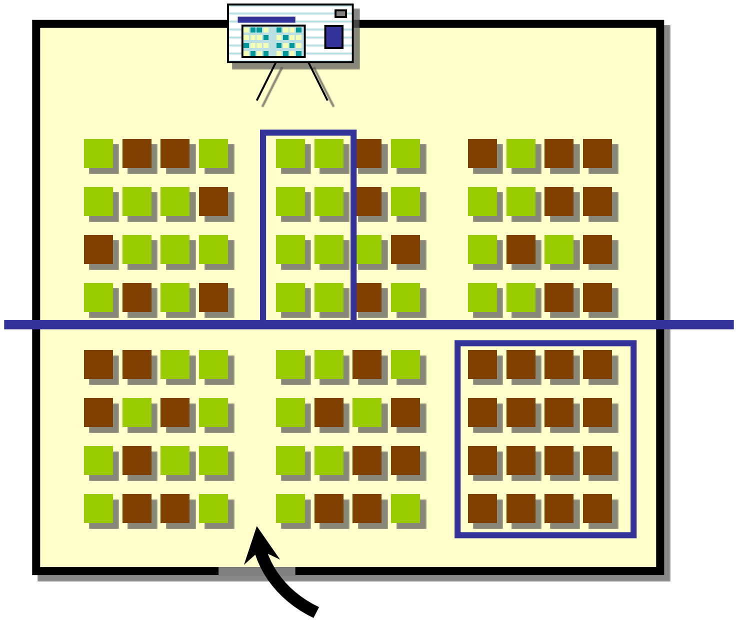
Advantage:

- Simplicity, accuracy

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, exact

Drawback:

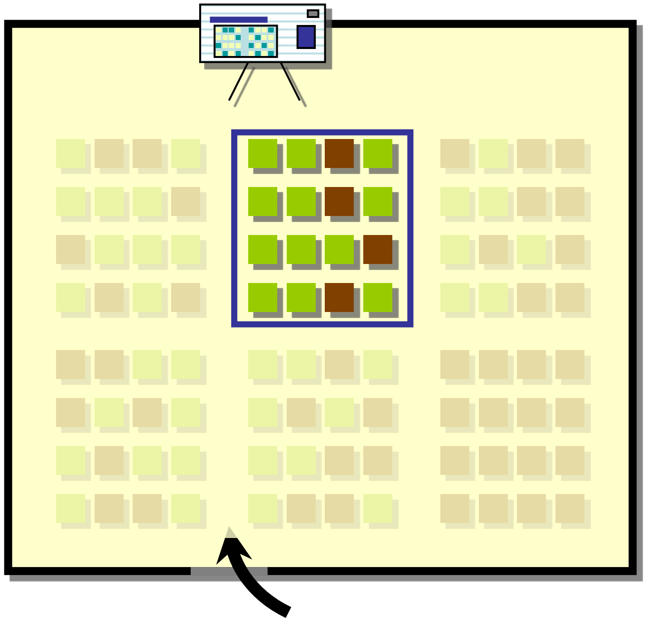
- Still "accounts for" every single person present: need extremely fine granularity
- Scalability

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:

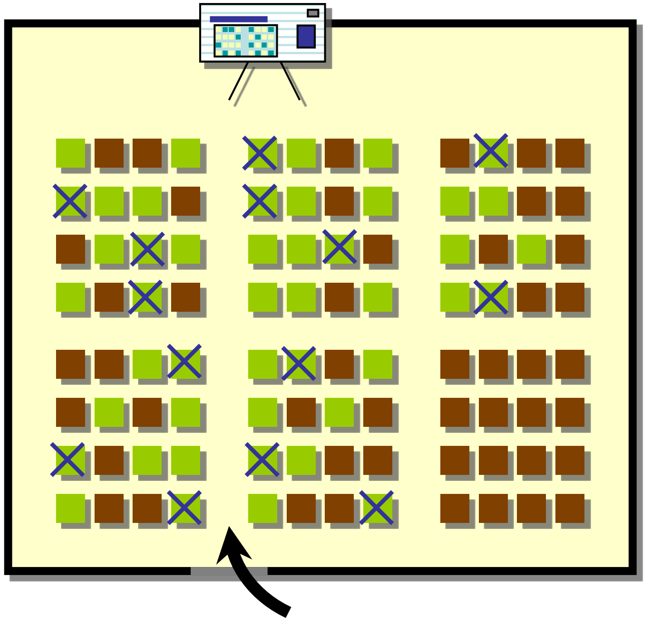
- Quite fast

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

No way of knowing if the answer is accurate or not!

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

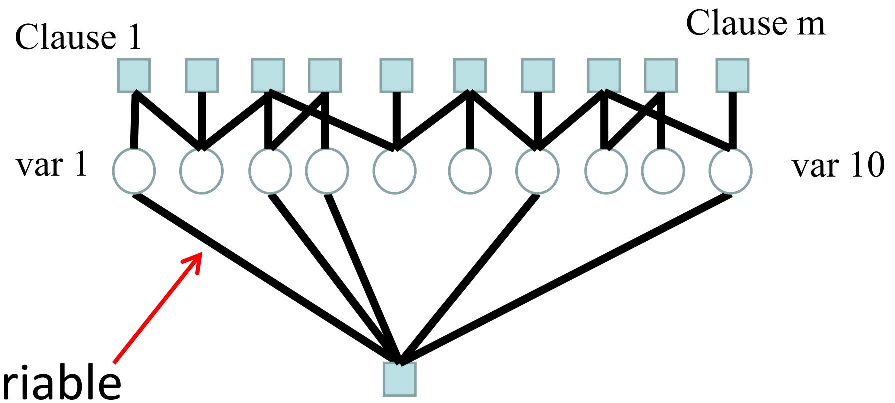
- How can we make each solution "flip" a coin?
  - Recall: solutions are implicitly "hidden" in the formula
  - Don't know anything about the solution space structure

- 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 variable added with prob. 0.5

Randomly generated parity constraint X

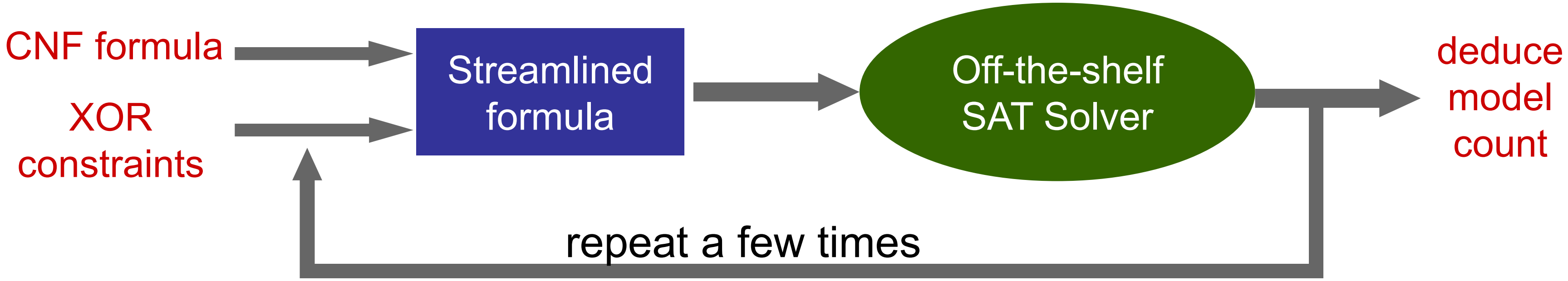
$$x_1 \oplus x_3 \oplus x_4 \oplus x_7 \oplus x_{10} = \mathbf{1}$$

- Each solution satisfies this random constraint with probability ½
- Pairwise independence: For every two configurations ***A*** and ***B***, "***A*** satisfies ***X***" and "***B*** satisfies ***X***" are independent events

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

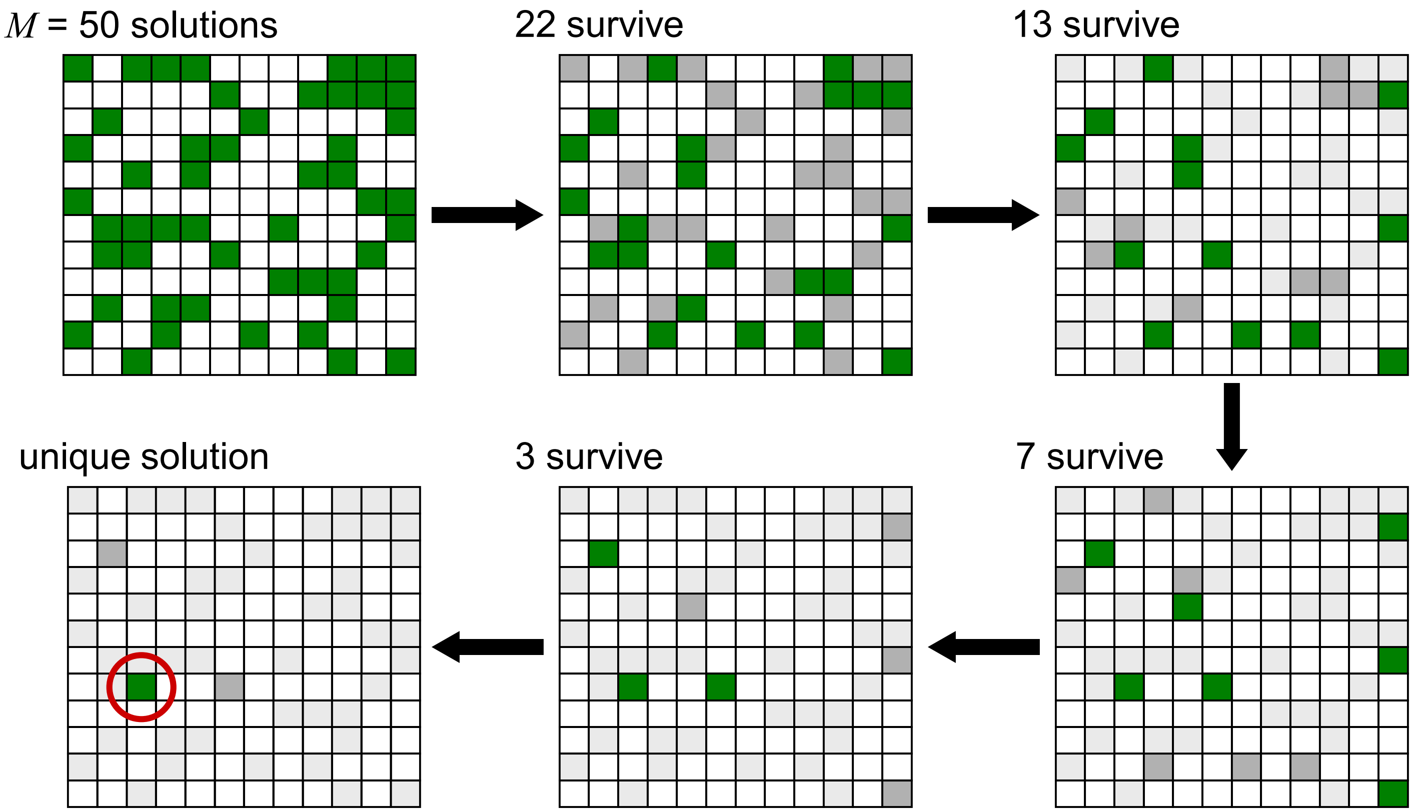


Key difference from previous methods:

- The formula changes
- The search method stays the same (**SAT solver**). If SAT solver is **sound**, so it this procedure!

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

7 survive

3 survive

unique solution

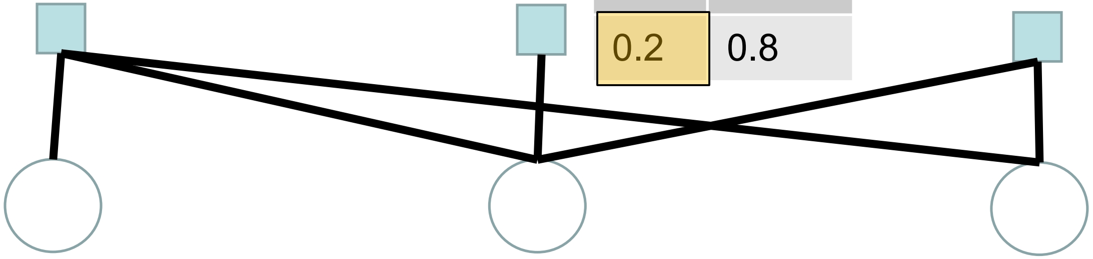
What about weighted counting?

“Rain OR Sprinkler => Wet Grass”

WET	
True	False
…	…
0.99	0.01
…	…

“Rain”

RAIN	
True	False
0.2	0.8



“Rain => ¬ Sprinkler”

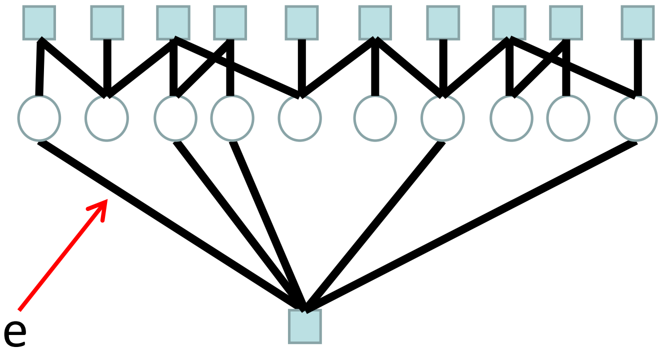
		SPRINKLER	
		True	False
RAIN	True	F	T
	False	T	T

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 [Rain=T, Sprinkler=T, Wet=T] ∝ 0.01 * 0.2 * 0.99

Hashing as a random projection

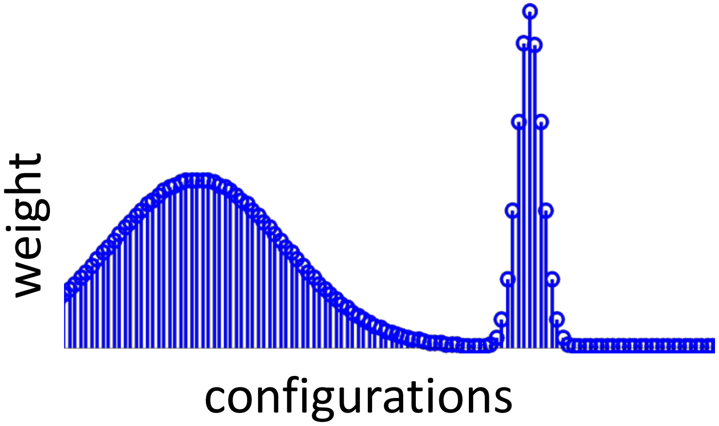
- XOR/parity constraints:
  - *Example:* $a \oplus b \oplus c \oplus d$ = 1 satisfied if an odd number of a,b,c,d are set to **1**



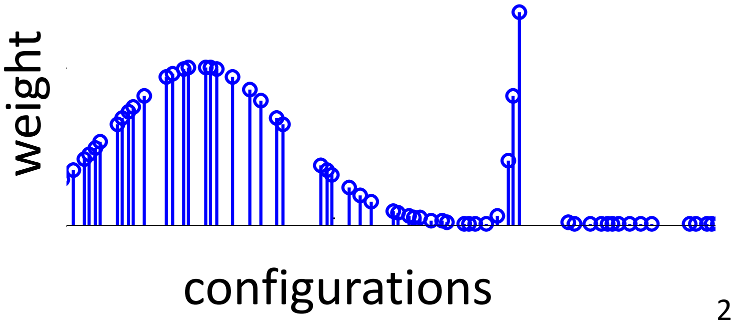
Each variable added with prob. 0.5

Randomly generated parity constraint X

$$x_1 \oplus x_3 \oplus x_4 \oplus x_7 \oplus x_{10} = \mathbf{1}$$



Random projection

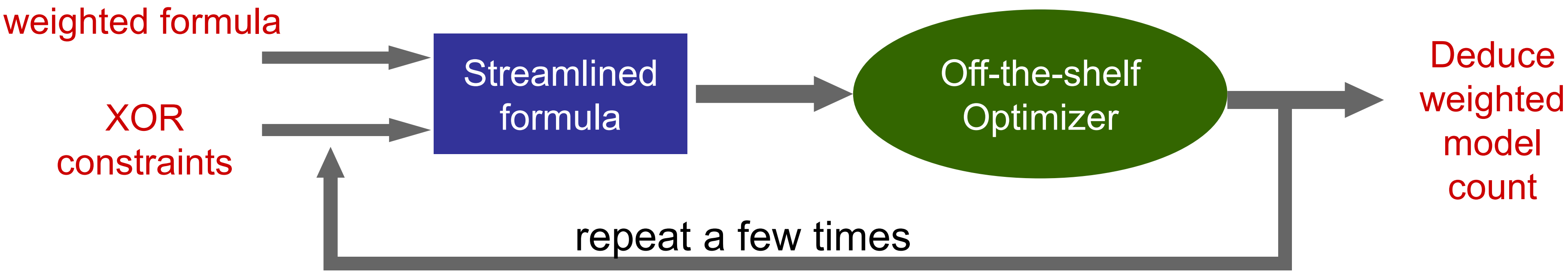


Set weight to zero if constraint is not satisfied

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-weight 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**)

Accuracy Guarantees

Main Theorem (stated informally):

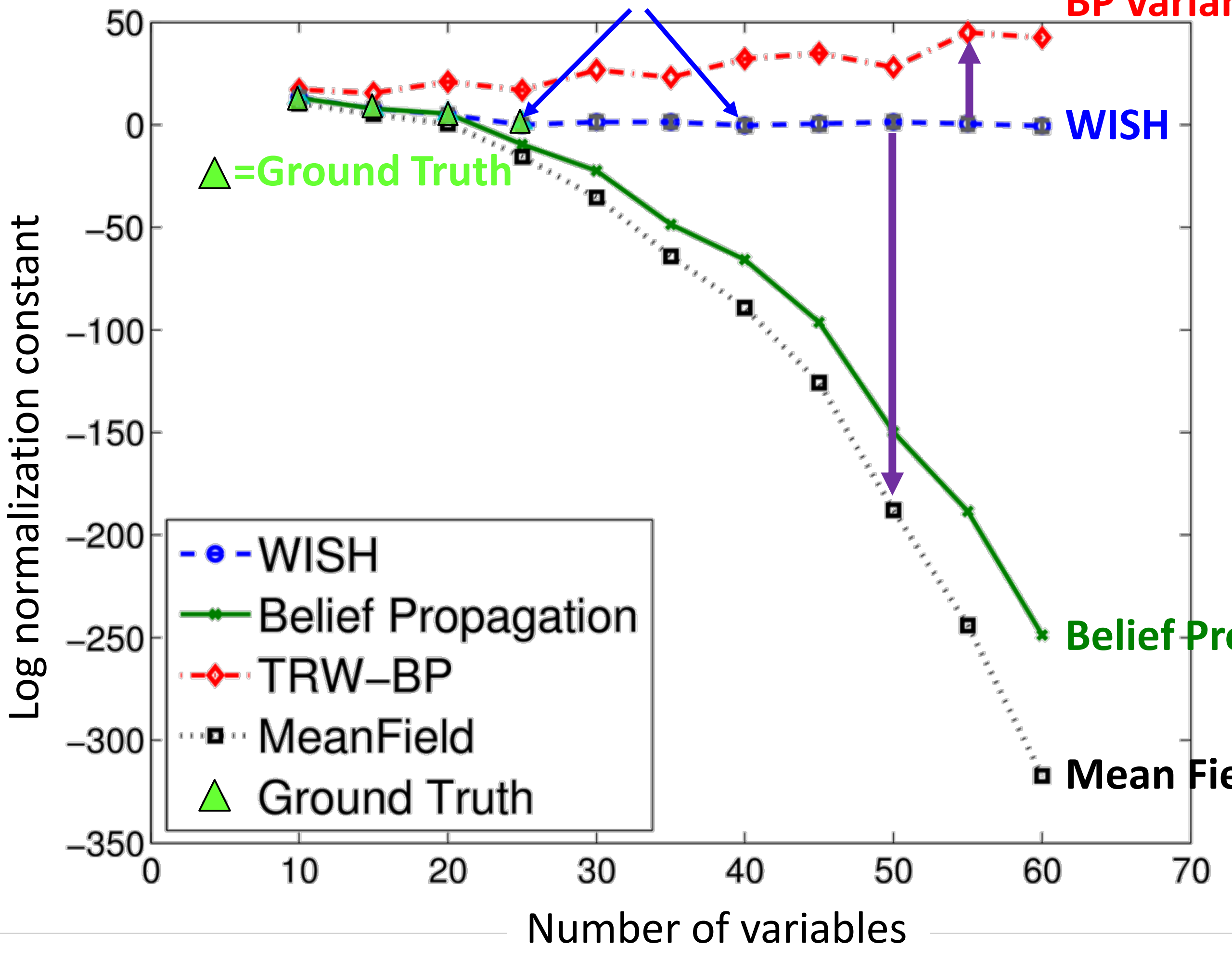
With probability at least 1- δ (e.g., 99.9%), WISH (**W**eighted-**S**ums-**H**ashing) computes a sum defined over 2^n configurations (**probabilistic inference**, #P-hard) with a relative error that can be made as small as desired, and it requires solving θ(n log n) **optimization instances** (NP-equivalent problems).

Key Features

- Strong accuracy guarantees
- Modular design: can plug in off-the-shelf optimization tools
  - **Branch and Bound / MaxSAT** (*Toulbar*)
  - **Integer Linear Programming** (*IBM CPLEX*)
- Decoding techniques for error correcting codes (LDPC)
- Even without optimality guarantees, bounds/approximations on the optimization translate to bounds/approximations on the weighted sum
- Straightforward **parallelization** (independent optimizations)
  - We experimented with >600 cores

Inference benchmark: Ising models from physics

Guaranteed relative error: ground truth is provably within this small error band ɪ (too small to see with this scale)



BP variant

WISH

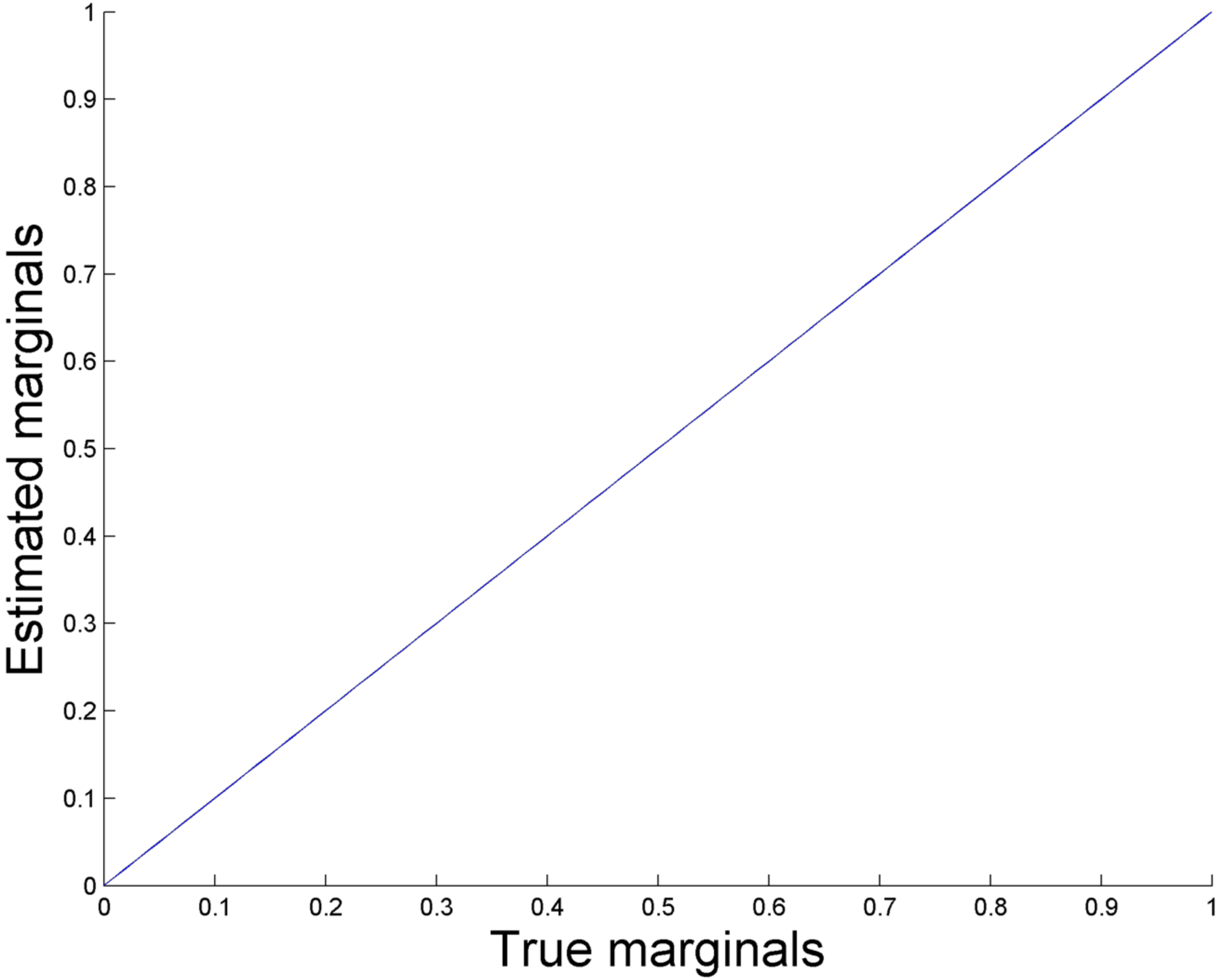
△=Ground Truth

Belief Propagation

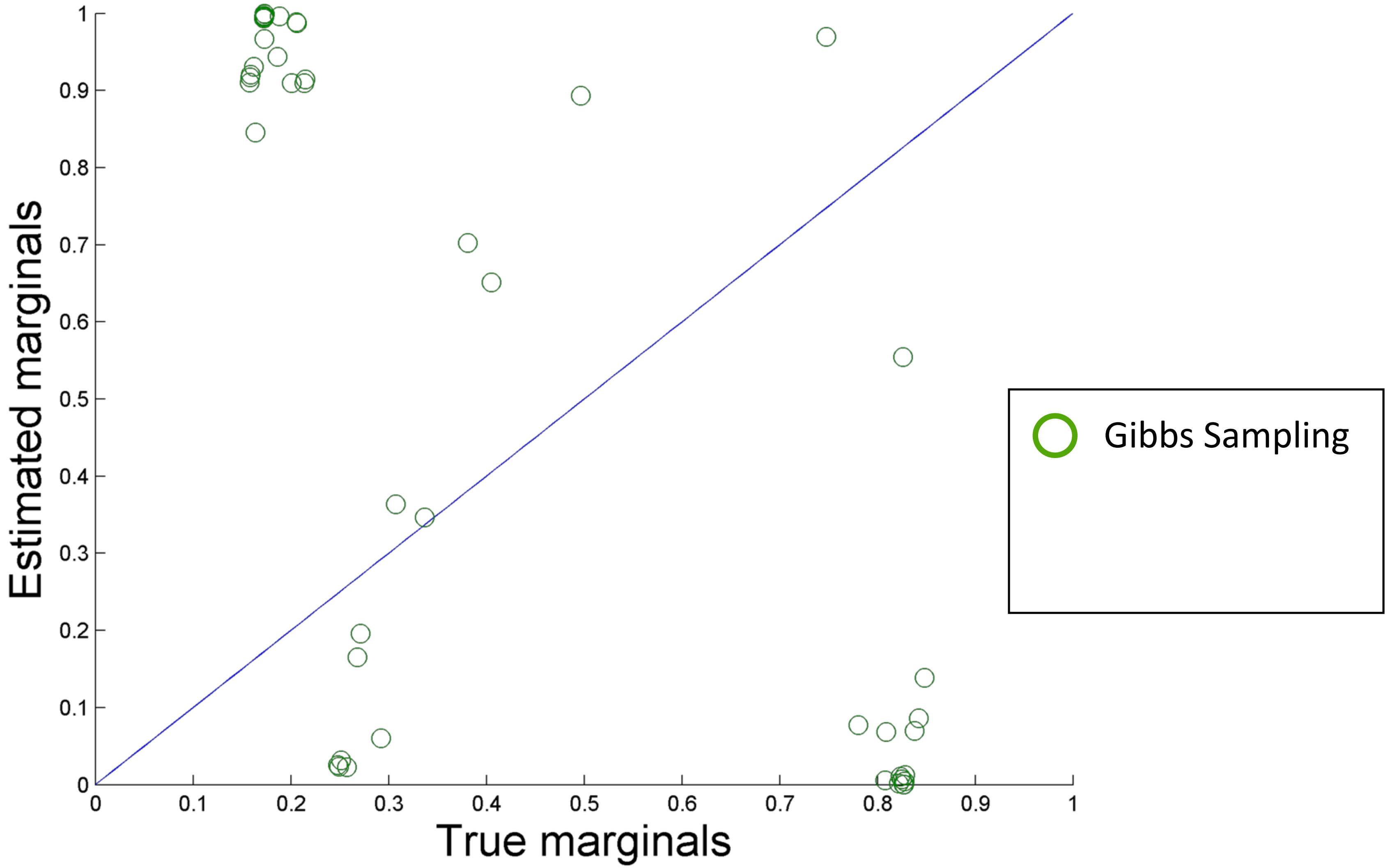
Mean Field

Variational methods provide inaccurate estimates
(well outside the error band)

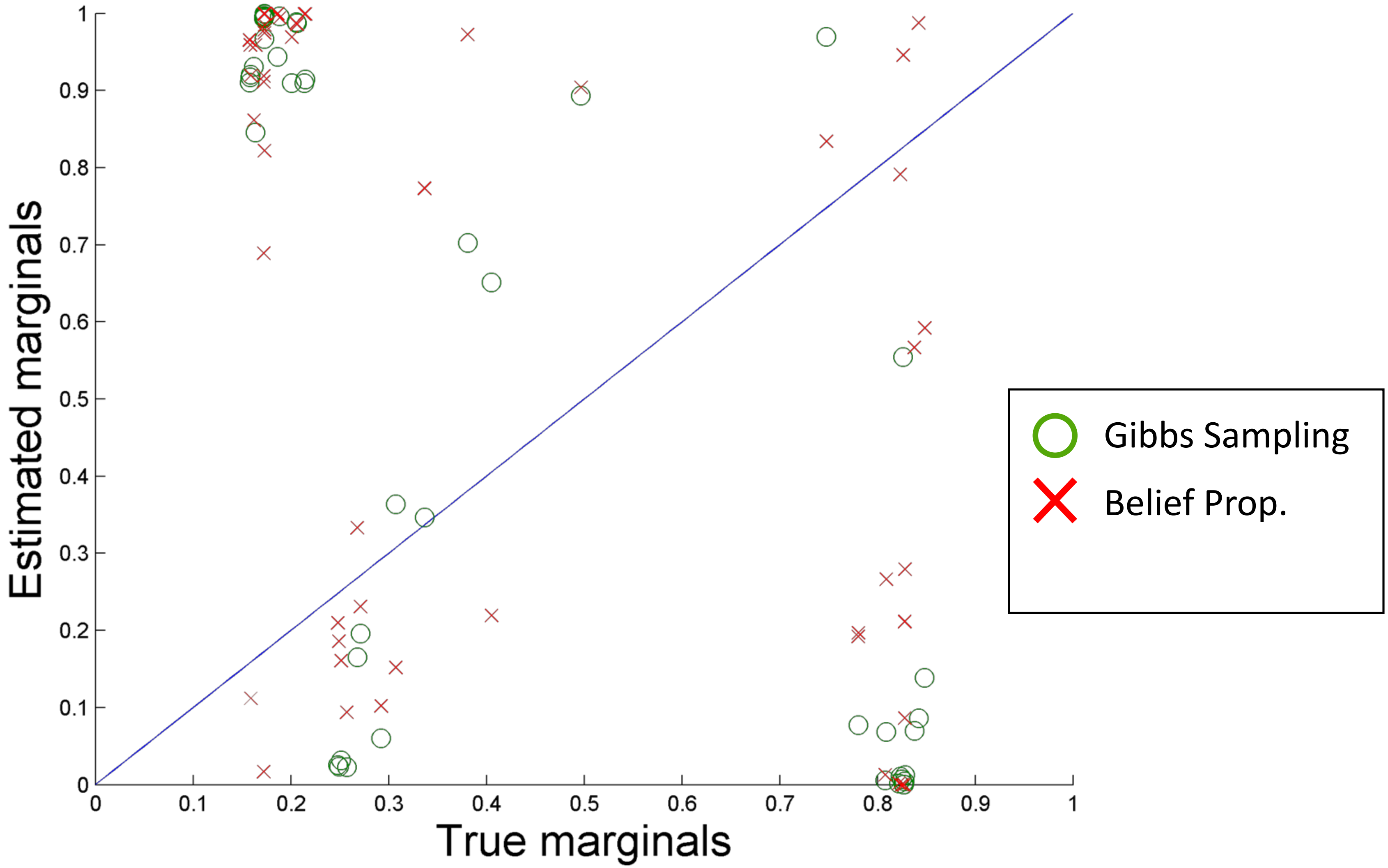
Inference benchmark: Ising models from physics



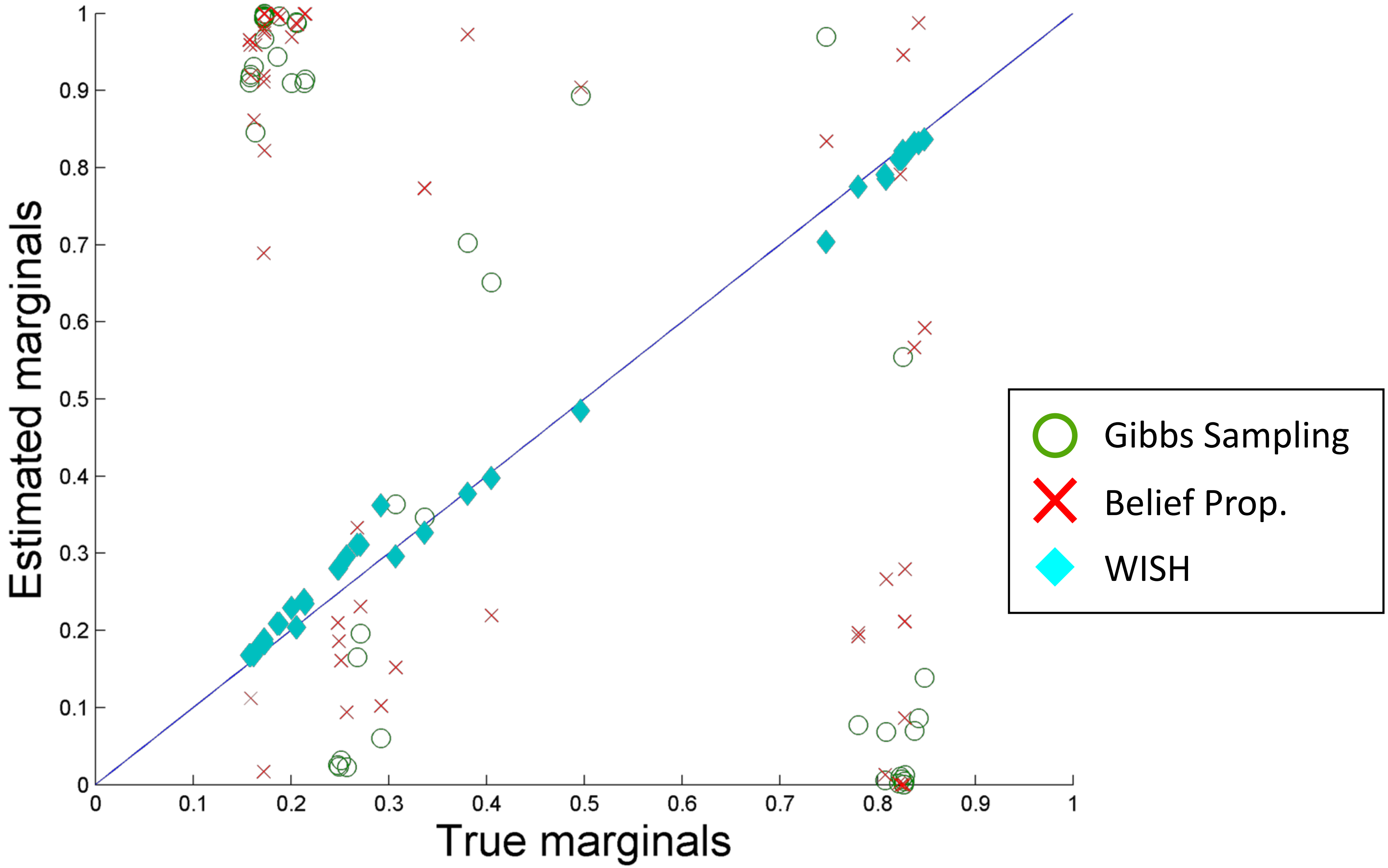
Inference benchmark: Ising models from physics



Inference benchmark: Ising models from physics



Inference benchmark: Ising models from physics



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 **sparse (low-density) parity constraints**

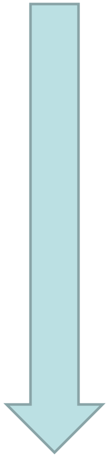
$X = 1$ → length 1, large variance

$X \oplus Y = 0$ → length 2, variance?

$X \oplus Y \oplus Q = 0$ → length 3, variance?

…

$X \oplus Y \oplus \ldots \oplus Z = 0$ → length n/2, small variance



Increasingly complex constraints ☹

Increasingly low variance ☺

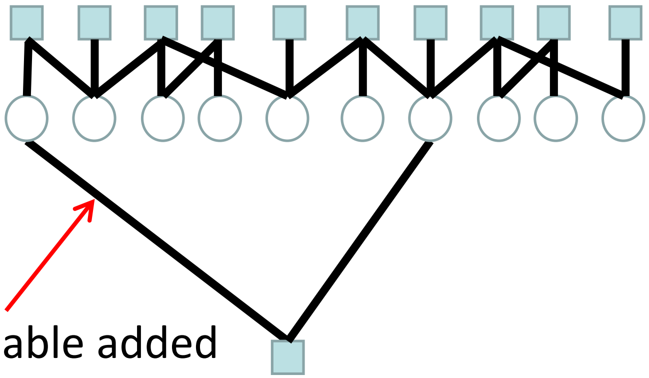
The shorter the constraints are, the easier they are to reason about.

The longer the constraints are, the more accurate the counts are

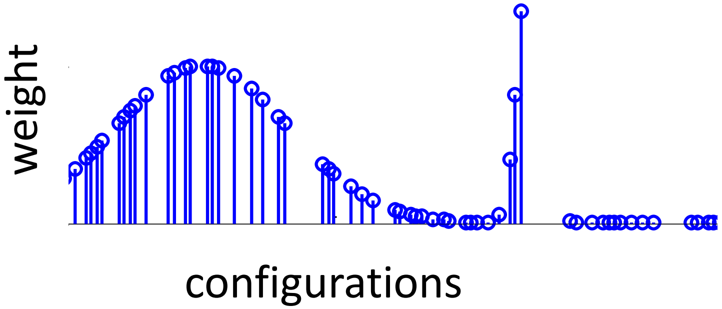
Can short constraints actually be used?

Low density parity constraints

- Short XOR/parity constraints
  - $a \oplus b \oplus c \oplus d$ = 1: satisfied if an odd number of a,b,c,d are set to **1**



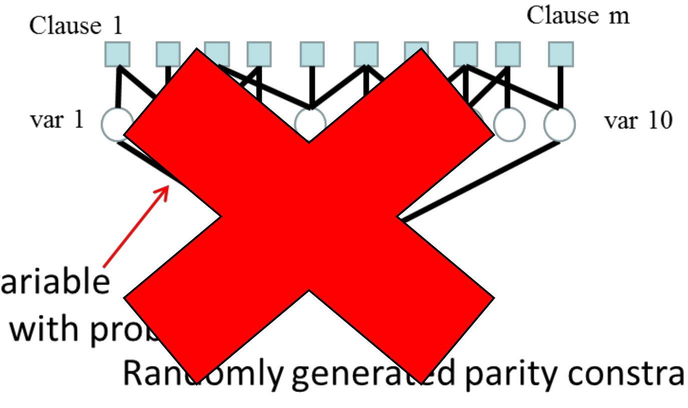
Each variable added with probab. << 0.5



Randomly generated parity constraint X

$$x_1 \oplus x_7 = 1$$

Before:



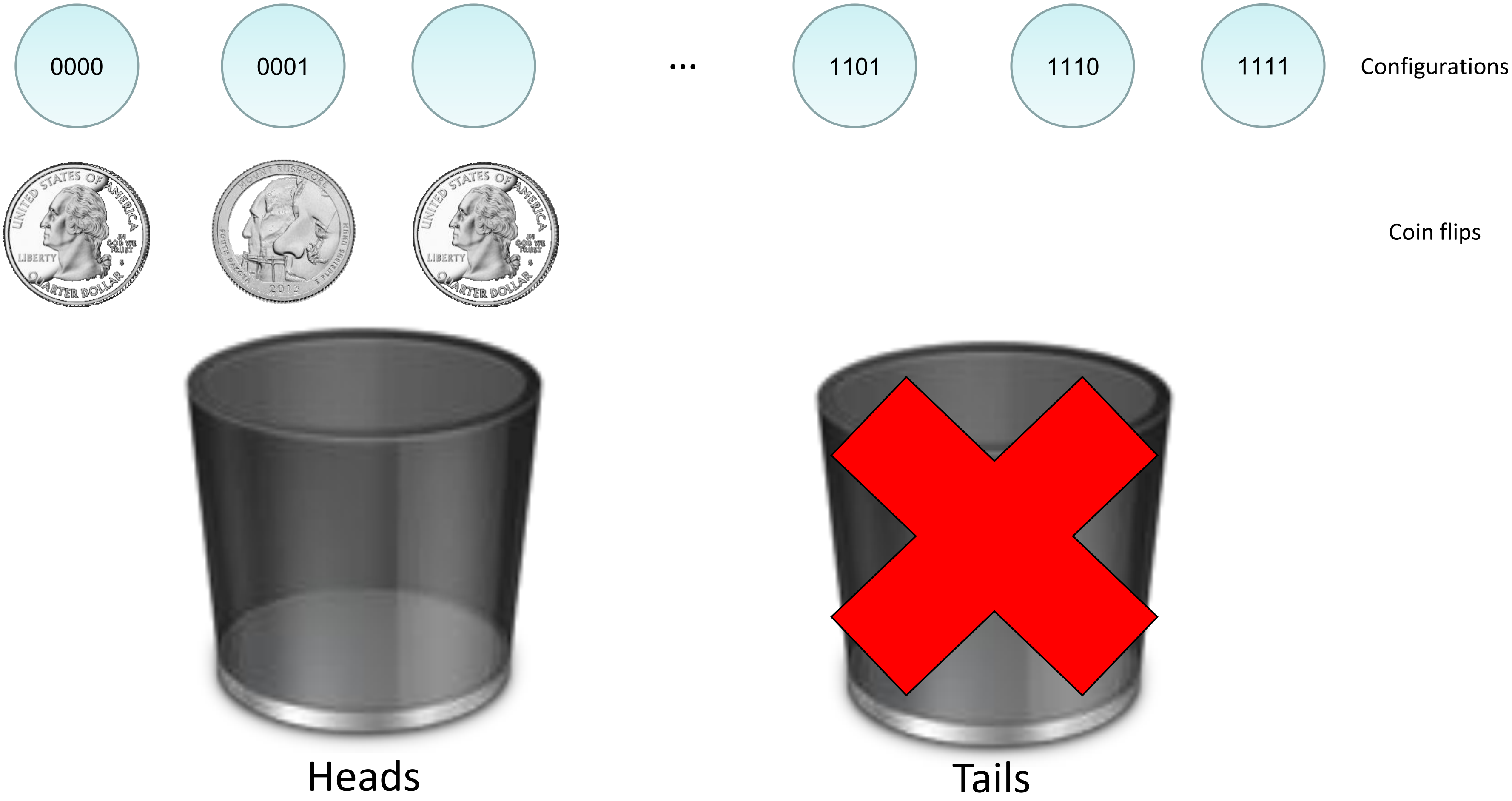
Each variable added with prob 0.5

Randomly generated parity constraint X

$$x_1 \oplus x_3 \oplus x_4 \oplus x_7 \oplus x_{10} = 1$$

Random coin flipping

- **Recall the distributed coin flipping mechanism**
- Ideally, each configuration flips a coin **independently**



0000 0001 … 1101 1110 1111 Configurations

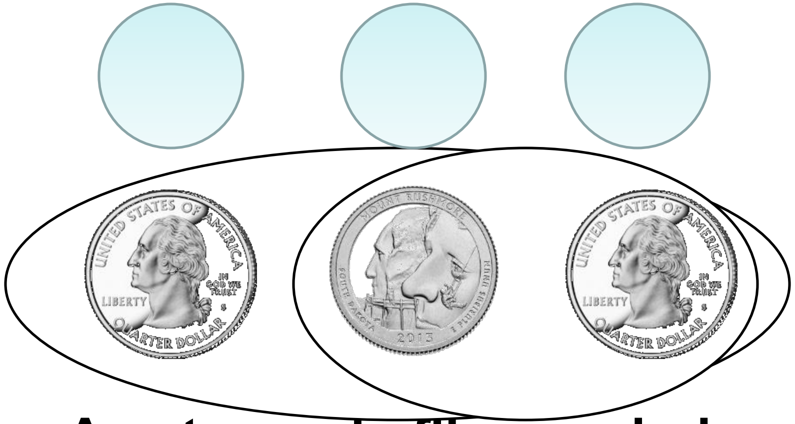
Coin flips

Heads

Tails

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**

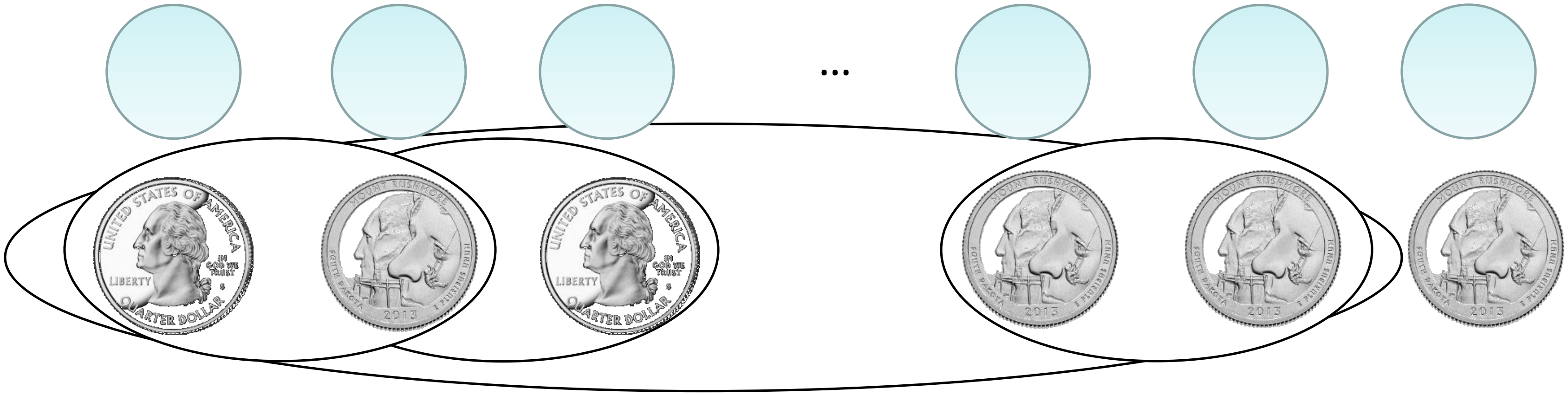
Still works! Pairwise independence guarantees that configurations do not cluster too much in single bucket.

Can be simulated using **random parity constraints**: simply add each variable with probability ½.

``Long'' parity constraints are difficult to deal with!

Average Universal Hashing

- New class of **average universal hash functions** (coin flips generation mechanism)



- 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

Short Parity Constraints for Probabilistic Inference and Model Counting

Main Theorem (stated informally): [AAAI-16]

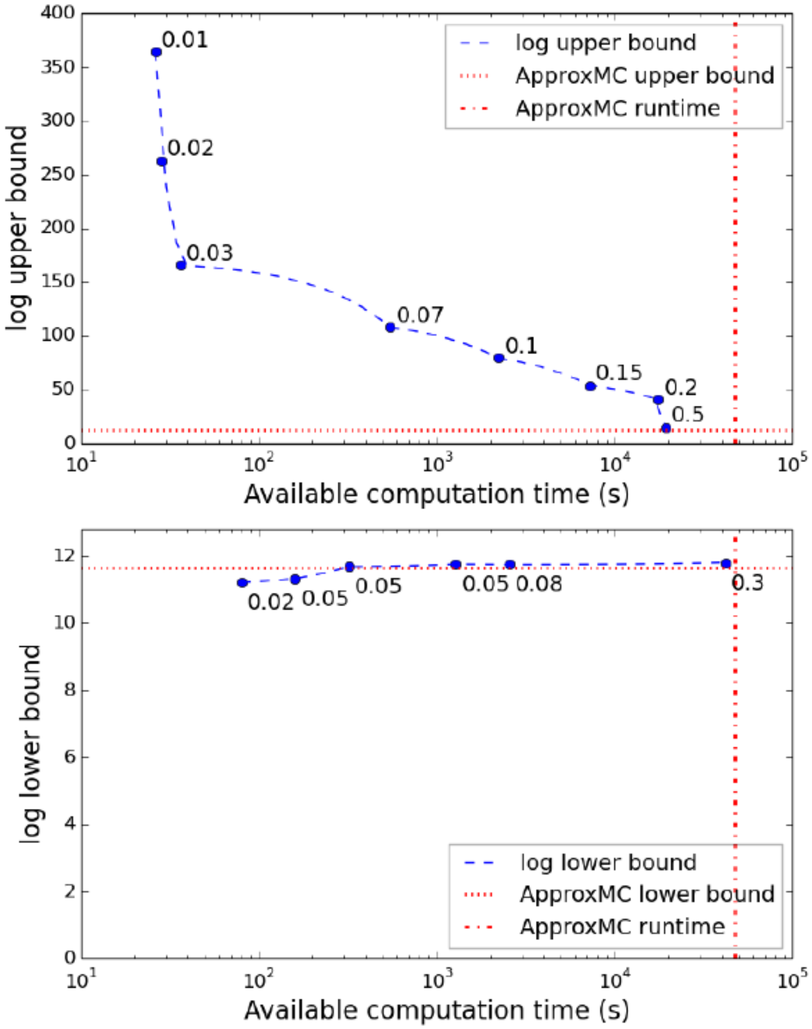
For large enough *n* (= number of variables),

- 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.

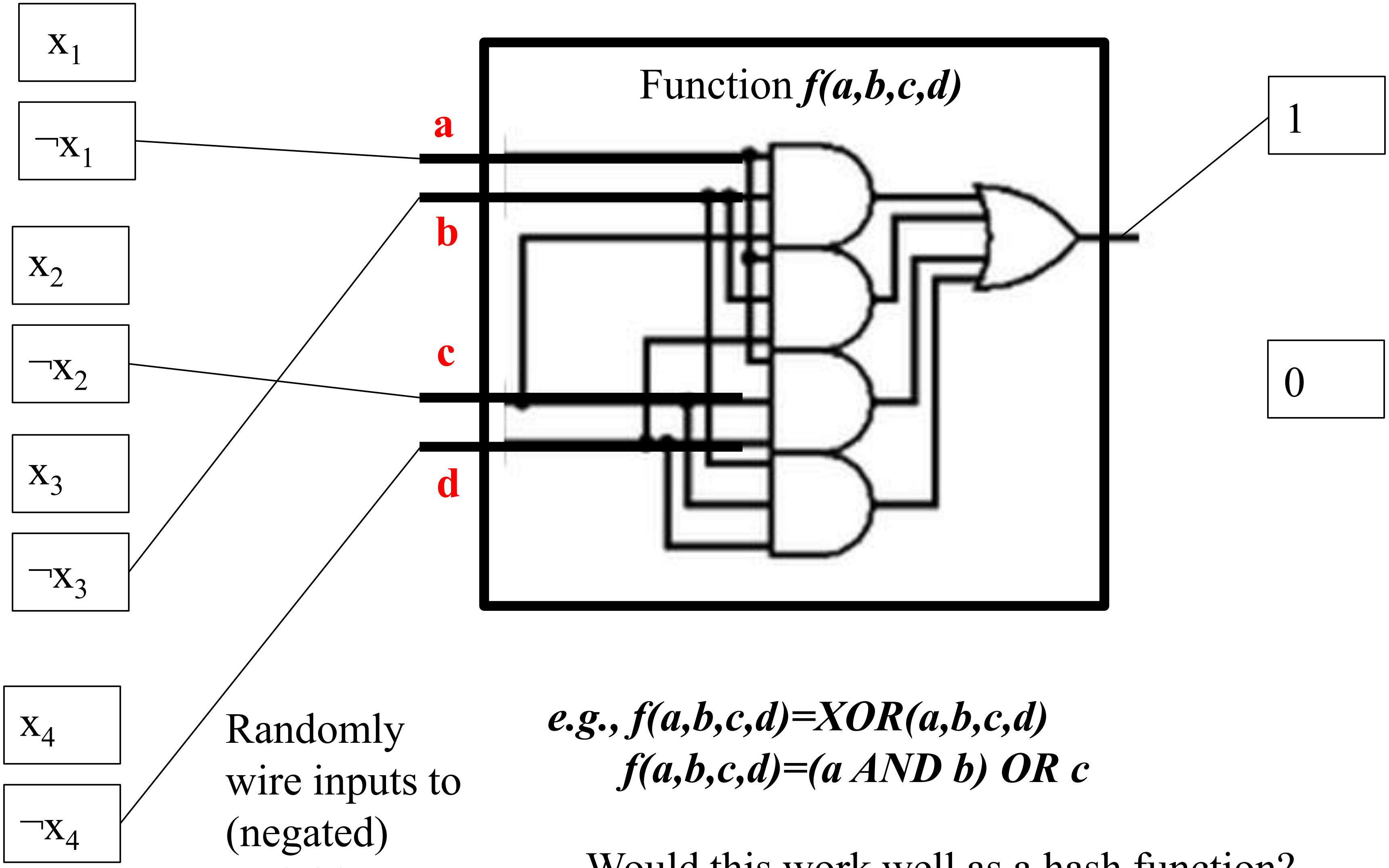
Example



Sparse XOR constraints provide lower and upper bounds

Can other constraints be used?

Random Constraints for Probabilistic Inference



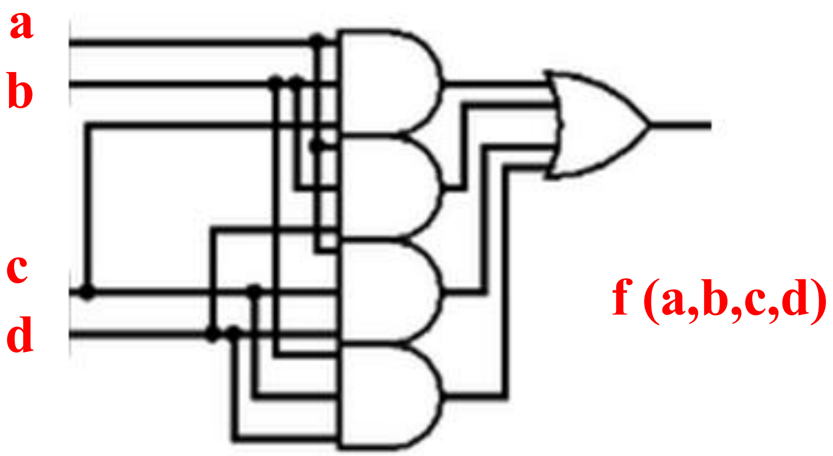
Randomly wire inputs to (negated) variables

e.g., f(a,b,c,d)=XOR(a,b,c,d)

f(a,b,c,d)=(a AND b) OR c

Would this work well as a hash function?

Noise sensitivity

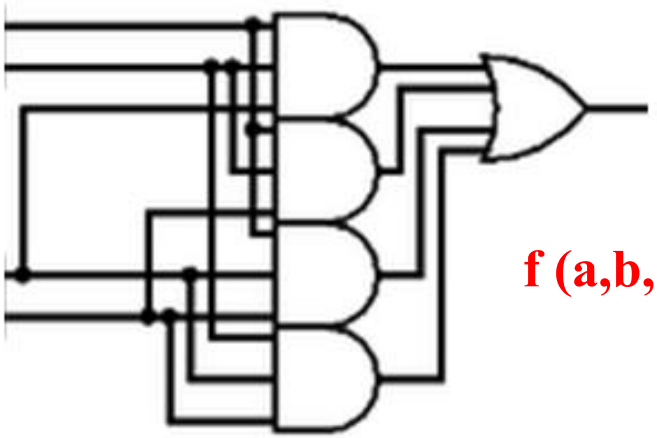


Performance of this family of hash functions depends on the **noise sensitivity** of f (a,b,c,d)

- Given a random input x, e.g. x=(0, 1,1,0)
- Randomly flip each bit of x with probability p → x'=(1, 1,1,0)
- How likely is that f(x) = f(x')?

Theorem (informal statement): The more noise sensitive a function f is, the better the corresponding family of hash functions (with random wiring) behaves. [ICML-16]

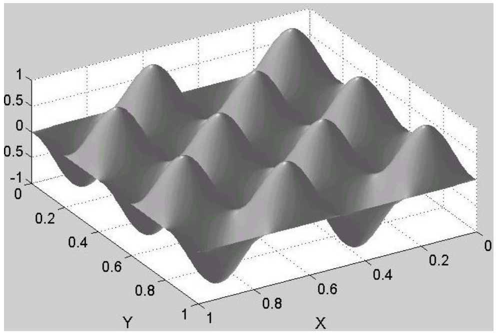
Random Constraints for Probabilistic Inference and Model Counting

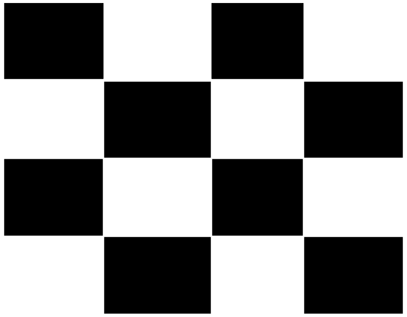


f (a,b,c,d)

- Noise sensitivity can be computed in *closed form* from the Fourier spectrum of **f.** Known for many common functions!
- Intuitively, the more "oscillatory" **f** is, the more noise sensitive it is

2-D sine wave





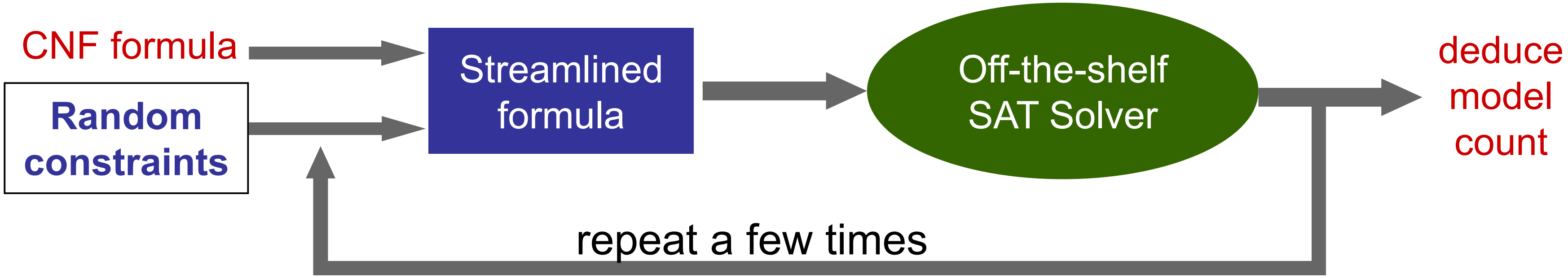
2-D parity function

- Corollary: XORs are the "best" hash function

Using random constraints for counting

Given a formula F

1. Add some random 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



Difference from previous method:

Can use other constraints instead of XORs! For example, *majority function* or "*tribes*"

Experimental results

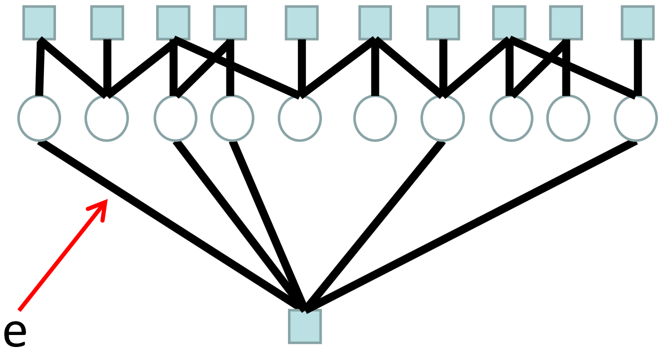
INSTANCE	GT	LB_{trib}	LB_{xor}	t_{trib}	t_{xor}
LS10	24	18	19	**1**	209
LS11	33	25	24	**28**	623
LS12	—	32	29	**34**	658
LS13	—	33	34	**3**	74
LS14	—	36	34	**12**	761
LS15	—	**39**	34	**51**	829
20RDR45	—	29	29	**1**	523
23RDR45	—	**27**	10	**7**	800

Tribes: One or two orders of magnitude faster!

$$\begin{aligned} & (x_1 \wedge x_2 \wedge \neg x_3) \\ \vee\ & (\neg x_4 \wedge \neg x_5) \\ \vee\ & (\neg x_6 \wedge x_7) \end{aligned}$$

Hashing as a random projection

- 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**

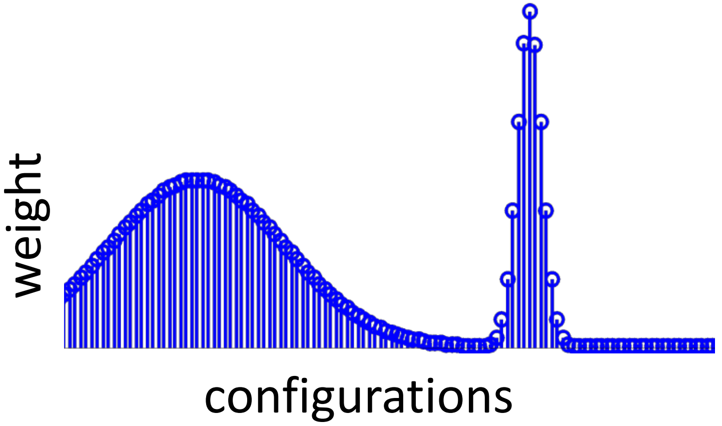


Each variable added with prob. 0.5

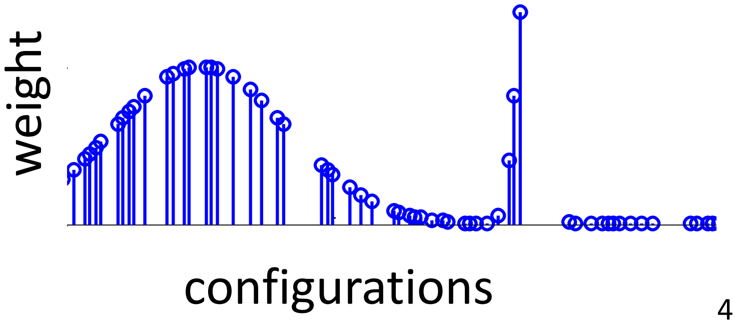
Randomly generated parity constraint X

$$x_1 \oplus x_3 \oplus x_4 \oplus x_7 \oplus x_{10} = \mathbf{1}$$

Set weight to zero if constraint is not satisf



Random projection

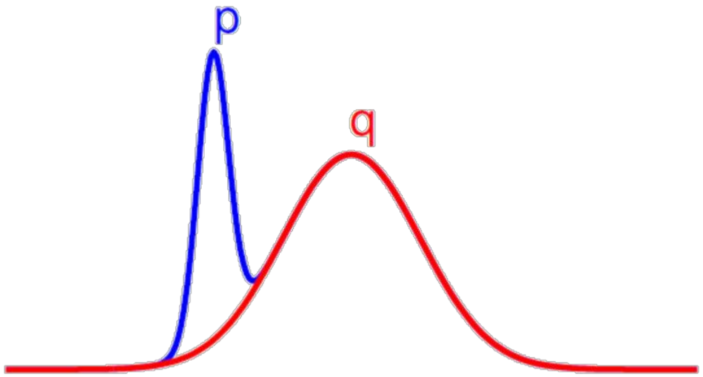


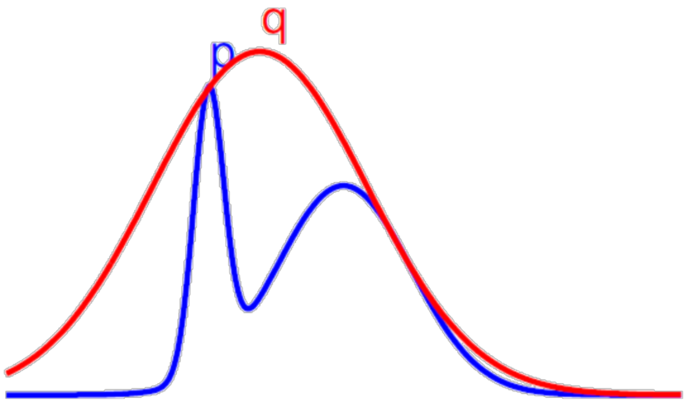
This random projection:

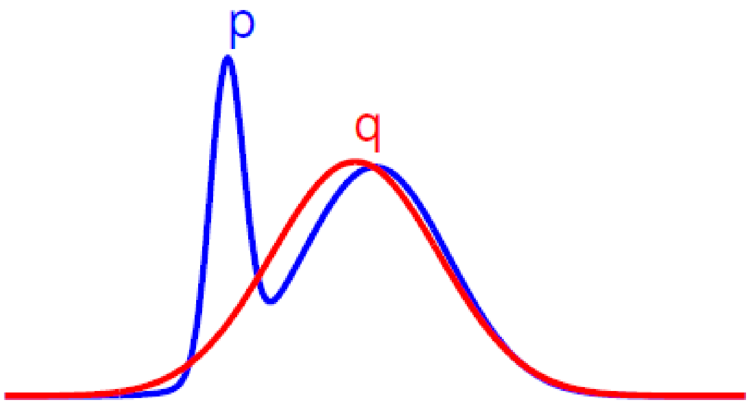
1. "**simplifies**" the model
2. preserves its "key properties"

Variational Inference: basic idea

Approximate a complex distribution ***p*** using a simpler (tractable) distribution ***q*** (e.g., a Gaussian distribution)



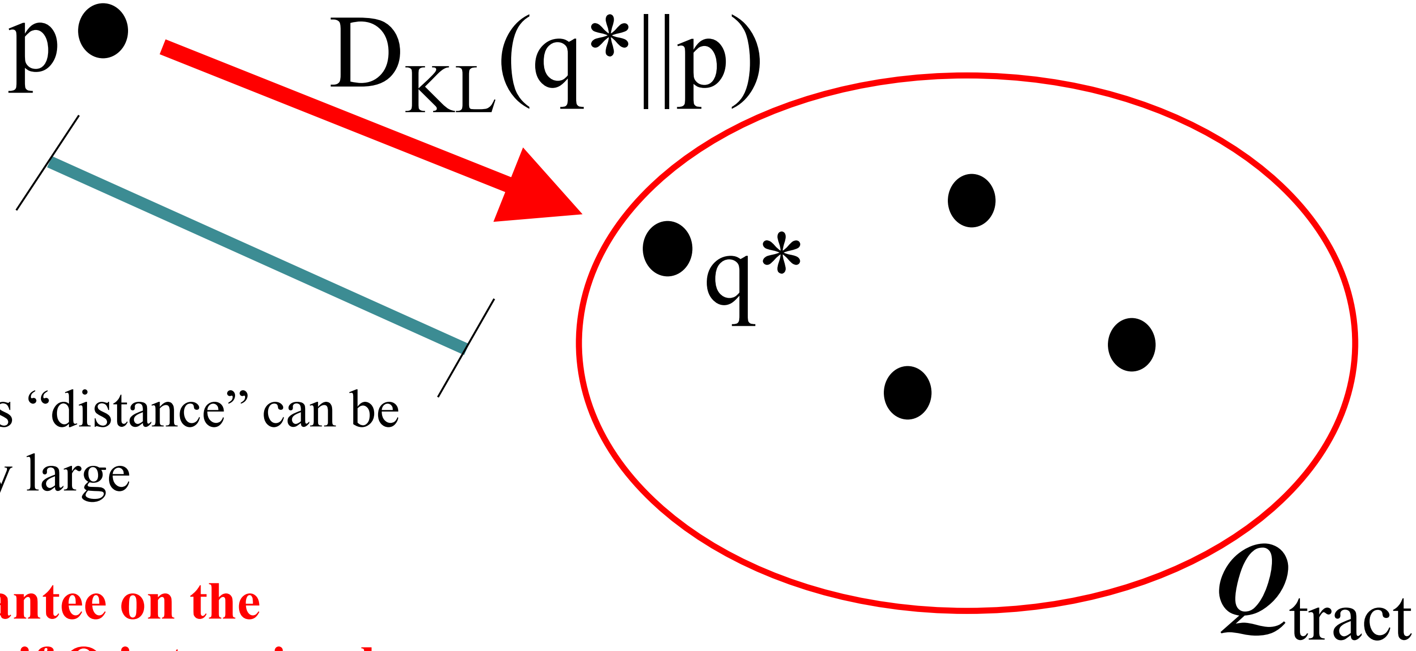




Credit: Minka et al.

Variational Inference

q* is the best approximation of p, among the tractable distributions in Q_{tract}

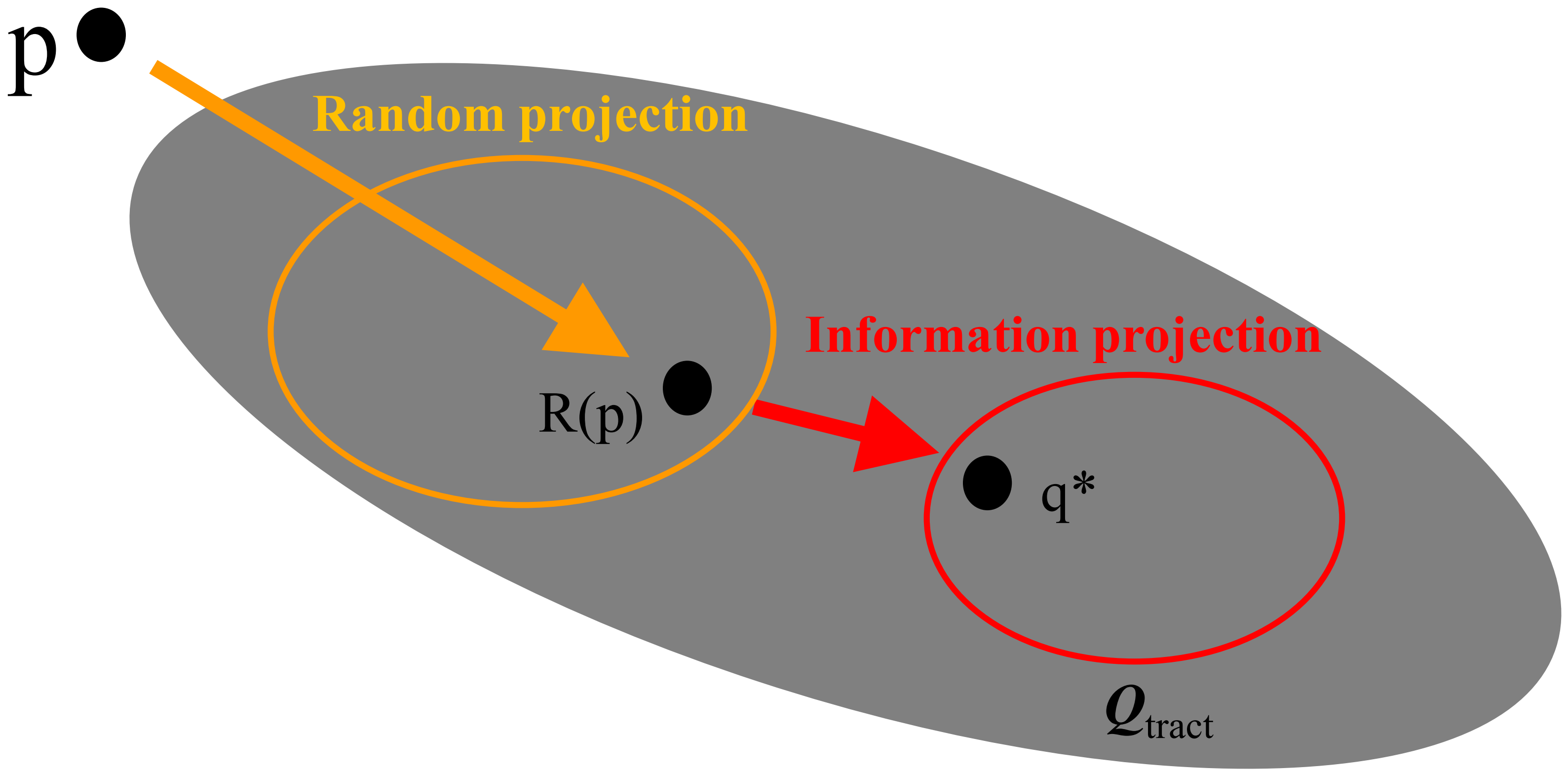


Issue: this “distance” can be arbitrarily large

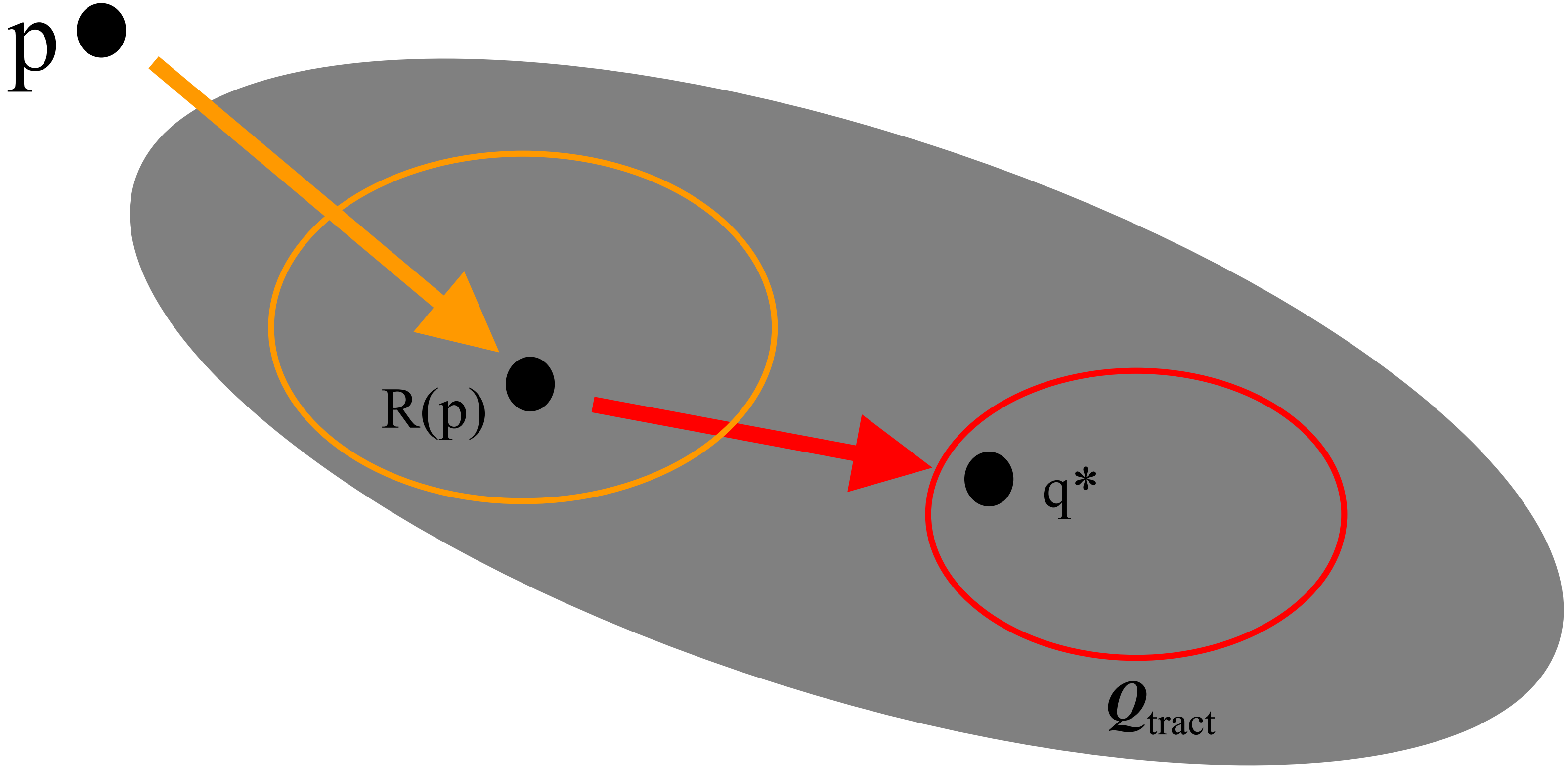
No guarantee on the accuracy if Q is too simple: the approximation can be arbitrarily bad

E.g., mean field
$q(x_1,x_2,x_3)= q_1(x_1)\ q_2(x_2)\ q_3(x_3)$

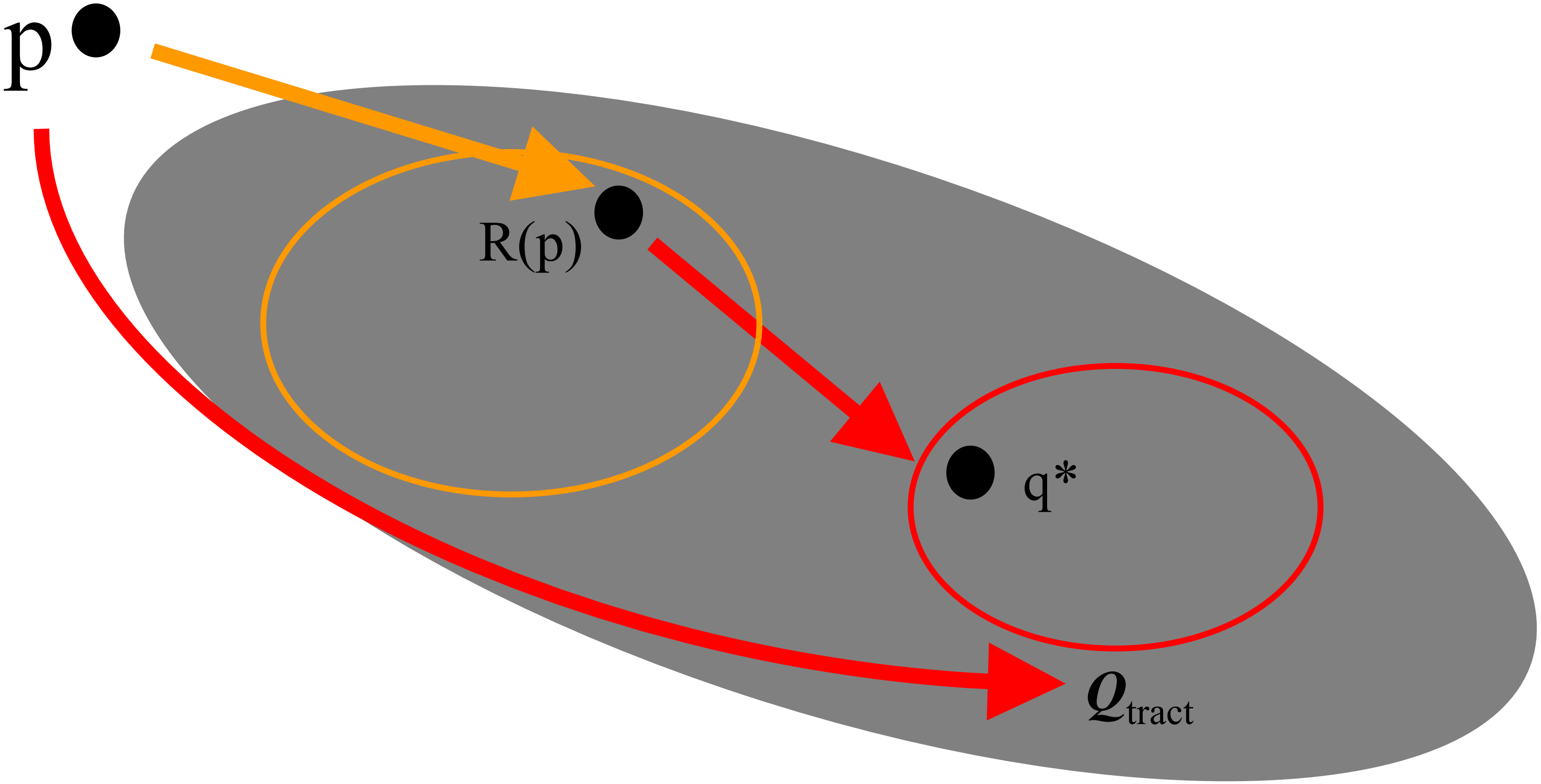
Variational Inference with Random Projections



Variational Inference with Random Projections



Variational Inference with Random Projections



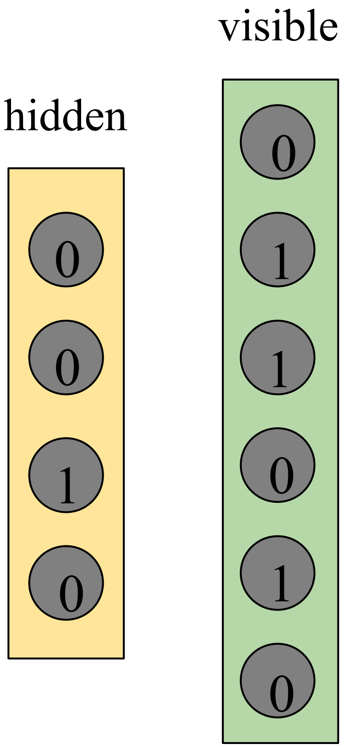
Main result: after the random projection (e.g., using XORs), the resulting distribution can be **well approximated** using standard variational inference (with guarantees)

General Variational Result

Theorem:

If an approximating set of distributions ***Q*** contains the set of degenerate distributions ***D***, then variational inference over ***Q*** using this random projection scheme will yield tight bounds on the partition function (model count).

Mean Field with Random Projections



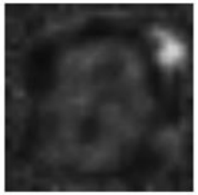
Fully factored distribution:

$$p(x) = p(x_1)\, p(x_2)\, \ldots\, p(x_n)$$

Randomly-projected mean field:

Variational objective is still coordinate-wise concave and has fewer variables

RBMs

No. Hidden Nodes	100	100	100	200	200	200
CD-k	1	5	15	5	15	25
MF log Z	501	348	297	203	293	279
MFRP log Z	501	**433**	**342**	**380**	**313**	**295**
MF μ						
MFRP μ						

Conclusions

- Reasoning engines for deterministic knowledge are guaranteed to provide the right answer
- **Probabilistic reasoning**:
  - So far, main focus on scalability rather than **accuracy/robustness**
  - Recent approaches: try to “reduce” to deterministic problems so that we can leverage existing solvers.
  - Provides nice theoretical guarantees, as opposed to traditional approaches (MCMC, variational)
  - Lots more to explore: continuous variables, satisfiability modulo theory, accuracy vs runtime tradeoffs
- Dream: general purpose, probabilistic inference engines that are both scalable and (provably) accurate