Improving Approximate Counting: From Linear to Logarithmic SAT calls

(When Practice Drives Theory)

Kuldeep S. Meel

Rice University

Joint work with Supratik Chakraborty (IIT Bombay) and Moshe Y. Vardi (Rice U.)

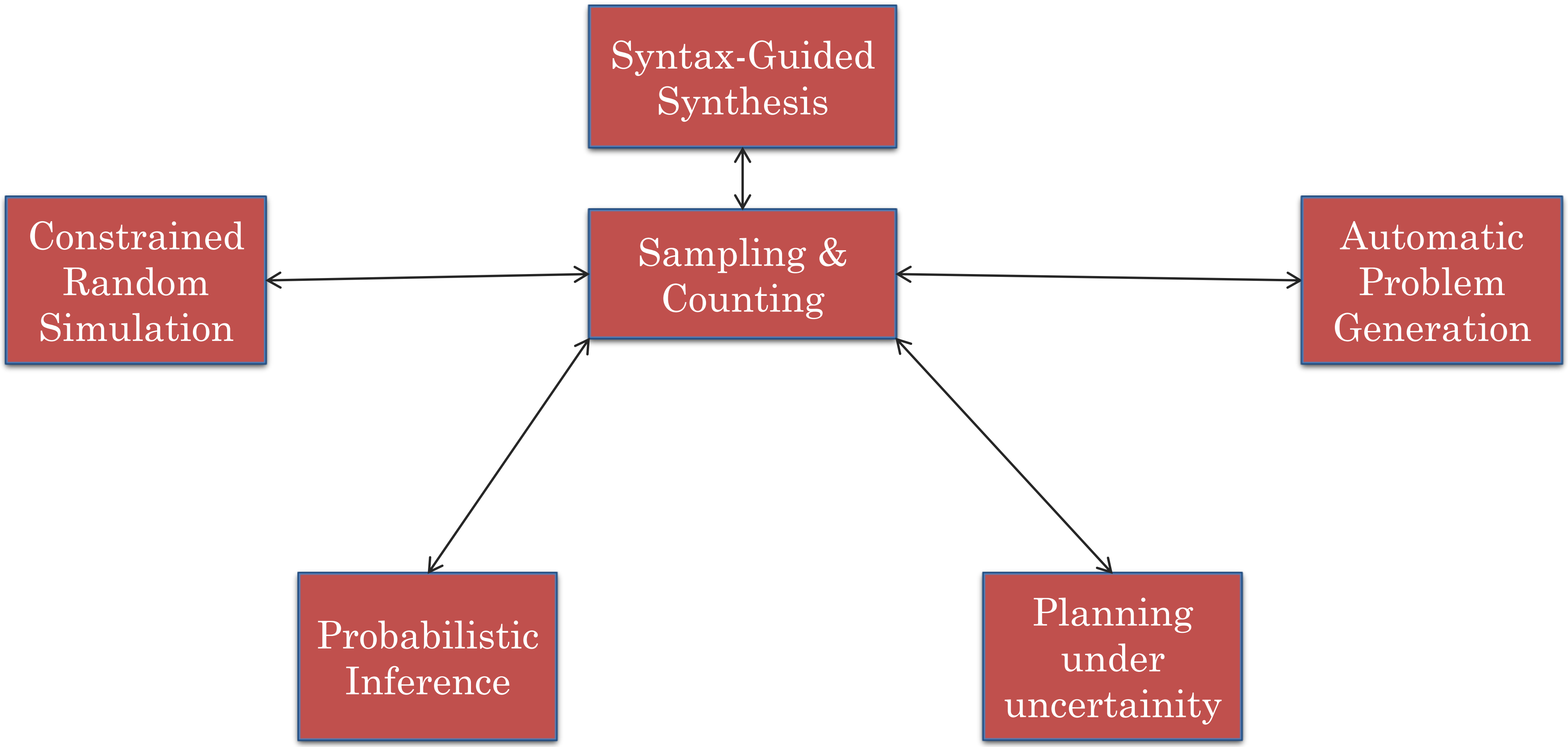
Constrained Counting

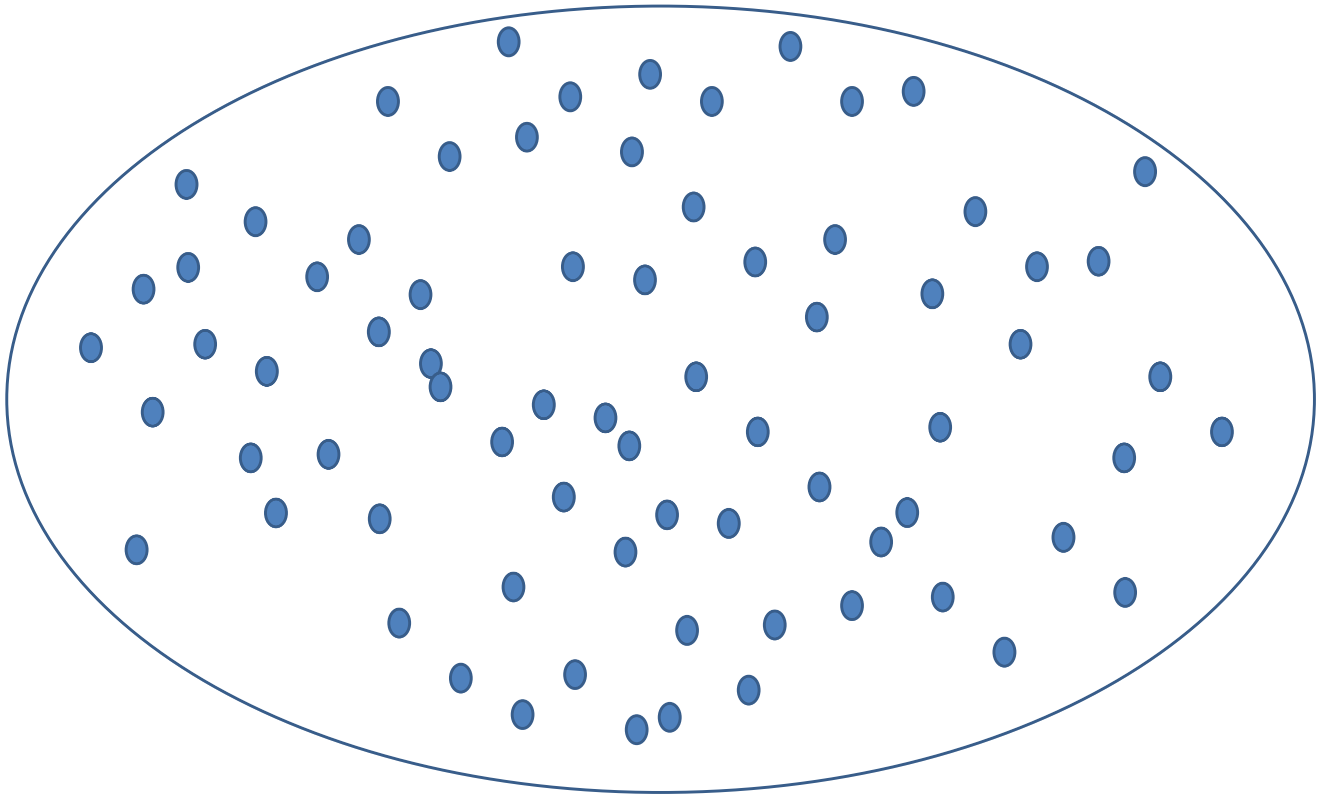
- F: CNF Formula; R_F : Solution Space of F
- F: $(a \vee b)$; $R_F = \{(0,1), (1,0), (1,1)$; $|R_F| = 3$

- Probably Approximately Correct (PAC) Counter
  - Input: F, tolerance: ε, confidence: δ Output: C

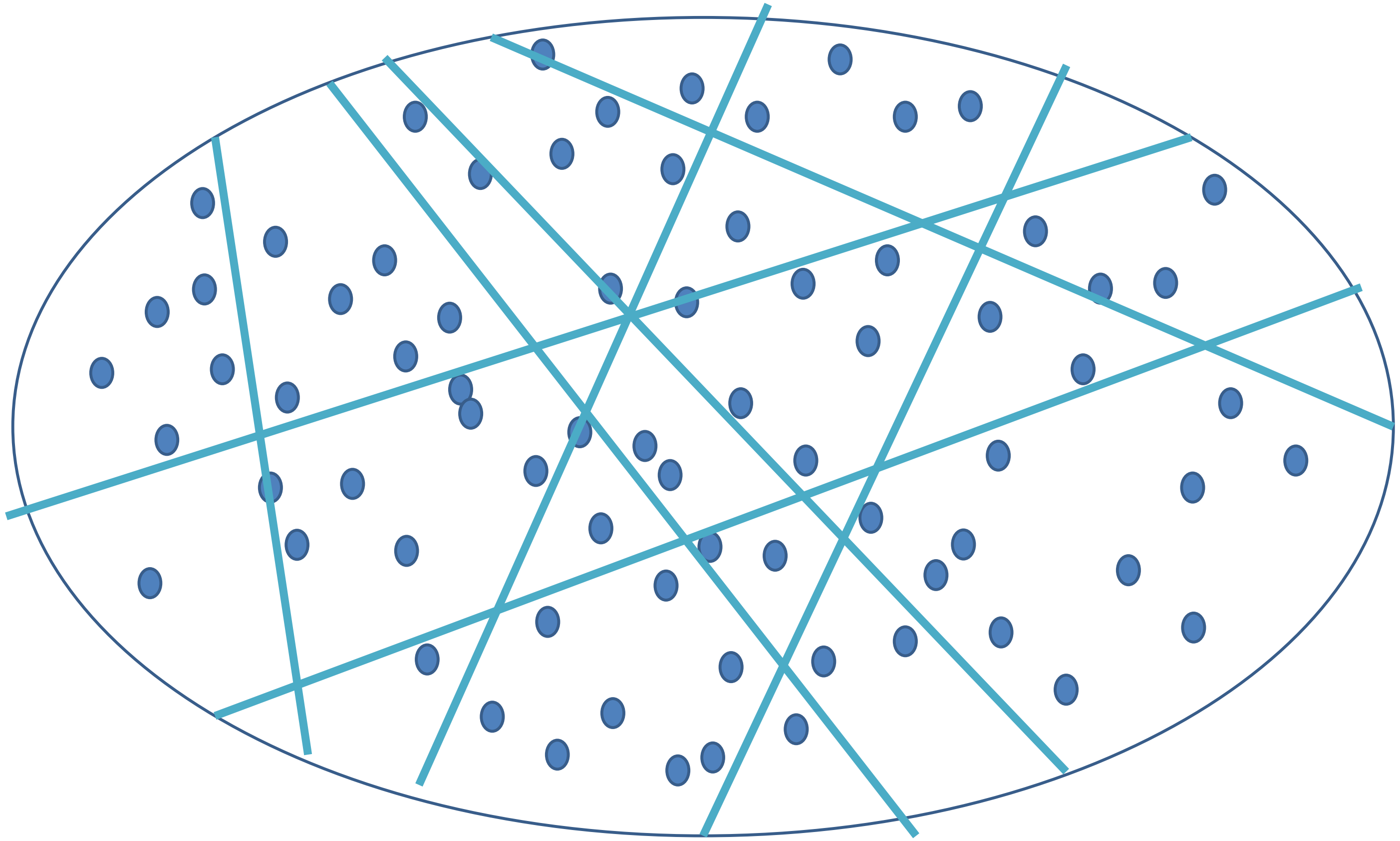
$$\Pr\left[\frac{|R_F|}{(1+\varepsilon)} \leq C \leq |R_F|(1+\varepsilon)\right] \geq \delta$$

Diverse Applications



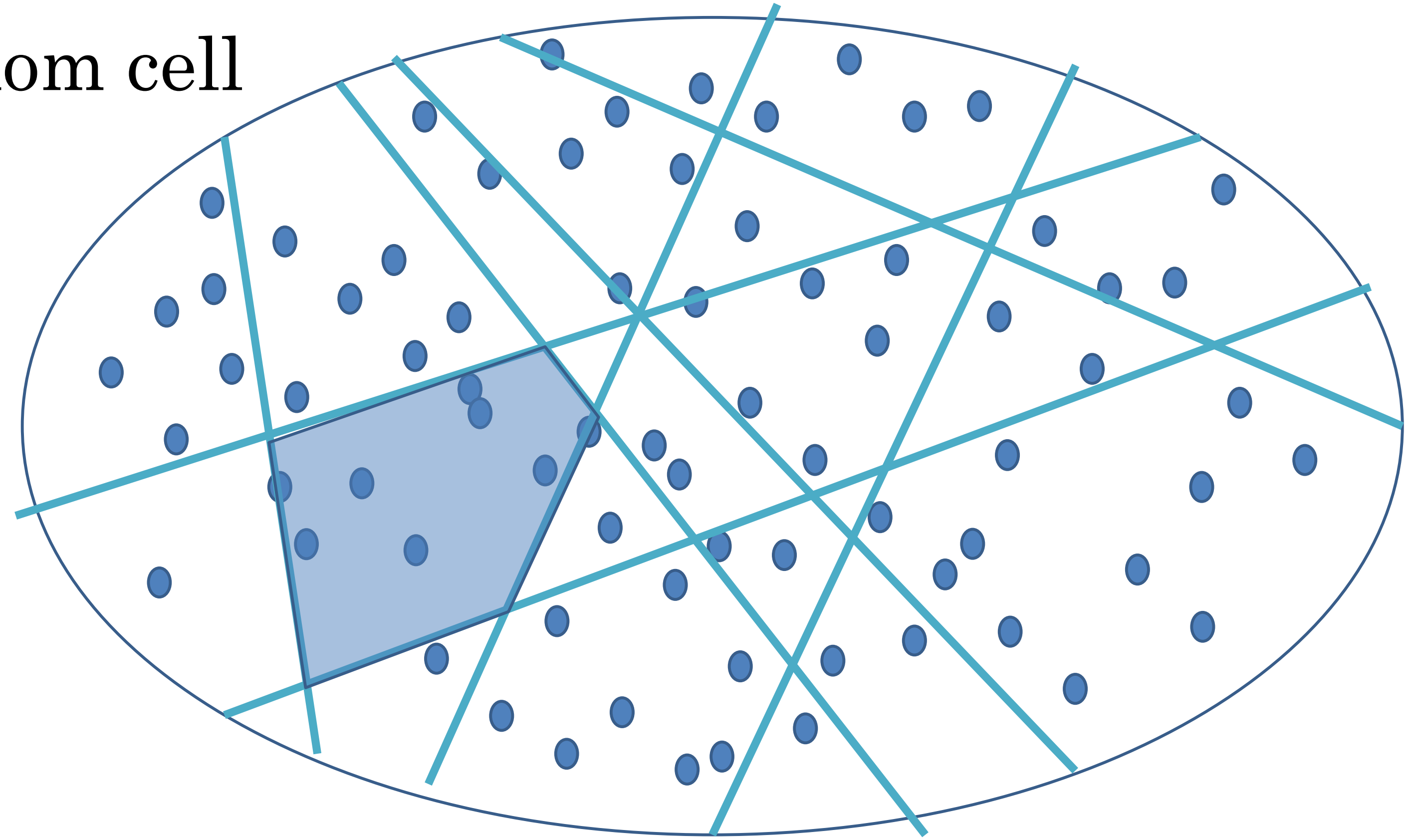


Partitioning into equal "small" cells



Approximate Counting

Pick a random cell



Estimate = # of solutions in cell * # of cells

Partitioning

1. How large is the “small” cell?

2. How do we compute solutions inside a cell?

3. How many cells?

Question 1: Size of cell

- Too large => Hard to enumerate
- Too small => Ratio of standard deviation to mean is very high

$$\textbf{thresh} = 5\left(1 + \frac{1}{\varepsilon^2}\right);$$

Question 2: Solving a cell

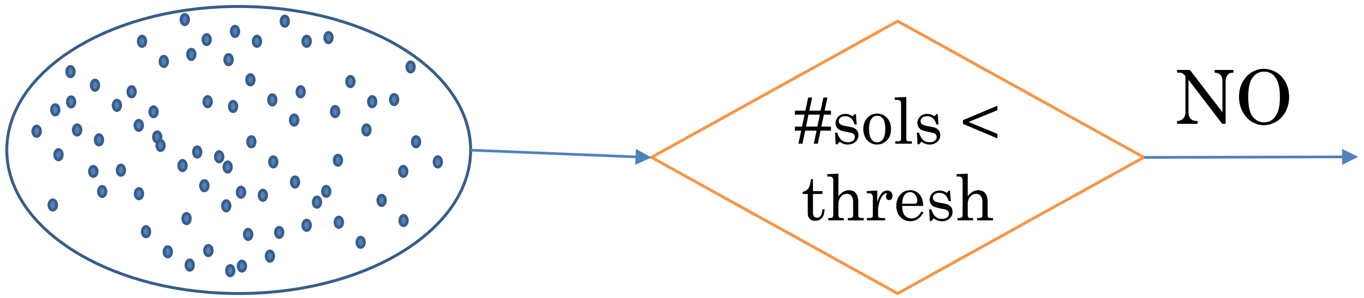
- Variables: $X_1, X_2, X_3, \ldots, X_n$
- To construct h: $\{0,1\}^n \rightarrow \{0,1\}^m$, choose m random XORs
- Pick every variable with prob. ½ , XOR them and add 1 with prob. ½
- E.g.: $X_1 \oplus X_3 \oplus X_6 \oplus \ldots \oplus X_{n-1}$
- $\alpha \in \{0,1\}^m \rightarrow$ Set every XOR equation to 0 or 1 randomly
- The cell: F ∧ XORs

$$(F \wedge Q_1 \wedge Q_2 \cdots \wedge Q_m)$$

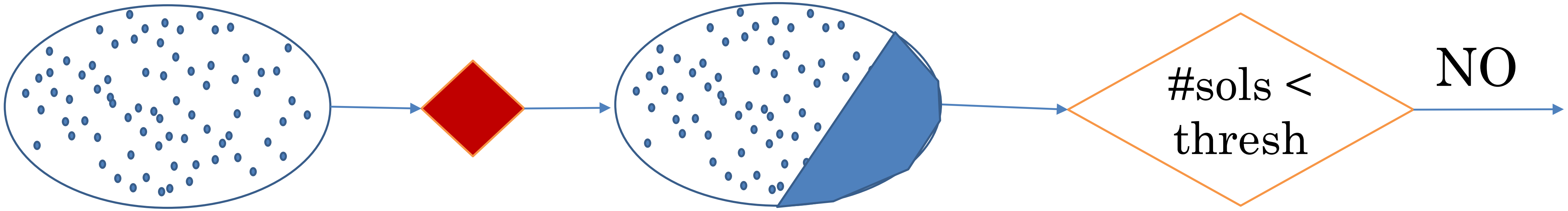
$$Q_1 \coloneqq (X_1 \oplus X_3 \oplus X_6 \oplus \ldots X_{n-1} = 0)$$
$$Q_2 \coloneqq (X_1 \oplus X_2 \oplus X_4 \oplus \ldots X_{n-1} = 1)$$
$$Q_3 \coloneqq (X_1 \oplus X_3 \oplus X_5 \oplus \ldots X_{n-1} = 0)$$
$$Q_4 \coloneqq (X_2 \oplus X_3 \oplus X_4 \oplus \ldots X_{n-1} = 0)$$
……
$$Q_m \coloneqq (X_1 \oplus X_2 \oplus X_3 \oplus \ldots X_{n-1} = 0)$$

m XORs

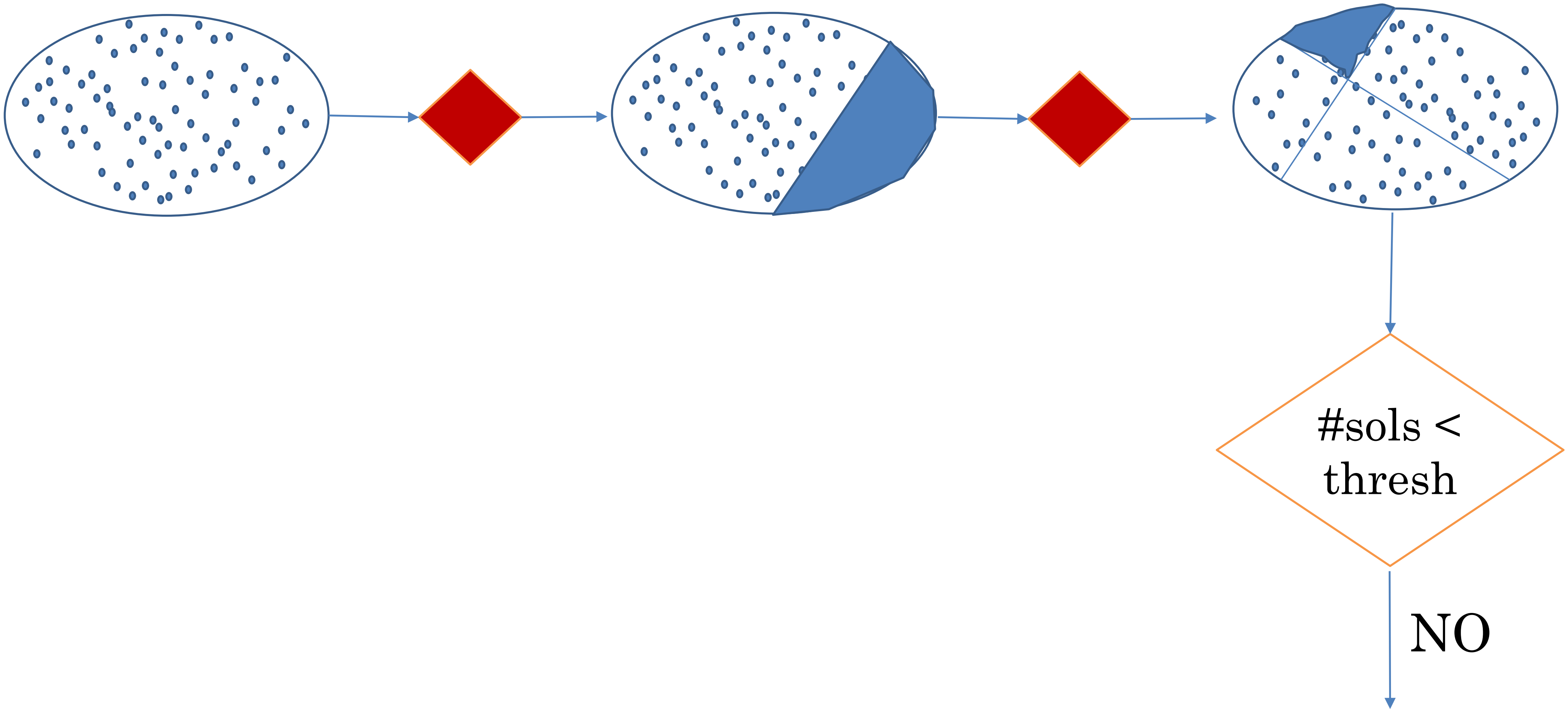
Question 3: How many cells?



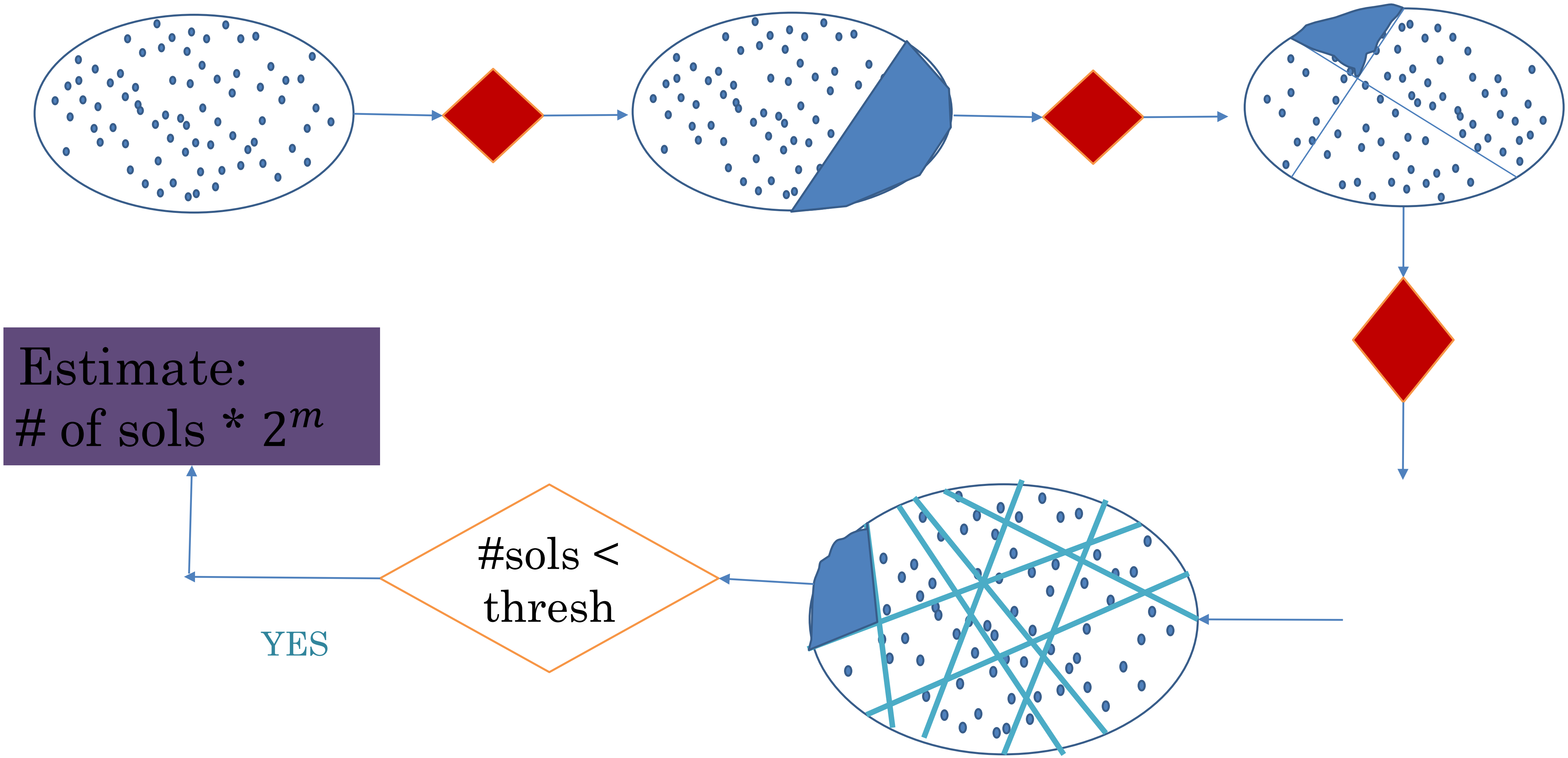
Question 3: How many cells?



Question 3: How many cells?



Question 3: How many cells?



Question 3: How many cells?

- Query 1: # of sols $(F \wedge Q_1^1) < thresh$
- Query 2: # of sols $(F \wedge Q_1^2 \wedge Q_2^2)$ < thresh
- ……
- Query n: # of sols $(F \wedge Q_1^n \wedge Q_2^n \cdots Q_n^n)$ < thresh
- Stop when query m returns YES and return

  # of sols$(F \wedge Q_1^m \wedge Q_2^m \cdots Q_n^m) * 2^m$
- # of SAT calls is O(n)

ApproxMC(F,ε, δ)

Theorem 1:

$$\Pr\left[\frac{|R_F|}{(1+\varepsilon)} \leq \text{ApproxMC}(\text{F},\varepsilon,\delta) \leq |R_F|(1+\varepsilon)\right] \geq \delta$$

Theorem 2:

$$\text{ApproxMC}(\text{F},\varepsilon,\delta) \text{ makes } O\left(\frac{n \log \frac{1}{1-\delta}}{\varepsilon^2}\right) \text{ calls to NP oracle}$$

Challenge

Hashing-based Approaches to counting and sampling

- Stockmeyer 1983
- Jerrum, Valiant, and Vazirani 1986
- CAV 2013
- CP 2013
- UAI 2013
- NIPS 2013
- DAC 2014
- ICML 2014
- AAAI 2014
- TACAS 2015
- IJCAI 2015
- ICML 2015
- UAI 2015
- AAAI 2016
- AISTATS 2016
- ICML 2016

Can we improve number of SAT calls from O(n)?

Improving SAT oracle based algorithms

Extend reach of SAT oracle computing

- Consider other complexity classes
  - Most successes are for the lower levels of the (F)PH
- Develop tighter query complexity results
  - Provide optimal guarantees on the number of oracle calls
  - Also, account for non-constant run time of CDCL SAT oracle?
- Target other high-profile applications

J. Maques-Silva (Day 1 of **this** workshop)

Beyond Classical Oracle Model

- Query 1: # of sols $(F \wedge Q_1^1) < thresh$
- Query 2: # of sols $(F \wedge Q_1^2 \wedge Q_2^2)$ < thresh
- ……
- Query n: # of sols $(F \wedge Q_1^n \wedge Q_2^n \cdots Q_n^n)$ < thresh

- Practitioner's view
  1. Query 1 and Query n are not equally hard in practice
  2. Solving $(F \wedge Q_1^1)$ followed by $(F \wedge Q_1^2 \wedge Q_2^2)$ is different than solving $(F \wedge Q_1^1)$ followed by $(F \wedge Q_1^1 \wedge Q_2^2)$

Beyond ApproxMC

- What if we do:
  - Query 1: # of sols$(F \wedge Q_1) <$ thresh
  - Query 2: # of sols$(F \wedge Q_1 \wedge Q_2) <$ thresh
  - ……
  - Query n: # of sols $(F \wedge Q_1 \wedge Q_2 \wedge \cdots Q_n) <$ thresh

- Independence has been crucial to analysis of counting algorithms (Stockmeyer 1983, Jerrum, Valiant and Vazirani 1986…..)
- T_i: Query i returns YES; S_i: Estimate retuned by Query i on termination is correct
- Independence helped us to simplify

  $\Pr[T_i | \neg T_{i-1}] = \Pr[T_i]$ and $\Pr[S_i | \neg T_{i-1}] = \Pr[S_i]$
- Contribution: A new analysis that applies to several hashing-based algorithms

The key idea behind New Analysis

- B: Event that estimate returned is outside the desired $(1+\varepsilon)$ interval
- $m^* = \log \frac{|R_F|}{\text{thresh}}$ (i.e., $2^{m^*} = \frac{|R_F|}{\text{thresh}}$)
- T_i: Query i returns YES ; S_i: Estimate computed in Query i on termination is correct
- Lemma 1: $\Pr[B] = \Pr[\cup_{i=1}^{m^*-2} T_i] + \Pr[\neg S_{m^*-1} \cap T_{m^*-1}] + \Pr[\neg S_{m^*}]$
- Lemma 2: $\Pr[\cup_{i=1}^{m^*-2} T_i] < 0.1$
- Informally, Probability of making a bad choice early on is very small.

ApproxMC2

- Query 1: # of sols$(F \wedge Q_1) <$ thresh
- Query 2: # of sols$(F \wedge Q_1 \wedge Q_2) <$ thresh
- ……
- Query n: # of sols $(F \wedge Q_1 \wedge Q_2 \wedge \cdots Q_n) <$ thresh
- Stop when query m returns YES and return

$$\text{\# of sols}(F \wedge Q_1 \wedge Q_2 \wedge \cdots Q_m) * 2^m$$

- Observation: # of sols of formula in query i < # of sols of formula in query i-1
  - If Query i answers No, then Query i-1 must answer No
  - Binary search to find m

ApproxMC2: The twist in Binary search

- Query m: # of sols $(F \wedge Q_1 \wedge Q_2 \wedge \cdots Q_m) <$ thresh
- The # of solutions is typically very small compared to 2^n
  - We terminate for m << n
- Performing "Query n/2" is very very expensive (in practice)
  - In fact, for almost all our benchmarks, CMS will timeout with "Query n/2"
- Galloping search

ApproxMC2 (F,ε, δ)

Theorem 1:

$$\Pr\left[\frac{|R_F|}{(1+\varepsilon)} \leq \text{ApproxMC2}(\text{F},\varepsilon,\delta) \leq |R_F|(1+\varepsilon)\right] \geq \delta$$

Theorem 2:

$$\text{ApproxMC2}(\text{F},\varepsilon,\delta) \text{ makes } O\left(\frac{(\log n)\log\frac{1}{1-\delta}}{\varepsilon^2}\right) \text{ calls to NP oracle}$$

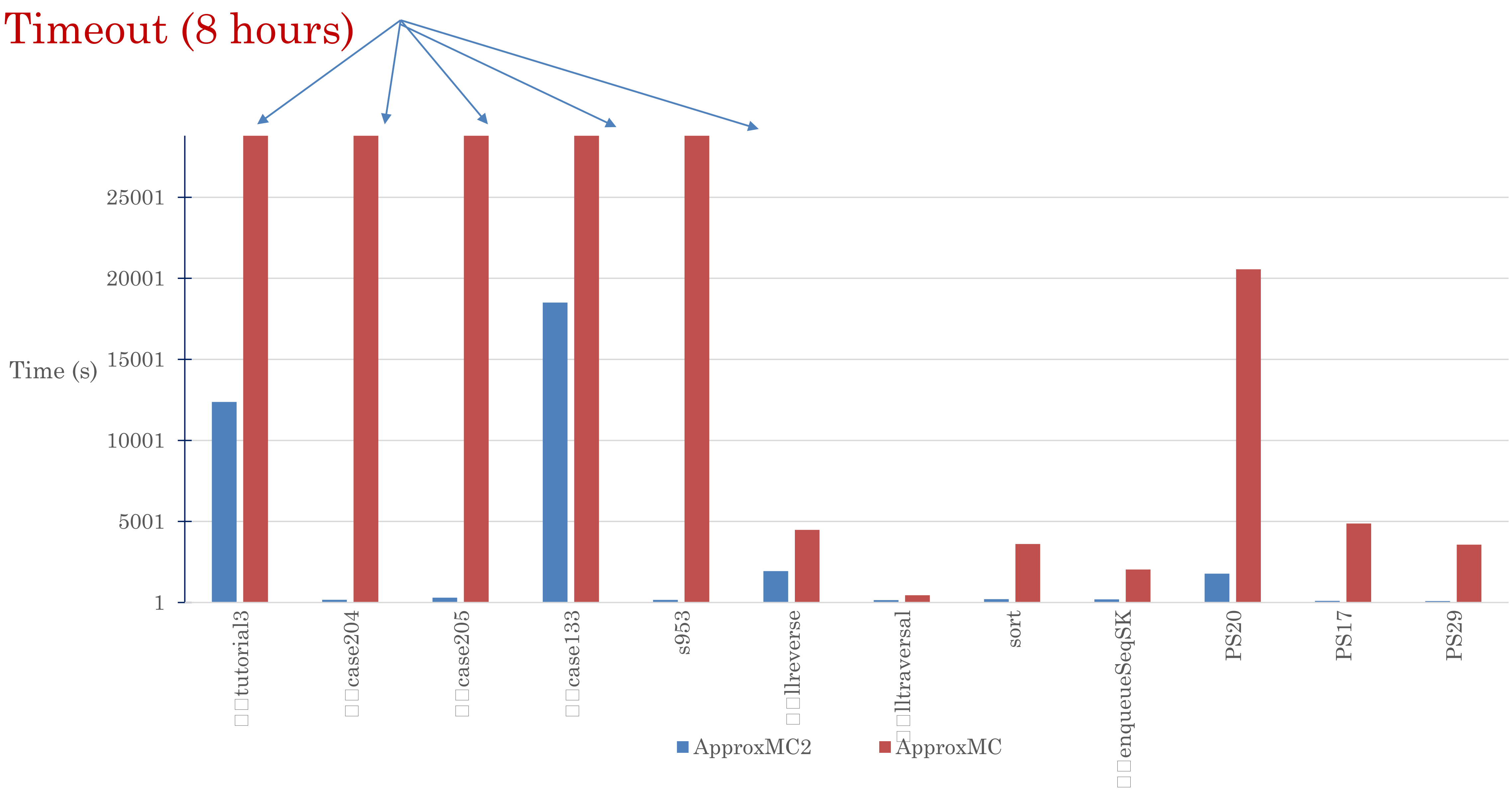
Theorem 3:

If F is DNF formula, then ApproxMC2 is FPRAS – fundamentally different from the only other known FPRAS for DNF (Karp, Luby 1983)

Beyond ApproxMC

- The proposed proof framework can be applied to other algorithms
  - PAWS (Ermon et al 2014)
  - WeightMC (Chakraborty et al 2014, Belle et al 2015)
- Reduces number of SAT calls from O(n) or O(n log n) to O(log n)

Runtime Performance Comparison



Conclusion

- The success of CDCL presents opportunities to solve problems in higher complexity classes
- Hashing-based techniques combine progress in SAT solving with theoretical strength of universal hashing
- Revisiting Oracle Model:
  - Not every call to SAT oracle requires similar computational effort
  - SAT oracles require more than constant time to run
- Resulting analysis improves both theoretical and practical complexity.