BIRD: Engineering an Efficient CNF-XOR SAT Solver and its Applications to Approximate Model Counting

Mate Soos Kuldeep S. Meel

National University of Singapore

AAAI 2019

Model Counting

- Given
  - Boolean variables $X_1, X_2, \cdots X_n$
  - Formula F over $X_1, X_2, \cdots X_n$
- $\mathsf{Sol}(F) = \{$ solutions of F $\}$
- SAT: Determine whether $\mathsf{Sol}(F)$ is non-empty

Model Counting

- Given
  - Boolean variables $X_1, X_2, \cdots X_n$
  - Formula F over $X_1, X_2, \cdots X_n$
- $\mathsf{Sol}(F) = \{$ solutions of F $\}$
- SAT: Determine whether $\mathsf{Sol}(F)$ is non-empty
- Model Counting: Determine $|\mathsf{Sol}(F)|$

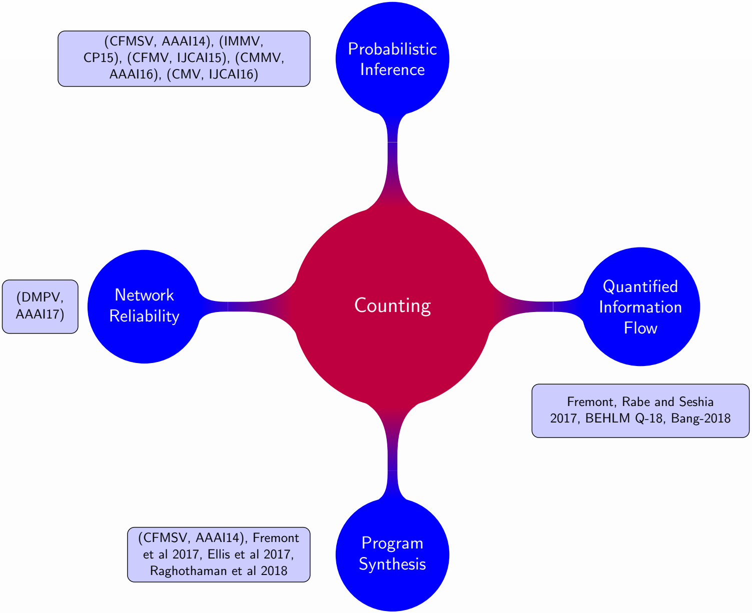
Model Counting

- Given
  - Boolean variables $X_1, X_2, \cdots X_n$
  - Formula F over $X_1, X_2, \cdots X_n$
- $\mathrm{Sol}(F) = \{$ solutions of F $\}$
- SAT: Determine whether $\mathrm{Sol}(F)$ is non-empty
- Model Counting: Determine $|\mathrm{Sol}(F)|$
- Given
  - $F := (X_1 \vee X_2)$
- $\mathrm{Sol}(F) = \{(0,1), (1,0), (1,1)\}$

Model Counting

- Given
  - Boolean variables $X_1, X_2, \cdots X_n$
  - Formula F over $X_1, X_2, \cdots X_n$
- $\mathrm{Sol}(F) = \{$ solutions of F $\}$
- SAT: Determine whether $\mathrm{Sol}(F)$ is non-empty
- Model Counting: Determine $|\mathrm{Sol}(F)|$
- Given
  - $F := (X_1 \vee X_2)$
- $\mathrm{Sol}(F) = \{(0,1),(1,0),(1,1)\}$
- $|\mathrm{Sol}(F)| = 3$

Applications across Computer Science



Model Counting

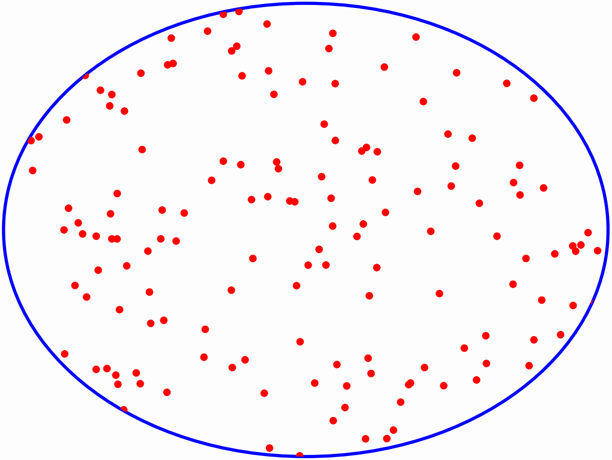
- Given
  - Boolean variables $X_1, X_2, \cdots X_n$
  - Formula F over $X_1, X_2, \cdots X_n$
- Exact Counting: #P-complete (Valiant 1979)

Model Counting

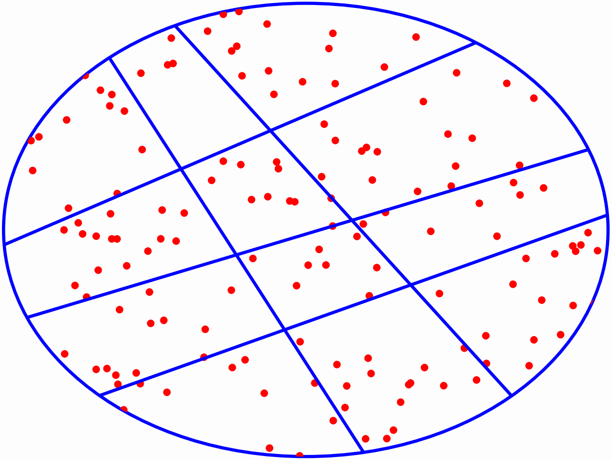
- Given
  - Boolean variables $X_1, X_2, \cdots X_n$
  - Formula F over $X_1, X_2, \cdots X_n$
- Exact Counting: #P-complete (Valiant 1979)
- ApproxCount(F, ε, δ): Compute C such that

$$\Pr[\frac{|\mathsf{Sol}(F)|}{1+\varepsilon} \leq C \leq |\mathsf{Sol}(F)|(1+\varepsilon)] \geq 1-\delta$$

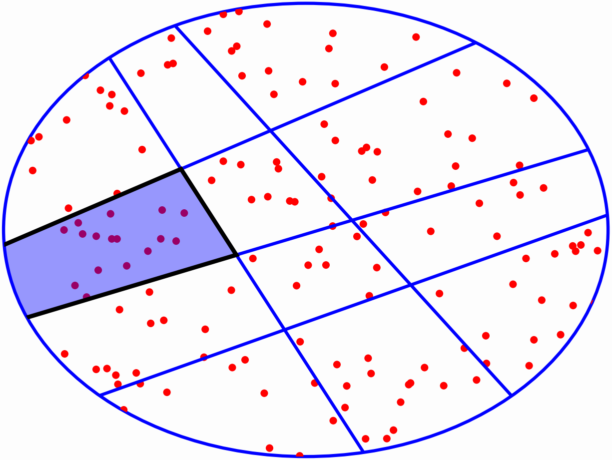
As Simple as Counting Dots



As Simple as Counting Dots



Pick a random cell



Estimate = Number of solutions in a cell × Number of cells

Challenges

How to partition into roughly equal small cells of solutions without knowing the distribution of solutions?

Challenges

How to partition into roughly equal small cells of solutions without knowing the distribution of solutions?

- Use random XORs ($x_1 \oplus x_3 = 0$)
- $F = C_1 \wedge C_2 \cdots C_m$
- $G = F \wedge XOR_1 \wedge XOR_2 \cdots XOR_k$.
- Need a Good SAT Solver to handle CNF+XORs

CryptoMiniSAT: Gaussian Elimination Meets CDCL Solving

Soos et al 2009

- Perform CDCL on CNF Formula
- Perform Gaussian Elimination on XORs
- Exchange Unit/Binary Clauses

CryptoMiniSAT: Gaussian Elimination Meets CDCL Solving

Soos et al 2009

- Perform CDCL on CNF Formula
- Perform Gaussian Elimination on XORs
- Exchange Unit/Binary Clauses
- Significantly better than performing CDCL Solving

CryptoMiniSAT: Gaussian Elimination Meets CDCL Solving

Soos et al 2009

- Perform CDCL on CNF Formula
- Perform Gaussian Elimination on XORs
- Exchange Unit/Binary Clauses
- Significantly better than performing CDCL Solving
- But

CryptoMiniSAT: Gaussian Elimination Meets CDCL Solving

Soos et al 2009

- Perform CDCL on CNF Formula
- Perform Gaussian Elimination on XORs
- Exchange Unit/Binary Clauses
- Significantly better than performing CDCL Solving
- But the formula in XORs never benefit from CDCL steps, in particular inprocessing steps

Research Meets Beers

Why not perform CDCL and Gaussian Elimination on the entire problem again and again.

(This research is not supported by an IPA brewery).

Research Meets Beers

Why not perform CDCL and Gaussian Elimination on the entire problem again and again.

- **B**: Blast XORs into CNF

(This research is not supported by an IPA brewery).

Research Meets Beers

Why not perform CDCL and Gaussian Elimination on the entire problem again and again.

- **B**: Blast XORs into CNF
- **I**: Perform CDCL (in particular, **Inprocessing**)

(This research is not supported by an IPA brewery).

Research Meets Beers

Why not perform CDCL and Gaussian Elimination on the entire problem again and again.

- **B**: Blast XORs into CNF
- **I**: Perform CDCL (in particular, **Inprocessing**)
- **R**: Recover XORs from CNF and perform Gaussian Elimination

(This research is not supported by an IPA brewery).

Research Meets Beers

Why not perform CDCL and Gaussian Elimination on the entire problem again and again.

- **B**: Blast XORs into CNF
- **I**: Perform CDCL (in particular, **Inprocessing**)
- **R**: Recover XORs from CNF and perform Gaussian Elimination
- **D**: Destroy the XORs

(This research is not supported by an IPA brewery).

Research Meets Beers

Why not perform CDCL and Gaussian Elimination on the entire problem again and again.

BIRD

- **B**: Blast XORs into CNF
- **I**: Perform CDCL (in particular, **Inprocessing**)
- **R**: Recover XORs from CNF and perform Gaussian Elimination
- **D**: Destroy the XORs
- Loop until SAT or UNSAT

(This research is not supported by an IPA brewery).

Why not perform CDCL and Gaussian Elimination on the entire problem again and again.

BIRD

- **B**: Blast XORs into CNF
- **I**: Perform CDCL (in particular, **Inprocessing**)
- **R**: Recover XORs from CNF and perform Gaussian Elimination
- **D**: Destroy the XORs
- Loop until SAT or UNSAT

(This research is not supported by an IPA brewery).

Challenge 1 Can you recover XORs from CNF efficiently? (Multiple times!)
Faster than reading all the clauses!

Challenge 2 Handling backtracking to avoid repeated Gaussian Elimination

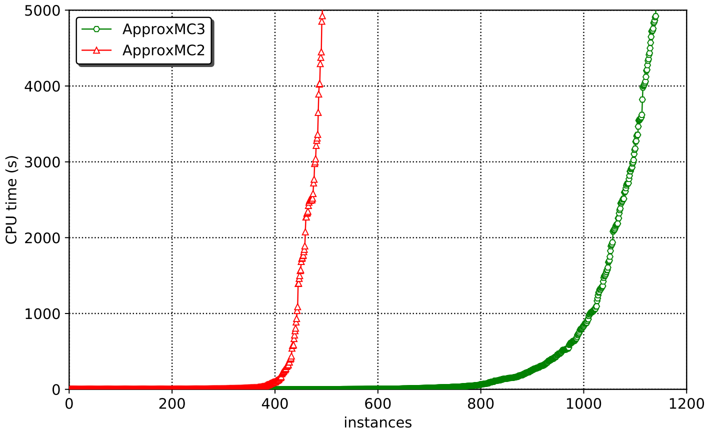
Experimental Results

- ApproxMC3 = ApproxMC2+BIRD
- 1896 Benchmarks from probabilistic reasoning, quantified information flow, program synthesis, functional synthesis....

Experimental Results

- ApproxMC3 = ApproxMC2+BIRD
- 1896 Benchmarks from probabilistic reasoning, quantified information flow, program synthesis, functional synthesis....
- Runtime performance comparison between ApproxMC2 and ApproxMC3
- More results in the paper

Experimental Results-I



ApproxMC3 solves 648 benchmarks more than ApproxMC2

Experimental Results-II

Benchmark	Vars	Clauses	ApproxMC2 time	ApproxMC3 time
or-50-10-9-UC-30	100	260	1814.78	2.66
blasted_squaring28	1060	3839	1845.47	2.48
55.sk_3_46	3128	12145	TO	1.35
s838_7_4	616	1490	TO	3.91
min-3s	431	1373	TO	3.86
blasted_case210	872	2937	TO	5.93
blasted_squaring16	1627	5835	TO	11.12
or-60-5-2-UC-30	120	315	TO	11.09
s5378a_3_2	3679	8372	TO	68.2
modexp8-4-1	79409	288110	TO	255.05
reverse.sk_ 11_ 258	75641	380869	TO	23.4
hash-6	282521	1133816	TO	246.04
modexp8-5-8	101553	402654	TO	1166.54
hash-11-8	518009	2078959	TO	4908.15
karatsuba.sk_7_41	19594	82417	TO	4865.53
01B-3	23393	103379	TO	4275.08

TO: Timeout after 5000 seconds

Mean Speedup of ApproxMC3 over ApproxMC2: 284.40

Conclusion

- CNF+XOR Handling is crucial for approximate model counting (and other hashing-based algorithms)
- Previous architecture for CNF+XOR did not allow in-processing and CDCL over XOR formulas
- We propose a new framework BIRD to have the best of both the worlds: CDCL+Gaussian Elimination
- **Significant** runtime improvement
- Open-source tool `https://tinyurl.com/approxmc`

B: Blast XORs into CNF

$$\begin{matrix} (x_1 \vee x_2 \vee \neg x_3) \\ (x_1 \vee \neg x_2 \vee x_3) \\ (\neg x_1 \vee x_2 \vee x_3) \\ (\neg x_1 \vee \neg x_2 \vee \neg x_3) \end{matrix} \Leftrightarrow x_1 \oplus x_2 \oplus x_3 = 0 \tag{1}$$

$$(x_1 \vee x_2) \text{ subsumes } (x_1 \vee x_2 \vee \neg x_3) \tag{2}$$

$$(x_1 \vee \neg\neg x_2) \text{ subsumes } (x_1 \vee \neg x_2 \vee x_3) \tag{3}$$

R: Recover

$$
\begin{array}{r}
(x_1 \vee x_2 \vee \neg x_3) \\
(x_1 \vee \neg x_2 \vee x_3) \\
(\neg x_1 \vee x_2 \vee x_3) \\
(\neg x_1 \vee \neg x_2 \vee \neg x_3)
\end{array}
\Leftrightarrow \qquad x_1 \oplus x_2 \oplus x_3 = 0 \qquad (4)
$$

R: Recover

$$\begin{aligned}(x_1 \vee x_2)\\(x_1 \vee \neg x_2)\\(\neg x_1 \vee x_2 \vee x_3)\\(\neg x_1 \vee \neg x_2 \vee \neg x_3)\end{aligned} \rightarrow x_1 \oplus x_2 \oplus x_3 = 0 \tag{4}$$