Constraint Networks Presentation

Constrained Sampling and Counting: Universal Hashing Meets SAT Solving

Kuldeep S. Meel, Moshe Y. Vardi, Supratik Chakraborty, Daniel J. Fremont, Sanjit A. Seshia, Dror Fried, Alexander Ivrii, Sharad Malik
Constrained sampling and counting

Counting
- Given a SAT formula F
- \( R_F \): Set of all solution of F
- Problem (\(#\text{SAT})\): Estimate the number of solutions of F, i.e. the cardinality of \( R_F \).
- Example: \( F = (a \lor b) \), \( R_F = \{(0,1), (1,0), (1,1)\} \), \( |R_F| = 3 \)

Sampling
- Sample randomly from \( R_F \) given F.
- Example: sample(F) returns (0,1), (1,0) or (1,1) each with a probability 1/3.

These problems are in \#P, which are much harder than NP complete.
Performance of exact counters

- Cachet – a well known exact counter.
- Performance measured on benchmarks – plan recognition, langford problems, circuit synthesis etc. - [http://www.cs.rice.edu/CS/Verification/Projects/ApproxMC/Benchmarks/](http://www.cs.rice.edu/CS/Verification/Projects/ApproxMC/Benchmarks/)
- Approximate counting with guarantees suffices for most of the applications.

Benchmarks range from 32 to 229,100 variables in CNF representation.

Cachet solves upto 13,000 variables.
Prior work on Model Counting

- Earliest approaches based on DPLL style sat solvers.
- Relsat, Cachet and sharpSAT improved on this by optimizations - clause learning, look ahead etc.
- Karp and Luby presented a fully polynomial randomized approximation scheme for counting models of DNF formula (1989)
- Although exact counting for DNF and CNF are polynomially inter-reducible, there in no known polynomial reduction for the corresponding approximate counting problems.
- Lower bound counters like Mbound (2006), SampleCount and Upper bound counters like MiniCount give confidence intervals but very weak count guarantees.
- Hashing based approaches were pioneered by Sipser (1983) used in treatments of counting and near uniform sampling.
Approximate Model counting

- Design an approximate model counter $G$
- Inputs:
  - A CNF formula $F$
  - tolerance $\varepsilon$
  - confidence $\delta$
- Guarantees:

\[
Pr \left[ \frac{|R_f|}{1+\varepsilon} \leq \text{ApproxCount} \left( F, \varepsilon, 1-\delta \right) \leq (1+\varepsilon) \times |R_f| \right] \geq 1-\delta
\]

ApproxMC – the first scalable approximate model counter
Core Idea

- Counting all the solutions is hard
- Make the problem more constrained
- Find the solutions of the new problem
- Use that solution to get an estimate of the original problem
- Hashing of the input variables is a good way to partition the input space.

- A family of hash functions selected by randomly picking and xor'ing input variables is shown to be a 3-universal hash function (Gomes, Sabharwal and Selman 2007).

- Example: Given \((X_1 \lor X_2 \lor X_3 \lor X_4)\) the condition \(X_1 \oplus X_3 = 0\) splits the solution space into half.
Core Idea

Solution Space

Constrained Space

Counting through partitioning

- Pick a random cell
- Total # of solutions =
  
  #of solutions in the cell * total #of cells
How to partition

- How to partition into roughly equal small cells of solutions without knowing the distribution of solutions
- Hash functions mapping $\{0,1\}^n$ to $\{0,1\}^m$ ($2^n$ elements to $2^m$ cells)
- Universal hash functions:
  - (any distribution) inputs $\rightarrow$ All cells are roughly small
- $H(n,m,r)$: Family of $r$-universal hash functions
  - Higher the $r$ $\rightarrow$ Stronger guarantees on range of size of cells
  - Lower universality $\rightarrow$ lower complexity
- Employ XOR-based hash functions instead of computationally infeasible algebraic hash functions
- Uses off-the-shelf SAT solver CryptoMiniSAT (MiniSAT+XOR support)
Algorithm

A Scalable Approximate Model Counter: Supratik Chakraborty, Kuldeep S. Meel, and Moshe Y. Vardi. Indian Institute of Technology Bombay, Rice University

- Use hashing functions from the $H_{\text{xor}}(n, m, 3)$ family for an appropriate $m$.
- Choose a random cell and check if it is non-empty and has no more than $\text{pivot}(\varepsilon)$ elements.
- If the chosen cell is not small, we randomly partition the set of models into twice as many cells as before by choosing a random hashing function from the family $H_{\text{xor}}(n, m + 1, 3)$.
- The above procedure is repeated until either a randomly chosen cell is found to be non-empty and small, or the number of cells exceeds $\sim 2^{n+1}$.
- The size of the cell last chosen is scaled by the number of cells to obtain an $\varepsilon$-approximate estimate of the model count.
- The procedure is repeated to get the desired confidence ($\delta$).
Algorithm ApproxMC(F, ?δ?)

\[
\text{counter} \leftarrow 0; \text{C} \leftarrow \text{emptyList}; \\
\text{pivot} \leftarrow 2 \times \text{ComputeThreshold}(\varepsilon); \\
\text{t} \leftarrow \text{ComputeIterCount}(\delta); \\
\text{repeat}: \\
\quad \text{c} \leftarrow \text{ApproxMCCore}(F, \text{pivot}); \\
\quad \text{counter} \leftarrow \text{counter} + 1; \\
\quad \text{if} (\text{c} \neq \bot) \\
\quad \quad \text{AddToList(C, c);} \\
\text{until} (\text{counter} < \text{t}); \\
\text{finalCount} \leftarrow \text{FindMedian(C);} \\
\text{return finalCount;}
\]

Algorithm ApproxMCCore(F, pivot)

\[
\text{S} \leftarrow \text{BoundedSAT}(F, \text{pivot} + 1); \\
\text{if} (|S| \leq \text{pivot}) \\
\quad \text{return } |S|; \\
\text{else} \\
\quad \text{l} \leftarrow \lfloor \log_2 (\text{pivot}) \rfloor - 1; \text{i} \leftarrow \text{l} - 1; \\
\quad \text{repeat} \\
\quad \quad \text{i} \leftarrow \text{i} + 1; \\
\quad \quad \text{Choose } h \text{ at random from } H_{\text{xor}}(n, i - l, 3); \\
\quad \quad \text{Choose } \alpha \text{ at random from } \{0, 1\}^{i-l}; \\
\quad \quad \text{S} \leftarrow \text{BoundedSAT}(F \land \text{ }(h(z_1, \ldots, z_n) = \alpha), \text{pivot} + 1); \\
\quad \text{until} (1 \leq |S| \leq \text{pivot}) \text{ or } (i = \text{n}); \\
\quad \text{if} (|S| > \text{pivot} \text{ or } |S| = 0) \text{return } \bot; \\
\quad \text{else return } |S| \cdot 2^{i-l};
\]

Algorithm ComputeThreshold(\varepsilon)

\[
\text{return } \left\lceil 3e^{0.5 \left(1 + \frac{1}{\varepsilon}\right)^2} \right\rceil
\]

Algorithm ComputeIterCount(\delta)

\[
\text{return } \left\lceil 35 \log \left(\frac{3}{\delta}\right) \right\rceil
\]

ApproxMC runs in time polynomial in \(\log (1 - \delta)^{-1}, |F|, \varepsilon^{-1}\) relative to SAT oracle

3/8/2016
Results

Objectives: Compare running time and quality of Bounds

Tolerance: \( \varepsilon = 0.75 \), Confidence: \( \delta = 0.9 \)
Results

Mean error is 4% which is much smaller than the theoretical guarantee of 75%.

Range of count from bounding counters $C_2 - C_1$

$C_1$: From lower bound counters (MBound/SampleSAT)

$C_2$: From upper bound counters (MiniCount)

Range from ApproxMC:

$[C/(1+\varepsilon), (1+\varepsilon)C]$

Smaller the range, better the algorithm

3/8/2016
Near uniform generator

A Scalable and Nearly Uniform Generator of SAT Witnesses: Supratik Chakraborty, Kuldeep S. Meel, and Moshe Y. Vardi. Indian Institute of Technology Bombay, Rice University

- Design an near uniform generator $G^{au}$
- Inputs: A CNF formula $F$
  - A CNF formula $F$
  - tolerance $\epsilon > 0$
- Guarantees:

$$\forall y \in R_f, \frac{1}{(1+\epsilon)|R_f|} \leq Pr[G^{au}(F, \epsilon)] \leq \frac{1+\epsilon}{|R_f|}$$
Algorithm

- The algorithm first calls ApproxMC to get an approximate count of the size of the solution space.
- This enables to fine-tune the parameters for using XOR constraints to partition the solution space.
- A randomly chosen cell is expected to be small enough so the solutions in it can be enumerated and sampled.
- Example: if we have $2^s$ count of solutions, we need to split the solution space with $s$ random constraints.
Results

- Able to handle formulas with approximately 0.5M variables.

- Compared with XORSample’ (Gomes, Sabharwal, and Selman 2007), which does not offer an approximation guarantee of almost uniformity.

- Compared UniGen with a uniform sampler on a benchmark with about 16K solutions.

- Generated 4M samples, each from US and UniGen; the resulting distributions were statistically indistinguishable
Future work

- Parallelism – increasing runtime performance without sacrificing theoretical guarantees
- Using wasted solutions in sampling
- Weighted counting
- Extensions to SMT
- More efficient hash functions
- Probabilistic Inference
Thank you