The Secrets of GANAK: Designing Scalable Exact Model Counter

Kuldeep S. Meel[1]

Joint work with Shubham Sharma[2], Mate Soos[1], and Subhajit Roy[2]

[1]National University of Singapore
[2]Indian Institute of Technology Kanpur, India

Ganak+ApproxMC won two out of three tracks at Model Counting Competition. Related Paper: IJCAI 2019

Propositional Model Counting

- Given:
  - Propositional formula F (CNF) over a set of variables X
- Propositional Model Counting (#SAT):
  - Compute the number of satisfying assignments of F
- #SAT is #P complete problem

Propositional Model Counting

- Probabilistic Exact Model Counting
  - Given a propositional formula F (CNF) and confidence $\delta \in (0, 1]$, counter returns $\mathtt{count}$ such that:

$$\mathsf{Pr}\Big[|\mathsf{Solutions\ of\ F}| = \mathtt{count}\Big] \geq 1 - \delta$$

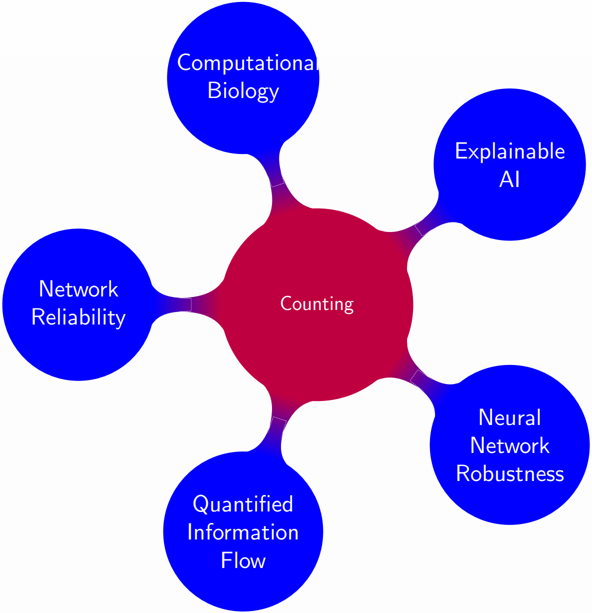
Propositional Model Counting

- Probabilistic Exact Model Counting
  - Given a propositional formula F (CNF) and confidence $\delta \in (0, 1]$, counter returns count such that:

$$\mathsf{Pr}\Big[|\text{Solutions of F}| = \mathtt{count}\Big] \geq 1 - \delta$$

- Probabilistic Exact Model Counting is almost as hard as Exact Model Counting[1]

Applications across Computer Science



Long Line of Work

Knowledge Compilation c2d [Darwiche, 2004], D4[Lagniez and Marquis, 2017]

Search-based Counters Cachet[Sang et al, 2004; 2005], sharpSAT [Thurley 2006]

Hashing-based Counting Stockmeyer 1983, Gomes et al. 2006, Chakraborty et al. 2013-,

Main Ingredients of Search-Based Counters

- Decision Process:
  - $(F \wedge l) \vee (F \wedge \neg l)$ mutually inconsistent
  - $\#(F) = \#(F \wedge l) + \#(F \wedge \neg l)$

Main Ingredients of Search-Based Counters

- Decision Process:
  - $(F \wedge l) \vee (F \wedge \neg l)$ mutually inconsistent
  - $\#(F) = \#(F \wedge l) + \#(F \wedge \neg l)$
- Component Decomposition:
  - $F = \Delta_1 \wedge \Delta_2 \cdots \Delta_n$ $\Delta_1 \cdots \Delta_n$ does not share any variables
  - $\#(F) = \#(\Delta_1) \times \#(\Delta_2) \cdots \times \#(\Delta_n)$ mutually disjoint

Main Ingredients of Search-Based Counters

- Decision Process:
  - $(F \wedge l) \vee (F \wedge \neg l)$ mutually inconsistent
  - $\#(F) = \#(F \wedge l) + \#(F \wedge \neg l)$
- Component Decomposition:
  - $F = \Delta_1 \wedge \Delta_2 \cdots \Delta_n$ $\Delta_1 \cdots \Delta_n$ does not share any variables
  - $\#(F) = \#(\Delta_1) \times \#(\Delta_2) \cdots \times \#(\Delta_n)$ mutually disjoint
- Conflict Driven Clause Learning

Main Ingredients of Search-Based Counters

- Decision Process:
  - $(F \wedge l) \vee (F \wedge \neg l)$ mutually inconsistent
  - $\#(F) = \#(F \wedge l) + \#(F \wedge \neg l)$
- Component Decomposition:
  - $F = \Delta_1 \wedge \Delta_2 \cdots \Delta_n$ $\Delta_1 \cdots \Delta_n$ does not share any variables
  - $\#(F) = \#(\Delta_1) \times \#(\Delta_2) \cdots \times \#(\Delta_n)$ mutually disjoint
- Conflict Driven Clause Learning
- Component Caching:

Key	Value
Δ_1	$\#(\Delta_1)$
Δ_2	$\#(\Delta_2)$

Model Counting Algorithm

```
l ← DecideLiteral(F)
for lit ← {l, ¬l} do
    F|lit ← UnitPropagation(F, lit)
    if F|lit contains an empty clause then
        count[lit] ← 0
    else
        count[lit] ← 1
        Comps ← DisjointComponents(F|lit)              ▷ Decomposition
        for C ← Comps do
            count ← GetCache(C)
            if count = NOT FOUND then
                count ← Counter(C)
            count[lit] = count[lit] × count
            if count = 0 then
                break
CacheStore(F, count[l] + count[¬l])
return count[l] + count[¬l]
```

Example

$$\boxed{F = (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_4 \vee x_5) \wedge (\neg x_1 \vee x_2 \vee x_3)}$$

Example

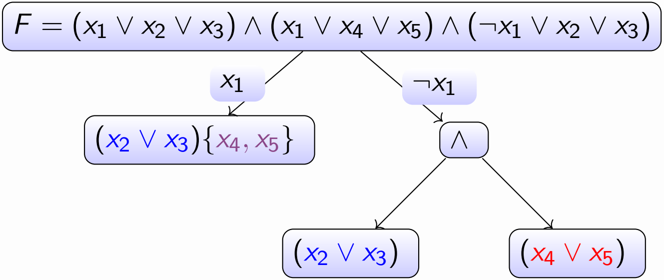
$$F = (x_1 \lor x_2 \lor x_3) \land (x_1 \lor x_4 \lor x_5) \land (\neg x_1 \lor x_2 \lor x_3)$$

$$\Big\downarrow x_1$$

$$(x_2 \lor x_3)\{x_4, x_5\}$$

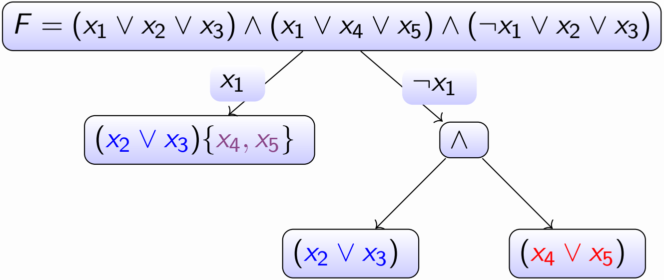
Key	Value
$(x_2 \lor x_3)$	3
$(x_2 \lor x_3)\{x_4, x_5\}$	12

Example



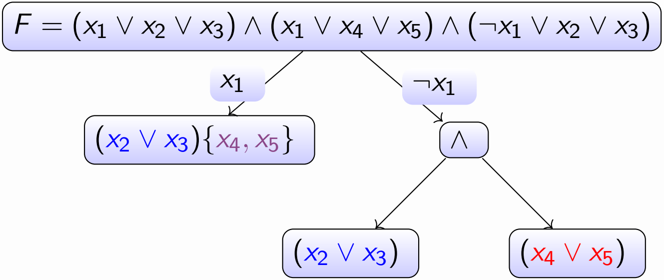
Key	Value
$(x_2 \vee x_3)$	3
$(x_2 \vee x_3)\{x_4, x_5\}$	12

Example



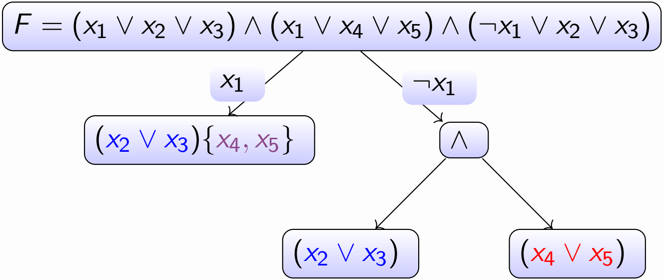
Key	Value
$(x_2 \vee x_3)$	3
$(x_2 \vee x_3)\{x_4, x_5\}$	12
$(x_4 \vee x_5)$	3

Example



Key	Value
$(x_2 \vee x_3)$	3
$(x_2 \vee x_3)\{x_4, x_5\}$	12
$(x_4 \vee x_5)$	3
$(x_2 \vee x_3) \wedge (x_4 \vee x_5)$	9

Example



Key	Value
$(x_2 \vee x_3)$	3
$(x_2 \vee x_3)\{x_4, x_5\}$	12
$(x_4 \vee x_5)$	3
$(x_2 \vee x_3) \wedge (x_4 \vee x_5)$	9
$F = (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_4 \vee x_5) \wedge (\neg x_1 \vee x_2 \vee x_3)$	21

Our Contribution

1. Probabilistic Component Caching (PCC)
2. Variable Branching Heuristic (CSVSADS)
3. Phase Selection Heuristic (PC)
4. Independent Support (IS)
5. Learn and Start Over (LSO)

Probabilistic Component Caching (PCC)

$$F = (\neg x_3 \vee \neg x_5 \vee x_6) \wedge (\neg x_1 \vee x_4 \vee \neg x_6) \wedge (x_2 \vee x_3 \vee x_6)$$

Schema	Key	Value
STD^2	-3, -5, 6, 0, -1, 4, -6, 0, 2, 3, 6, 0	$\#(F)$
HC^3	1, 2, 3, 4, 5, 6, 1, 2, 3	$\#(F)$
GANAK	m bit hash of HC/STD clhash: universal hash functions	$\#(F)$

Variable Branching Heuristic (CSVSADS)

- Score(VSADS)[4] $= \underline{\text{p} \times \text{Score(VSIDS)}} + \underline{\text{q} \times \text{Score(DLCS)}}$
  - VSIDS: Prioritize variables present in recently generated conflict clauses
  - Dynamic Largest Com-bined Sum(DLCS): Prioritize the highest occurring variable in the residual formula

Variable Branching Heuristic (CSVSADS)

- Score(VSADS)[4] $= \underline{\text{p} \times \text{Score(VSIDS)}} + \underline{\text{q} \times \text{Score(DLCS)}}$
  - VSIDS: Prioritize variables present in recently generated conflict clauses
  - Dynamic Largest Com-bined Sum(DLCS): Prioritize the highest occurring variable in the residual formula
- Score(CSVSADS) $= \underline{\alpha \times \text{CacheScore}} + \underline{\beta \times \text{Score(VSADS)}}$

Phase Selection Heuristic (PC)

- $$\text{DLIS} = \begin{cases} l & |l| \geq |\neg l| \\ \neg l & \textit{otherwise} \end{cases}$$

Phase Selection Heuristic (PC)

- $$\text{DLIS} = \begin{cases} l & |l| \geq |\neg l| \\ \neg l & \textit{otherwise} \end{cases}$$
- We reduce our trust on DLIS by adding randomness in DLIS if the difference in $|l|$ and $|\neg l|$ is not overwhelmingly high

Independent Support (IS)

- An independent support, $\mathcal{I} \subseteq \mathsf{Vars}(F)$, is a subset of the support such that if two satisfying assignments σ_1 and σ_2 agree on $\mathcal{I}$, then $\sigma_1 = \sigma_2$

Independent Support (IS)

- An independent support, $\mathcal{I} \subseteq \mathsf{Vars}(F)$, is a subset of the support such that if two satisfying assignments σ_1 and σ_2 agree on $\mathcal{I}$, then $\sigma_1 = \sigma_2$
- Example: $(x \vee \neg y) \wedge (\neg x \vee y) \qquad \mathcal{I} = \{x\}$

Independent Support (IS)

- An independent support, $\mathcal{I} \subseteq \mathsf{Vars}(F)$, is a subset of the support such that if two satisfying assignments σ_1 and σ_2 agree on $\mathcal{I}$, then $\sigma_1 = \sigma_2$
- Example: $(x \vee \neg y) \wedge (\neg x \vee y) \qquad \mathcal{I} = \{x\}$
- We use the MIS[5] algorithm for computing the *minimal* $\mathcal{I}$ for hard instances

Independent Support (IS)

- An independent support, $\mathcal{I} \subseteq \mathsf{Vars}(F)$, is a subset of the support such that if two satisfying assignments σ_1 and σ_2 agree on $\mathcal{I}$, then $\sigma_1 = \sigma_2$
- Example: $(x \vee \neg y) \wedge (\neg x \vee y) \qquad \mathcal{I} = \{x\}$
- We use the MIS[5] algorithm for computing the *minimal* $\mathcal{I}$ for hard instances
- Perform decision process on variables from $\mathcal{I}$
  1. If residual formula is SAT – model count equal to 1
  2. If residual formula is UNSAT – model count equal to 0

Learn and Start Over (LSO)

- Modern SAT solvers use random restarts aggressively in search of a good variable ordering that can quickly lead to a satisfiable assignment[6]

Learn and Start Over (LSO)

- Modern SAT solvers use random restarts aggressively in search of a good variable ordering that can quickly lead to a satisfiable assignment[6]
- Restart solver after the first 5000 decisions
- Learn from the previous invocation by maintaining all the scores obtained in the previous run to explore different and better ordering of decision variables

Tool

- GANAK[7]: First Scalable Probabilistic Exact Model Counter
- Given a propositional formula F (CNF) and confidence $\delta \in (0, 1]$ GANAK(F, δ) returns $\texttt{count}$ such that

$$\Pr\left[|Sol(F)| = \texttt{count}\right] \geq 1 - \delta$$

- Tool is available at: https://github.com/meelgroup/ganak

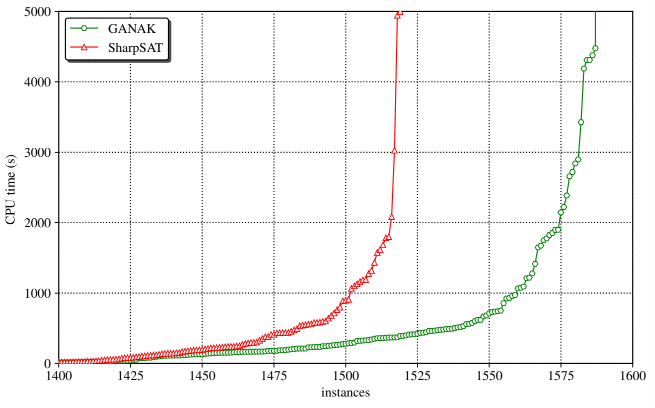
Experimental Evaluation

- Benchmarks arising from probabilistic reasoning, plan recognition, DQMR networks, ISCAS89 combinatorial circuits, quantified information flow, etc

Experimental Evaluation

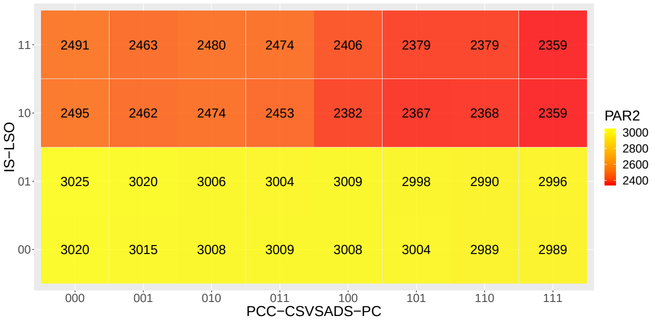
- Benchmarks arising from probabilistic reasoning, plan recognition, DQMR networks, ISCAS89 combinatorial circuits, quantified information flow, etc
- Objectives:
  1. Study the impact of different configurations of heuristics Shubham's Talk
  2. Study the performance of GANAK with respect to the state-of-the-art model counters

Experimental Evaluation: Comparison with other tools



- In our experiments, the model count returned by GANAK was equal to the exact model count for all benchmarks

Experimental Evaluation: Individual Analysis



- GANAK performed best when all the heuristics are turned on

Conclusion

- GANAK demostrates that #SAT solvers can significantly benefit from probabibistic component caches, especially when ably supported by heuristics like IS, CSVSADS, PC and LSO
- We believe that the heuristics proposed in this work will also significantly benefit exhaustive DPLL-based knowledge compilation frameworks and related tools (like c2d [Darwiche, 2004], D4 [Lagniez and Marquis, 2017], DSHARP [Muise et al., 2012])
- Tool is available at: https://github.com/meelgroup/ganak