Engineering an Efficient Probabilistic Exact Model Counter

Mate Soos, Kuldeep S. Meel

University of Toronto, Georgia Tech

24th of July 2025, Zagreb, CAV 2025

Outline

Propositional Model Counting

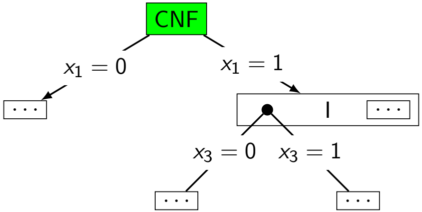
- Count the number of solutions to a CNF, e.g $(x_1 \vee x_2) \wedge (\neg x_1 \vee x_3)$
- Notice that finding a single solution is in NP. Counting is $\#$P
- Sometimes, we project the solution space, like so:

```
Projection set | Other variables
--------------------------------
0001           |010111
0001           |101010
1011           |111010
```

  This problem only has 2 solutions over the projection set (but 3 overall)
- Sometimes, literals in the projection set have a “weight”
- Weight is usually a rational number, but it can be any element of a semiring
- Then, each solution has a “weight”: the product of each literal's weight
- Final solution in this case is the sum of each solution's weight, i.e. **sum-of-products**

Our Contributions to d-DNNF Model Counting

We count solutions to the CNF by compiling the CNF into a **d-DNNF**[Darwiche&Marquis, 2002]: a deterministic decomposable negation normal form circuit.



1. **Enhanced Residual Formula Processing**: optimized SAT solver architecture for residual formula processing that incorporates VSIDS scoring, restarts, and polarity caching.
2. **Dual Independent Set Framework**: maintains distinct SAT-eligibility (S) and decision (D) sets, where the S-set determines SAT solver transitions while the D-set guides branching decisions.
3. **Chronological Backtracking**: Adaptation of chronological backtracking to model counting

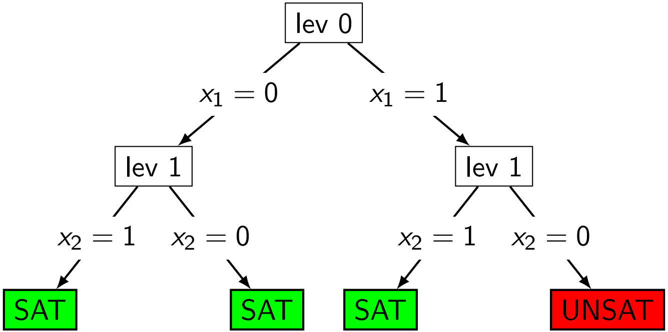
Calling a SAT solver

Once all variables in P have been decided there is only one solution: we can run a SAT solver!

$V = \{x_1, x_2, x_3, x_4, \ldots\}$
$P = \{x_1, x_2\}$

Once $\{x_1, x_2\}$ have all been branched on, we can call a SAT solver on the residual formula!

This has been exploited before by gpmc



Enhanced Residual Formula Processing

We added:

- **VSIDS Scoring**: Model counters use VSADS[1] scoring, but for SAT, VSIDS is faster to compute, and more efficient.
- **Restarts**: Restart the SAT solver regularly (Luby heuristic[2]), to explore different parts of the search space. Note we only need one solution.
- **Polarity Caching**[3]: We cache the polarity of variables to avoid unnecessary recomputation, especially due to restarts

Basically: let's lift all the SAT techniques to model counting's residual formula processing.

[1] Variable State Aware Decaying Sum, from Sang et al.'05
[2] Luby et al, '93
[3] Pipatsrisawat et al, '07

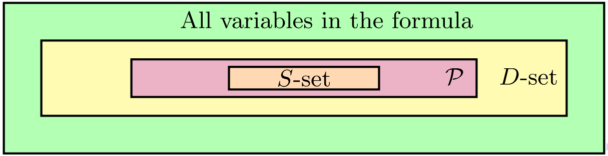
Dual Independent Set

Definition (Dual Independent Set)

Given a Boolean formula F defined over variables V and a projection set $\mathcal{P}$, a dual independent set consists of:

1. **S-set** A SAT-eligibility set $S \subseteq \mathcal{P}$ that determines when SAT solver mode transition is permissible.
2. **D-set** A decision set $D \subseteq V$ that guides branching variable selection, where $D \supseteq \mathcal{P}$.

This formulation generalizes traditional approaches where $D = S \subseteq P$, instead aiming for $S \subseteq P \subseteq D$.



Computing Maximal Decision Set

$D \leftarrow \mathcal{P}$, $G \leftarrow \textsc{ExtractGates}(F)$ ▷ Initialize D with $\mathcal{P}$, Extract gates

procedure $\textsc{SyntacticExpansion}(V)$

  changed $\leftarrow V$

  while changed is not empty **do**

    $v \leftarrow$ changed.*pop*()

    for gate $g \in G$ where v is input **do** If possible, extend D with output of the gate

  return D

end procedure

procedure $\textsc{SemanticExpansion}$

  for $v \notin D$ **do**

    if $\textsc{ValidateDecisionVar}(D \cup \{v\}, F)$ **then**

      $D \leftarrow D \cup \{v\}$

      $D \leftarrow \textsc{SyntacticExpansion}(\{v\})$ ▷ Quick check with syntactic analysis

  return D

end procedure

We run $\textsc{SyntacticExpansion}(P)$ and then $\textsc{SemanticExpansion}$

Chronological Backtracking

Notice that we should NOT backjump or we lose already counted solutions.

- Model counters go back to the deepest level possible, flip the decision, and throw away the learnt clause in case it would violate propagation invariants
- But we can relax some of the invariants: Chronological Backtracking (ChronoBT)
- ChronoBT allows the have out-of-order implication levels in the trail
- ChronoBT allows us to keep the learnt clause, and force the literal that the learnt clause entails.
- We used the fuzzer `SharpVelvet` by Latour et al. to find bugs in our

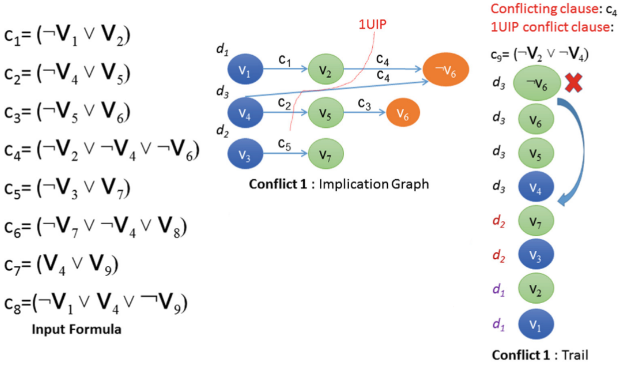
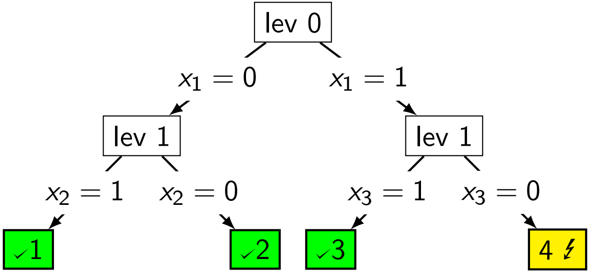


Figure from Chronological Backtracking by Nadel&Ryvchin, 2018

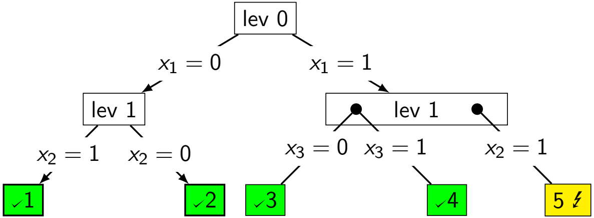
Compensating Weights

1. We explore the left side of a graph, nodes 1 and 2
2. We backtrack to level 0 and explore the right side, nodes 3 and 4.
3. At node 4, we find the unit clause x_3. This unit clause's level is 0, but due to ChronoBT, we only backtrack to level 1. However, the system **has already multiplied in the weight of $\mathbf{x_3}$** into nodes 1, 2, and 3.
4. These nodes' weights, which on the left side of an already explored branch need to be compensated



Compensating Weights

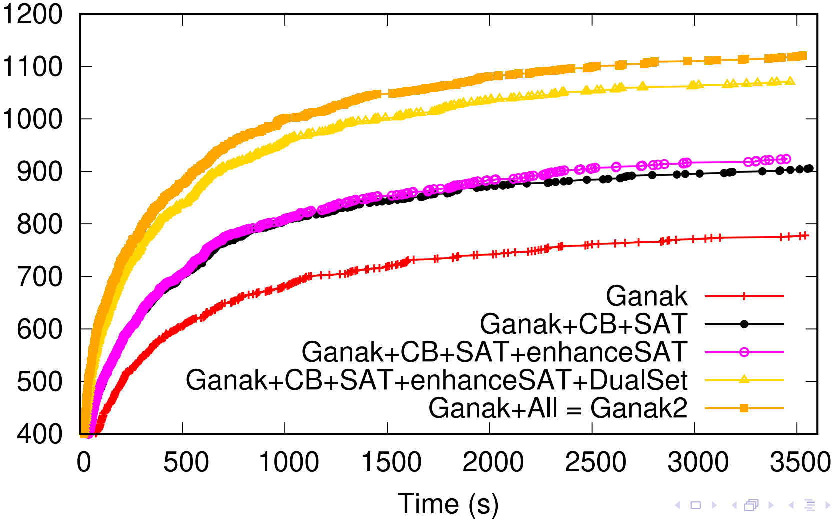
1. x_4 is part of components $\{1,2,3,4\}$ but not 5 is learned to be false at level 0.
2. Sounds impossible. x_4 is clearly not part of component 5 (since it is part of 3 and 4), so it should never be part of a learned clause while examining component 5
3. Components are decided *purely* based on irredundant clauses. It is possible that learned clauses connect components!
4. Can lead to contradictions over variables *not* part of the component examined!
5. Compensating for these is non-trivial: *sibling* components need to be examined and compensated for



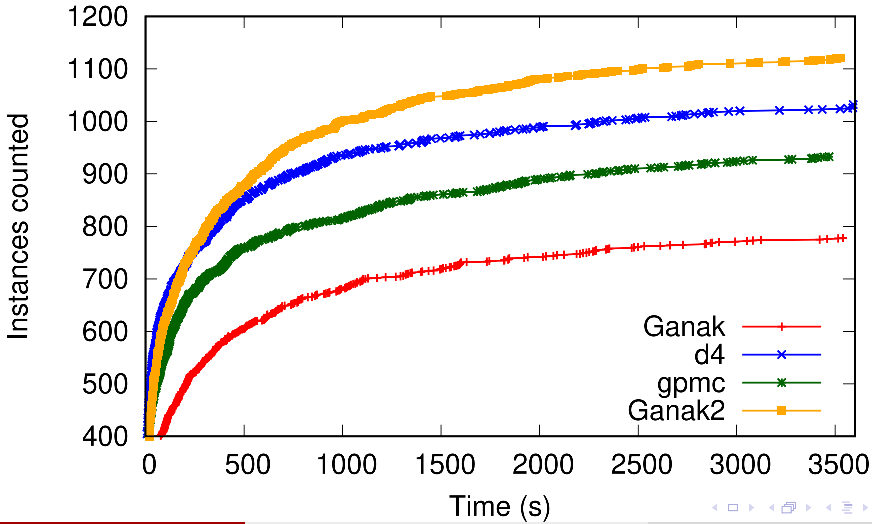
Compensating Weights

- Notice that to compensate we need to **divide**
- But weighted counting can be done with weights over any **semiring**
- But Ganak2 needs a **field**
- In some sense, we reorder computations, exploiting flexibility in building the d-DNNF
- Side-note: Ganak2 supports **any field**. Implementing a new field should take ≈ 10 minutes. Currently supports:
  - Integers
  - Counting modulo prime
  - Rational numbers
  - Floating point – not a field actually
  - Complex rational numbers
  - Multivariate polynomials over the rationals

Ganak Versions over all Instances



All Instances, All Solvers



Conclusions

We created a new probabilistically exact model counter, Ganak2, that incorporates:

- **Enhanced Residual Formula Processing**
- **Dual Independent Set Framework**
- **Chronological Backtracking**

This allowed us to achieve state-of-the-art performance. Ganak2 won all tracks of the Model Counting Competition 2024.

Code: Paper:

Also, try it out online: `https://www.msoos.org/ganak/`