DPMC: Weighted Model Counting by Dynamic Programming on Project-Join Trees

Jeffrey M. Dudek, **Vu H. N. Phan** (speaker), **Moshe Y. Vardi**
{jmd11, vhp1, vardi}@rice.edu — Rice University

CP 2020: 3-minute overview + 17-minute technical part

Abstract

- *Unifying* dynamic-programming framework for exact literal-weighted model counting
- Faster than weighted model counters `cachet`, `miniC2D`, `c2d`, and `d4` on 584 of 1976 benchmarks

3-Minute Overview

CNF formula $\varphi = (x_1 \lor x_3) \land (x_1 \lor \neg x_2) \land (x_2 \lor \neg x_3)$

Planning phase $\downarrow$

Project-join tree

$$\textstyle\sum_{\{x_2, x_3\}}$$

$$\textstyle\sum_{\{x_1\}} \qquad x_2 \lor \neg x_3$$

$$x_1 \lor x_3 \qquad x_1 \lor \neg x_2$$

Execution phase $\downarrow$

Model count $\#\varphi = 4$

Model Counting

Model counting (#SAT): computing number of satisfying assignments of Boolean formula

- Complexity: #P-complete (Valiant, 1979)
- Numerous applications, especially in probabilistic reasoning
    - Medical diagnosis (Shwe et al., 1991)
    - Reliability analysis of power transmission (Duenas-Osorio et al., 2017)

Model Counting with Dynamic Programming

Formula in conjunctive normal form (CNF):

$$\varphi = \text{clause1} \wedge \text{clause2} \wedge \text{clause3} = (x_1 \vee x_3) \wedge (x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3)$$

Bottom-up dynamic programming for model counting:

$$\varphi$$

$$\downarrow$$

$$\sum_{\{x_2, x_3\}}$$

$$\sum_{\{x_1\}} \qquad x_2 \vee \neg x_3$$

$$x_1 \vee x_3 \qquad x_1 \vee \neg x_2$$

$$\downarrow$$

Model count $\#\varphi = 4$

Model Counting with Project-Join Tree Planning and Execution

CNF formula $\varphi = (x_1 \vee x_3) \wedge (x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3)$

Planning phase $\downarrow$

Project-join tree

$\sum_{\{x_2, x_3\}}$

$\sum_{\{x_1\}}$ $\quad$ $x_2 \vee \neg x_3$

$x_1 \vee x_3$ $\quad$ $x_1 \vee \neg x_2$

Execution phase $\downarrow$

Model count $\#\varphi = 4$

Contributions: Framework and Implementation (End of Overview)

Planner: CNF formula $\mapsto$ *project-join tree*

$\downarrow$ `Model counter` (*planner*+**executor**)

Executor: project-join tree $\mapsto$ **model count**

One-shot constraint-programming (CP) heuristics | *Anytime tree decompositions*

`ADDMC` `Crossover1` `Crossover2` `TensorOrder`

Sparse algebraic decision diagrams (ADDs) | **Dense tensors**

Performance: `Crossover1` (new) $>$ `ADDMC` $>$ `TensorOrder` $>$ `Crossover2` (new)

Source code and experimental data: https://github.com/vardigroup/DPMC

Part II

17-Minute Technical Part

HTB *planner:* *one-shot CP heuristics*

LG *planner:* *anytime tree decompositions*

ADDMC

Crossover1

Crossover2

TensorOrder

DMC **executor:** **sparse ADDs**

tensor **executor:** **dense tensors**

Progress

1. Boolean Model-Counting Problem (#SAT)
2. Dynamic Programming for Model Counting with Project-Join Trees
3. Planning Phase: Constructing Project-Join Trees
   - `HTB`: *One-Shot* CP Heuristics
   - `LG`: *Anytime* Tree Decompositions
4. Execution Phase: Valuating Project-Join Trees
   - `DMC`: *Sparse* Algebraic Decision Diagrams (ADDs)
   - `tensor`: *Dense* Tensors
5. Experimental Evaluation

Problem: Model Counting

Boolean formula in conjunctive normal form (CNF): $\varphi = (x_1 \vee x_3) \wedge (x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3)$

Table 1: Truth table of Boolean variable assignments α and corresponding evaluations $\varphi(\alpha)$.

Assignment $\alpha \in 2^{\texttt{Vars}(\varphi)}$			Boolean function $\varphi(\alpha) : 2^{\texttt{Vars}(\varphi)} \to \mathbb{B}$	Is α a **model** of φ?
x_1	x_2	x_3		
0	0	0	0	Yes iff $\varphi(\alpha) = 1$
0	0	1	1	
0	1	0	0	
0	1	1	0	
1	0	0	0	
1	0	1	1	
1	1	0	1	
1	1	1	1	

Model count (unweighted): $\#\varphi = \sum_{\alpha \in 2^{\texttt{Vars}(\varphi)}} \varphi(\alpha) = 4$

Related Work: Model Counting

Existing approaches and tools:

1. Search: DPLL-based exploration of solution space
   - `cachet`: component caching and clause learning (Sang et al., 2004)
2. Knowledge compilation: efficient data structures
   - `miniC2D`: sentential decision diagrams (Oztok & Darwiche, 2015)
   - `c2d`: deterministic decomposable negation normal form (Darwiche, 2004)
   - `d4`: decision decomposable negation normal form (Lagniez & Marquis, 2017)
3. Dynamic programming: solving overlapping subproblems
   - `ADDMC`: algebraic decision diagrams (Dudek, Phan, & Vardi, 2020)
   - `TensorOrder`: tensor networks (Dudek, Dueñas-Osorio, & Vardi, 2019)
   - `dpdb`: database tables (Fichte et al., 2020)

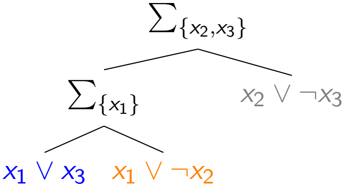
Contribution: *unifying* framework for model counting with dynamic programming using *project-join trees*

Progress

Project-Join Trees for Model Counting with Dynamic Programming

Formula in conjunctive normal form (CNF): $\varphi = (x_1 \vee x_3) \wedge (x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3)$

Project-join tree



Mappings

`Clause`	: leaf node	$\mapsto$	clause
`Label`	: internal node	$\mapsto$	$\sum$-variables

Bottom-up **valuation** $\mathtt{val}(n)$ of node n of project-join tree:

- Leaf node: corresponding clause, interpreted as pseudo-Boolean function $2^{\{x,x'\}} \to \mathbb{R}$
$$\mathtt{val}(n) = \mathtt{Clause}(n)$$
- Internal node: product of valuations of children, followed by projection of $\sum$-variables
$$\mathtt{val}(n) = \sum_{\mathtt{Label}(n)} \left(\prod_{q \in \mathtt{Children}(n)} \mathtt{val}(q) \right)$$

Contributions: Dynamic-Programming Framework and Implementation

Planner: CNF formula $\mapsto$ *project-join tree*

$\downarrow$ `Model counter` (*planner*+**executor**)

Executor: project-join tree $\mapsto$ **model count**

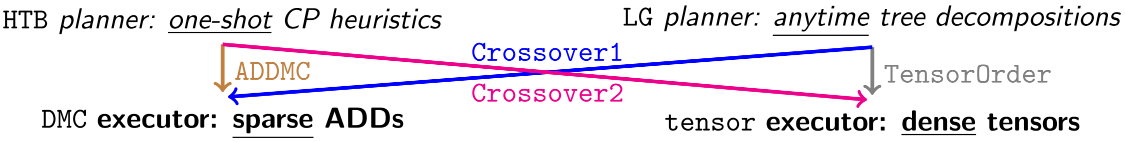


Figure 1: `DPMC dynamic-programming model-counting framework`

Advantages of decoupling planning and execution phases:

- New crossover model counters, which may outperform existing model counters
- Separate development of planning and execution algorithms

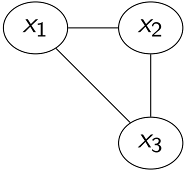
Progress

1. Boolean Model-Counting Problem (#SAT)
2. Dynamic Programming for Model Counting with Project-Join Trees
3. Planning Phase: Constructing Project-Join Trees
   - `HTB`: *One-Shot* CP Heuristics
   - `LG`: *Anytime* Tree Decompositions
4. Execution Phase: Valuating Project-Join Trees
   - `DMC`: *Sparse* Algebraic Decision Diagrams (ADDs)
   - `tensor`: *Dense* Tensors
5. Experimental Evaluation

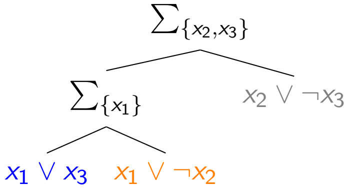
Planner HTB with *One-Shot* CP Heuristics

CNF formula: $(x_1 \lor x_3) \land (x_1 \lor \neg x_2) \land (x_2 \lor \neg x_3)$

Gaifman graph (primal constraint graph)



Project-join tree

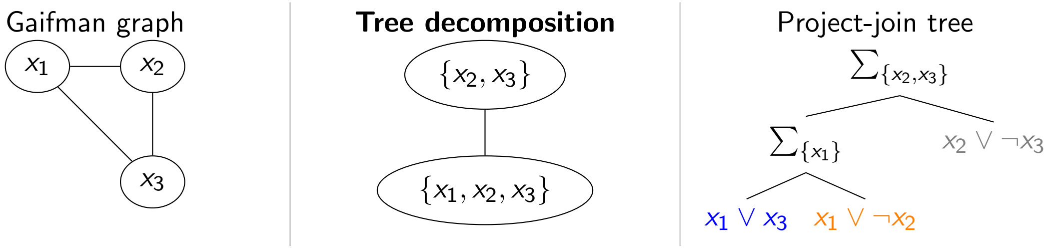


One-shot planner HTB constructs project-join trees with CP heuristics:

- Variable order: *maximal-cardinality search* (Tarjan & Yannakakis, 1984), *lexicographic search for perfect/minimal order* (Koster, Bodlaender, & Van Hoesel, 2001), and *minimal fill-in* (Dechter, 2003)
- Clause order: *bucket elimination* (Dechter, 1999) and *Bouquet's Method* (Bouquet, 1999)

Planner LG with *Anytime* Tree Decompositions

CNF formula: $(x_1 \vee x_3) \wedge (x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3)$



Anytime planner `LG` constructs project-join trees with tree decomposers: `FlowCutter` (Strasser, 2017), `htd` (Abseher, Musliu, & Woltran, 2017), and `Tamaki` (Tamaki, 2019)

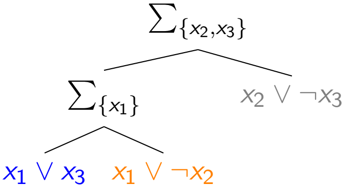
Tree decomposition (Robertson & Seymour, 1991) has also been applied to join-query optimization (McMahan et al., 2004)

Progress

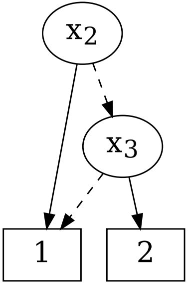
Executor: DMC with *Sparse* Algebraic Decision Diagrams (ADDs)

Bottom-up valuations of nodes of project-join tree are intermediate computations

Project-join tree



Algebraic decision diagram (ADD)



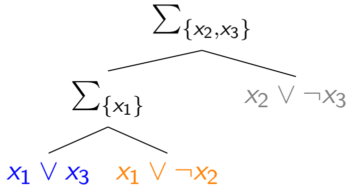
Executor DMC valuates project-join trees using **sparse** ADDs (Bahar et al., 1997)

- ADD package CUDD (Somenzi, 2015)

Executor: `tensor` with *Dense* Tensors

Bottom-up valuations of nodes of project-join tree are intermediate computations

Project-join tree



Tensors (multi-dimensional arrays)

- 0-dimension (scalar): $s \in \mathbb{R}$
- 1-dimension (list): $A[i] \in \mathbb{R}$
- 2-dimension (matrix): $M[i][j] \in \mathbb{R}$
- 3-dimension: $T[i][j][k] \in \mathbb{R}$
- ...

Executor `tensor` valuates project-join trees using **dense** tensors

- Tensor package `NumPy` (Oliphant, 2006)

Progress

1. Boolean Model-Counting Problem (#SAT)
2. Dynamic Programming for Model Counting with Project-Join Trees
3. Planning Phase: Constructing Project-Join Trees
   - `HTB`: *One-Shot* CP Heuristics
   - `LG`: *Anytime* Tree Decompositions
4. Execution Phase: Valuating Project-Join Trees
   - `DMC`: *Sparse* Algebraic Decision Diagrams (ADDs)
   - `tensor`: *Dense* Tensors
5. **Experimental Evaluation**

Benchmarks: 1976 CNF Weighted Model-Counting Instances

Bayesian class: 1080 benchmarks
(Sang, Beame, & Kautz, 2005)

- *Deterministic Quick Medical Reference*
- *Grid Networks*
- *Plan Recognition*

Non-Bayesian class: 896 benchmarks
(Clarke et al., 2001; Klebanov, Manthey, & Muise, 2013; Palacios & Geffner, 2009; Sinz, Kaiser, & Küchlin, 2003)

- *Planning*
- *Bounded Model Checking*
- *Circuit*
- *Configuration*
- *Quantitative Information Flow*
- *Scheduling*
- *Handmade*
- *Random*

Experimental Setup

High-performance computing cluster at Rice University (NOTS):

- Hardware: Xeon E5-2650v2 CPU (2.60-GHz)
- Memory limit: 30 GB of RAM
- Time limit: 1000 seconds

Experiment: Comparing DPMC Planners and Executors

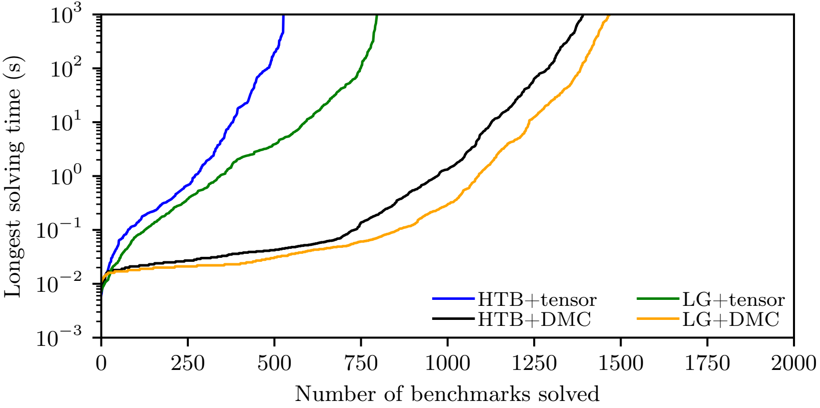


Figure 2: Planning: `LG` (*anytime* tree decompositions) outperforms `HTB` (*one-shot* CP heuristics). Execution: `DMC` (*sparse* ADDs) outperforms `tensor` (*dense* tensors).

Experiment: Comparing Weighted Model Counters on 1976 Benchmarks

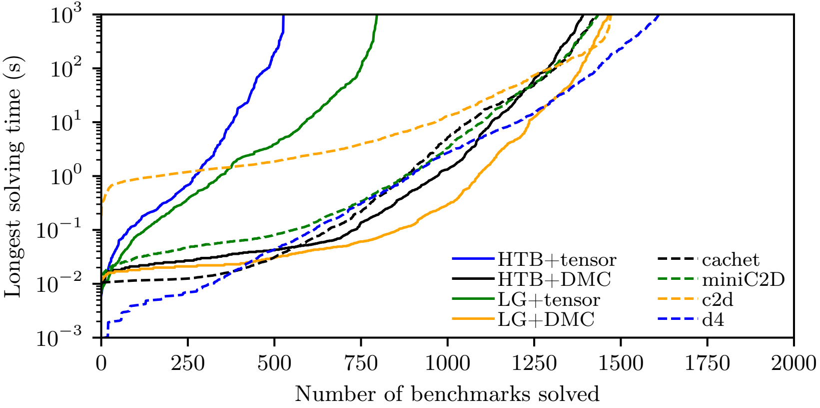


Figure 3: `LG+DMC` is fastest on 471 benchmarks. `DPMC` (all four planner+executor combinations) is fastest on 584 benchmarks.

Experiment: Virtual Best Solvers

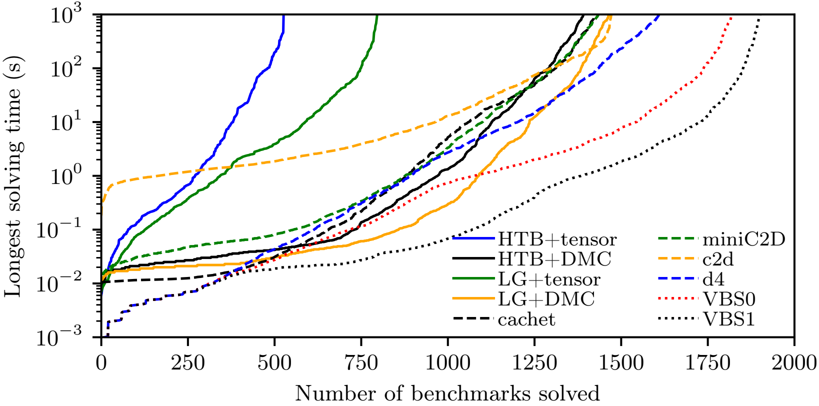


Figure 4: Virtual best solvers (simulating running actual solvers in parallel): `VBS1` (with `DPMC`) is significantly faster than `VBS0` (without `DPMC`).

Model Counting by Dynamic Programming (End of Technical Part)

Summary:

Planner: CNF formula $\mapsto$ *project-join tree*

$\downarrow$ `Model counter` (*planner*+**executor**)

Executor: project-join tree $\mapsto$ **model count**

`HTB` *planner: one-shot CP heuristics*

`LG` *planner: anytime tree decompositions*

`ADDMC`

`Crossover1`

`Crossover2`

`TensorOrder`

`DMC` **executor: sparse ADDs**

`tensor` **executor: dense tensors**

Future work:

- Planning phase: fractional hypertree decomposition (Gottlob et al., 2020)
- Execution phase: database tables, as in model counter `dpdb` (Fichte et al., 2020)

Source code and experimental data: https://github.com/vardigroup/DPMC

References I

Abseher, M., Musliu, N., & Woltran, S. (2017). htd–a free, open-source framework for (customized) tree decompositions and beyond. In *Cpaior*. Springer. https://doi.org/10.1007/978-3-319-59776-8_30

Bahar, R. I., Frohm, E. A., Gaona, C. M., Hachtel, G. D., Macii, E., Pardo, A., & Somenzi, F. (1997). Algebraic decision diagrams and their applications. *Form Method Syst Des*, *10*(2-3), 171–206. https://doi.org/10.1023/A:1008699807402

Bouquet, F. (1999). *Gestion de la dynamicité et énumération d'impliquants premiers: une approche fondée sur les Diagrammes de Décision Binaire* (Doctoral dissertation). Aix-Marseille 1. https://www.theses.fr/1999AIX11011

Clarke, E., Biere, A., Raimi, R., & Zhu, Y. (2001). Bounded model checking using satisfiability solving. *Form Method Syst Des*, *19*(1), 7–34. https://doi.org/10.1023/A:1011276507260

Darwiche, A. (2004). New advances in compiling CNF to decomposable negation normal form. In *Ecai*. https://dl.acm.org/doi/10.5555/3000001.3000069

References II

Dechter, R. (1999). Bucket elimination: a unifying framework for reasoning. *AIJ*, *113*(1-2), 41–85. https://doi.org/10.1016/S0004-3702(99)00059-4

Dechter, R. (2003). *Constraint processing*. Morgan Kaufmann. https://doi.org/10.1016/B978-1-55860-890-0.X5000-2

Dudek, J. M., Dueñas-Osorio, L., & Vardi, M. Y. (2019). Efficient contraction of large tensor networks for weighted model counting through graph decompositions. *arXiv preprint arXiv:1908.04381*. https://arxiv.org/abs/1908.04381

Dudek, J. M., Phan, V. H. N., & Vardi, M. Y. (2020). ADDMC: weighted model counting with algebraic decision diagrams. In *Aaai*. https://doi.org/10.1609/aaai.v34i02.5505

Duenas-Osorio, L., Meel, K. S., Paredes, R., & Vardi, M. Y. (2017). Counting-based reliability estimation for power-transmission grids. In *Aaai*. https://dl.acm.org/doi/10.5555/3298023.3298219

References III

Fichte, J. K., Hecher, M., Thier, P., & Woltran, S. (2020). Exploiting database management systems and treewidth for counting. In *Padl*. Springer. https://doi.org/10.1007/978-3-030-39197-3_10

Gottlob, G., Lanzinger, M., Pichler, R., & Razgon, I. (2020). Complexity analysis of general and fractional hypertree decompositions. *arXiv preprint arXiv:2002.05239*.

Klebanov, V., Manthey, N., & Muise, C. (2013). SAT-based analysis and quantification of information flow in programs. In *Qest*. https://doi.org/10.1007/978-3-642-40196-1_16

Koster, A. M., Bodlaender, H. L., & Van Hoesel, S. P. (2001). Treewidth: computational experiments. *Electron Notes Discrete Math*, *8*, 54–57. https://doi.org/10.1016/S1571-0653(05)80078-2

Lagniez, J.-M., & Marquis, P. (2017). An improved decision-DNNF compiler. In *Ijcai*. https://doi.org/10.24963/ijcai.2017/93

References IV

McMahan, B. J., Pan, G., Porter, P., & Vardi, M. Y. (2004). Projection pushing revisited. In *Edbt*. Springer. https://doi.org/10.1007/978-3-540-24741-8_26

Oliphant, T. E. (2006). *A guide to NumPy* (Vol. 1). Trelgol Publishing USA. https://dl.acm.org/doi/book/10.5555/2886196

Oztok, U., & Darwiche, A. (2015). A top-down compiler for sentential decision diagrams. In *Ijcai*. https://dl.acm.org/doi/10.5555/2832581.2832687

Palacios, H., & Geffner, H. (2009). Compiling uncertainty away in conformant planning problems with bounded width. *JAIR*, *35*, 623–675. https://dl.acm.org/doi/10.5555/1641503.1641518

Robertson, N., & Seymour, P. D. (1991). Graph minors. X. Obstructions to tree-decomposition. *J Combinatorial Theory B*, *52*(2), 153–190. https://doi.org/10.1016/0095-8956(91)90061-N

References V

Sang, T., Bacchus, F., Beame, P., Kautz, H. A., & Pitassi, T. (2004). Combining component caching and clause learning for effective model counting. *SAT*, *4*, 20–28. http://www.satisfiability.org/SAT04/accepted/65.html

Sang, T., Beame, P., & Kautz, H. A. (2005). Performing Bayesian inference by weighted model counting. In *Aaai*. AAAI Press. https://dl.acm.org/doi/10.5555/1619332.1619409

Shwe, M. A., Middleton, B., Heckerman, D. E., Henrion, M., Horvitz, E. J., Lehmann, H. P., & Cooper, G. F. (1991). Probabilistic diagnosis using a reformulation of the INTERNIST-1/QMR knowledge base. *Methods of Information in Medicine*. https://europepmc.org/article/med/1762578

Sinz, C., Kaiser, A., & Küchlin, W. (2003). Formal methods for the validation of automotive product configuration data. *AI EDAM*, *17*(1), 75–97. https://doi.org/10.1017/S0890060403171065

Somenzi, F. (2015). *CUDD: CU decision diagram package–release 3.0.0*. University of Colorado at Boulder. https://github.com/ivmai/cudd

References VI

Strasser, B. (2017). Computing tree decompositions with FlowCutter: PACE 2017 submission. *arXiv preprint arXiv:1709.08949*. https://arxiv.org/abs/1709.08949

Tamaki, H. (2019). Positive-instance-driven dynamic programming for treewidth. *J Comb Optim*, *37*(4), 1283–1311. https://doi.org/10.1007/s10878-018-0353-z

Tarjan, R. E., & Yannakakis, M. (1984). Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SICOMP*, *13*(3), 566–579. https://doi.org/10.1137/0213035

Valiant, L. G. (1979). The complexity of enumeration and reliability problems. *SICOMP*, *8*(3), 410–421. https://doi.org/10.1137/0208032