Proof Systems for Tensor-based Model Counting

**Olaf Beyersdorff, Joachim Giesen, Andreas Goral,
Tim Hoffmann, Kaspar Kasche, Christoph Staudt**

Friedrich Schiller University Jena, Germany
olaf.beyersdorff@uni-jena.de, joachim.giesen@uni-jena.de, andreas.goral@uni-jena.de,
hoffmann.t@uni-jena.de, kaspar.kasche@uni-jena.de, christoph.staudt@uni-jena.de

Abstract

Solving the *model counting problem* #SAT, asking for the number of satisfying assignments of a propositional formula, has been explored intensively and has gathered its own community. While most existing solvers are based on knowledge compilation, another promising approach is through contraction in tensor hypernetworks.

We perform a theoretical *proof-complexity analysis* of this approach. For this, we design two new tensor-based proof systems that we show to tightly correspond to tensor-based #SAT solving. We determine the simulation order of #SAT proof systems and prove exponential separations between the systems. This sheds light on the relative performance of different #SAT solving approaches.

1 Introduction

The *model counting problem* #SAT asks to compute the number of satisfying assignments for a propositional formula. A large variety of real-world questions can be encoded very naturally and succinctly in #SAT, including various problems in probabilistic reasoning (Bacchus, Dalmao, and Pitassi 2003; Latour et al. 2017), risk analysis (Zhai et al. 2020; Dueñas-Osorio et al. 2017) and explainable artificial intelligence (Shi et al. 2020; Baluta et al. 2021).

#SAT is a generalisation of the well-studied SAT problem asking if a given Boolean formula is satisfiable. In fact, #SAT is #P-complete (Valiant 1979) and appears to be a much harder problem than the NP-complete SAT problem as by the theorem of (Toda 1991) any problem in the polynomial hierarchy can be solved in polynomial time with access to a #SAT oracle.

Despite its high complexity, there exist various competitive algorithmic approaches for #SAT, and research in #SAT solving has increased significantly over the past two decades as witnessed by the annual model counting competition (Fichte, Hecher, and Hamiti 2021). In contrast to SAT solving where conflict-driven clause learning (CDCL) (Marques Silva, Lynce, and Malik 2021) dominates the scene, state-of-the-art #SAT solvers apply conceptually very different approaches, including the lifting of standard techniques from SAT solving (Thurley 2006), employing knowledge compilation (Lagniez and Marquis 2017), and dynamic programming (Fichte et al. 2020). A fourth promising approach uses tensor network contractions (Kourtis et al. 2019), which is the topic of this paper.

Tensor (hyper-)networks are a versatile computational framework that, among many other things, has been used extensively in signal processing, inference in graphical models, and in decoding for state-of-the-art codes (Aji and McEliece 2000). They are also a staple of computational physics, where they are used for simulating quantum systems (Orús 2019), including the simulation of quantum circuits (Biamonte and Bergholm 2017). The physics community was first to observe that tensor networks can also be used for solving the SAT and #SAT problems (García-Sáez and Latorre 2012; Biamonte, Morton, and Turner 2015).

More recently, the core model counting community started to explore the competitive potential of tensor networks for model counting (Kourtis et al. 2019; Dudek, Dueñas-Osorio, and Vardi 2019; Dudek, Phan, and Vardi 2020b; de Beaudrap, Kissinger, and Meichanetzidis 2020; Goral et al. 2024; Dudek, Shrotri, and Vardi 2022; Dudek 2021). Experimental results (Dudek, Dueñas-Osorio, and Vardi 2019; Kourtis et al. 2019; Dudek, Shrotri, and Vardi 2022) suggest that tensor networks can be competitive with state-of-the-art model counters based on knowledge compilation (Darwiche and Marquis 2002; Korhonen and Järvisalo 2023). On benchmark data sets, implementations of the tensor network approach contribute to the virtual best solver, that is, there are problem instances, where these implementations are faster than competing approaches. Here, our aim is to perform a theoretical analysis of tensor-based model counting that contributes to a better understanding of its relative performance.

The main theoretical approach for gauging the strength of SAT solvers is through *proof complexity* (Buss and Nordström 2021), whereby solvers are modelled by rigorously defined proof systems and in particular, solver traces can be efficiently mapped to proofs. The most important correspondence of this kind is between the modern CDCL paradigm for SAT solving and the propositional resolution proof system (Beame, Kautz, and Sabharwal 2004; Pipatsrisawat and Darwiche 2011; Atserias, Fichte, and Thurley 2011). This connection between solvers and proof systems manifests

in three facets. First, during the past decade, proof systems have become the standard tool to certify the answers of solvers through *proof logging*. Modern SAT solvers use proof systems optimised for this purpose (Heule, Jr., and Wetzler 2013; Wetzler, Heule, and Jr. 2014). Second, proof systems are the right tool to show lower bounds for the running time of solvers. In the propositional case, the plethora of proof complexity results for resolution can be applied to CDCL. In particular, lower bounds for resolution proof size directly translate into lower bounds for CDCL runtime. Third, these insights can be used to theoretically compare the strength of different solvers. In particular, if we have two proof systems P and Q that tightly correspond to two algorithmic approaches while P is a stronger proof system than Q, then we can conclude that the solver corresponding to P is theoretically superior to the other one.

On the *correspondence between #SAT solvers and proof systems* far less is known. A number of conceptually different proof systems for #SAT have been introduced recently. There are static systems that exploit the fact that Decision-DNNFs – a very common circuit class in knowledge compilation (Darwiche 1999) – can be efficiently extracted from most model counting solvers (Capelli 2017). These systems based on Decision-DNNFs are kcps (Capelli 2019) and its improved version (Capelli, Lagniez, and Marquis 2021). A similar more powerful static proof system is CPOG for certified proof checking of #SAT solvers (Bryant et al. 2023), similar in spirit to the general proof-checking formats RAT and DRAT (Heule, Jr., and Wetzler 2013; Wetzler, Heule, and Jr. 2014) used for SAT solving. Further, there is the more dynamic line-based proof system MICE (Fichte, Hecher, and Roland 2022; Beyersdorff, Hoffmann, and Spachmann 2023). The relative strength of these three proof systems was recently determined in (Beyersdorff et al. 2024). A fourth very strong #SAT proof system CLIP was very recently introduced in (Chede, Chew, and Shukla 2024). For MICE, some additional proof complexity results are available (Beyersdorff, Hoffmann, and Spachmann 2023, 2024), but in general proof complexity for #SAT is in its infancy.

The strongest system CPOG allows proof logging for a number of state-of-the-art solvers in model counting, including sharpSAT (Thurley 2006), DPDB (Fichte et al. 2020) and D4 (Lagniez and Marquis 2017). Hence CPOG proofs can verify the correctness of answers of these #SAT solvers.

For tensor-based model counting, neither proof systems nor theoretical results on the runtime of these solvers exist.

1.1 Our contributions

We perform a proof-complexity analysis for tensor-based proof systems and #SAT solvers. We summarise our results and explain their practical implications.

(a) Modelling tensor-based #SAT solvers by proofs. We introduce a new proof system specifically tailored towards modelling tensor-based #SAT solvers. The proof system comes in two flavours T_{dense} and T_{sparse}, depending on how tensors are represented in solvers and proofs: either as matrices for which we use the proof system T_{dense}, or as lists which corresponds to the system T_{sparse}.

With a view towards proof complexity analysis, it is paramount to keep the systems simple, and T_{dense} and T_{sparse} meet this criterion as they just use three intuitive rules, namely, axioms to introduce tensors from clauses and two rules to combine and aggregate them (Figure 3).

We show that the proof systems closely correspond to tensor-based #SAT solving (Algorithm 1) in the sense that T_{dense} and T_{sparse} proofs can be efficiently extracted from runs of #SAT solvers following the paradigm of Algorithm 1 (Theorem 3.3). The correspondence between the running time of Algorithm 1 and minimal proof size in T_{dense} is tight.

This implies that lower bounds for T_{dense} and T_{sparse} proof size directly translate into lower bounds for the running time of tensor-based #SAT solvers. Further, T_{dense} and T_{sparse} could be used for proof logging of these solvers.

(b) Simulations between #SAT proof systems. Our proof-complexity analysis of T_{dense} and T_{sparse} starts by comparing the new systems with the existing proof system MICE in terms of proof size. For this we use the notion of p-simulation between two proof systems P and Q, where P p-simulates Q if Q-proofs can be efficiently transformed into P-proofs.

While it is straightforward to observe that T_{sparse} p-simulates T_{dense}, it is technically involved to prove that MICE p-simulates T_{sparse} (Theorem 4.2). The resulting simulation order of #SAT proof systems is depicted in Figure 1.

From a practical perspective this has two implications. First, MICE can also be used for tensor-based proof logging (Corollary 4.4). Second, existing proof size lower bounds for MICE of (Beyersdorff, Hoffmann, and Spachmann 2023) yield hardness results for tensor-based #SAT solving (Corollary 4.3).

(c) Separations between #SAT proof systems. In our third contribution we determine the precise relative strength of the three #SAT proof systems T_{dense}, T_{sparse}, and MICE and show that they form a strict chain as depicted in Figure 1. This entails proving exponential separations between the systems, i.e. we exhibit specific formulas with short proofs in T_{sparse}, but requiring exponential-size proofs in T_{dense} (Theorem 6.2), and analogously for MICE vs T_{sparse} (Theorems 6.5 and 6.6).

Technically, we achieve the first separation by tightly characterising proof size for T_{dense} by the treewidth of the primal graph of the formula (Theorem 5.1). Connections between the runtime of Algorithm 1 and treewidth were known before (Dudek, Dueñas-Osorio, and Vardi 2019; Dudek 2021) and transfer to proof size. We exhibit both separations on satisfiable as well as on unsatisfiable formulas. Intuitively, the second separation is harder to show on CNFs with few models (or in the extreme case none), because the sparse representation is superior on tensors with few non-zero entries. Indeed, our separation between MICE and T_{sparse} is technically more involved. We achieve this on the pebbling formulas, well known in proof complexity (Theorem 6.6).

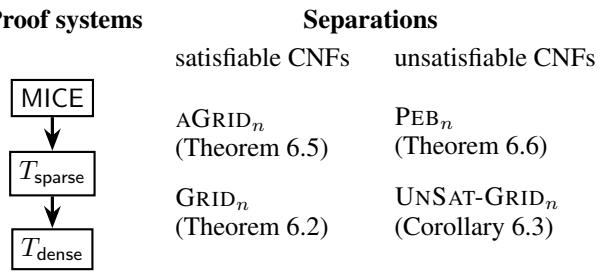

Figure 1: The simulation order of #SAT proof systems. An arrow from proof system P to Q means that P p-simulates Q and that P is exponentially separated from Q.

Practically, this means that algorithms using sparse tensor encodings can exponentially outperform algorithms using dense encodings, which is also confirmed by recent experiments (Staudt et al. 2025) (cf. also the discussion in Section 7).

We stress that while our results bear practical relevance, this paper performs a purely theoretical proof-complexity analysis of tensor-based model counting. We believe that this will lay the ground for practical follow-up research, particularly on sparse representations.

1.2 Organisation

Section 2 reviews concepts from #SAT, proof complexity, and tensors. In Section 3 we formalise tensor-based #SAT solving and model it with the new tensor-based proof systems T_{dense} and T_{sparse}. Sections 4 to 6 contain our proof-complexity analysis and show the simulations, proof size characterisation for T_{dense}, and separations, respectively. We conclude in Section 7 with a discussion.

Due to space constraints, proofs are omitted or sketched.

2 Preliminaries

2.1 Propositional satisfiability and model counting

A literal l is a variable z or its negation $\overline{z}$. A clause is a disjunction of literals, a conjunctive normal form (CNF) is a conjunction of clauses. Often, we write clauses as sets of literals and formulas as sets of clauses. We assume that propositional formulas are given in CNF.

For a formula F, $\mathsf{vars}(F)$ denotes the set of all variables that occur in F, and $\mathsf{lits}(F)$ is the set of all literals of F. If $C \in F$ is a clause and $V \subseteq \mathsf{vars}(F)$ is a set of variables, we define $C|_V = \{l \in C \mid \mathsf{vars}(l) \in V\}$. $F|_V$ denotes the formula F with every clause C replaced by $C|_V$. An assignment is a function α mapping variables to Boolean values. For a (partial) assignment α, $F[\alpha]$ denotes the formula where we replace all occurrences of variables x with $\alpha(x)$. If $F[\alpha] = 1$, we say α satisfies F and write $\alpha \models F$. Occasionally, we interpret an assignment as a CNF consisting of precisely those unit clauses that specify the assignment. Therefore, the set operations are well-defined for formulas and assignments. We say that two assignments are consistent if they agree on their intersection. For some set of variables V, let $\langle V \rangle$ denote the set of all $2^{|V|}$ assignments to V.

For a formula φ, $\mathsf{Mod}(\varphi) := \{\alpha \in \langle \mathsf{vars}(\varphi) \rangle \mid \alpha \models \varphi\}$ is the set of all models of φ. The *model counting problem* #SAT asks to compute $|\mathsf{Mod}(\varphi)|$ for a given formula φ.

2.2 Proof systems for model counting

In the framework of Cook and Reckhow (Cook and Reckhow 1979), a *proof system* for model counting fulfils three properties: *soundness* (proofs are correct), *completeness* (for each formula φ, there is a proof that φ has indeed $|\mathsf{Mod}(\varphi)|$ models), and proofs must be easy to check (in polynomial time). We only consider *line-based proof systems* where proofs consist of lines (comprised of formulas or tensors in our systems) that are derived by a set of rules. A proof is *treelike*, if any proof line is used at most once to derive a new one.

We compare proof systems by simulations. Let P and Q be #SAT proof systems. Then P *p-simulates* Q if every Q-proof can be translated in polynomial time into a P-proof of the same formula. Two proof systems are p-equivalent if they p-simulate each other. Further, P is *exponentially separated* from Q if there is a family of formulas that have polynomial sized P-proofs, but all Q-proofs require exponential size.

One of the first #SAT proof systems is MICE (Fichte, Hecher, and Roland 2022). We use a simplified version (Beyersdorff, Hoffmann, and Spachmann 2023), which is p-equivalent to the original MICE system.

MICE is a line-based proof system with *claims* as lines. A *claim* is a 3-tuple (F, A, c) consisting of a CNF F, a partial assignment A of $\mathsf{vars}(F)$ (called *assumption*) and a count c. A claim is *correct* if F has exactly c models under the assignment A. A MICE *proof* π for a formula φ is a sequence of claims $I_1, \ldots, I_k$ that are derived with the inference rules in Figure 2 such that the final claim has the form $(\varphi, \emptyset, c)$ for some count c. Then, π proves that φ has exactly c models. As the count c in a correct claim (F, A, c) is determined by F and A, we sometimes omit it and use the notation (F, A).

2.3 Tensor hypernetworks

Because of their expressiveness and conceptual simplicity, tensor networks and tensor hypernetworks are used in many different areas. So far, however, there is no established standard notation for tensor hypernetworks. Here, we give a brief formal definition of tensor hypernetworks that is well suited for the model counting application.

Intuitively, a tensor is an n-dimensional array. We formally define it as a discrete function $T : K \to \mathbb{R}$. Here, K is called the set of *positions* and is the Cartesian product of n sets called *axes*. For example, the axis x might be a set with the two elements $x = 0$ and $x = 1$, the axis y might also consist of the elements $y = 0$ and $y = 1$, and if these are the only two axes then $(x = 0, y = 1)$ is one of the positions in K.

Let $\mathcal{A} = \{A_1, \ldots, A_n\}$ be a set of axes and $K_{\mathcal{A}} = \bigtimes_{i=1}^{n} A_i$ the corresponding set of positions. A *tensor hypernetwork* N is a set of tensors with axes from $\mathcal{A}$. We view it

Axiom. $\dfrac{}{(\emptyset, \emptyset, 1)}$ (Ax)

Composition. $\dfrac{(F, A_1, c_1) \quad \cdots \quad (F, A_n, c_n)}{(F, A, \sum_{i\in[n]} c_i)}$ (Comp)

C-1 $\mathsf{vars}(A_1) = \cdots = \mathsf{vars}(A_n)$ and $A_i \neq A_j$ for $i \neq j$,

C-2 $A \subseteq A_i$ for all $i \in [n]$,

C-3 there exists a resolution refutation of $A \cup F \cup \{\overline{A}_i \mid i \in [n]\}$. Such a refutation is included into the trace and is called an *absence of models statement*.

Join. $\dfrac{(F_1, A_1, c_1) \quad (F_2, A_2, c_2)}{(F_1 \cup F_2, A_1 \cup A_2, c_1 \cdot c_2)}$ (Join)

J-1 A_1 and A_2 are consistent,

J-2 $\mathsf{vars}(F_1) \cap \mathsf{vars}(F_2) \subseteq \mathsf{vars}(A_i)$ for $i \in \{1, 2\}$.

Extension. $\dfrac{(F_1, A_1, c_1)}{(F, A, c_1 \cdot 2^{|\mathsf{vars}(F)\setminus(\mathsf{vars}(F_1)\cup\mathsf{vars}(A))|})}$ (Ext)

E-1 $F_1 \subseteq F$,

E-2 $A|_{\mathsf{vars}(F_1)} = A_1$,

E-3 A satisfies $F \setminus F_1$.

Figure 2: Inference rules for MICE (Beyersdorff, Hoffmann, and Spachmann 2023).

as a hypergraph; for each axis A it has a hyperedge that contains all tensors with this axis. The *full combination* of N is a tensor T_N with positions $K_{\mathcal{A}}$, and every position $k \in K_{\mathcal{A}}$, $T_N(k) = \prod_{T\in N} T(k)$. In the expression $T(k)$, we ignore all irrelevant axes of k. The *full contraction* of N is the sum of all entries of T_N and can be written as

$$\sum_{k\in K_{\mathcal{A}}} \prod_{T\in N} T(k).$$

The challenge of the tensor contraction problem is to find algorithms that efficiently compute the full contraction.

The model counting problem on a CNF φ can be reduced to computing the full contraction of a tensor hypernetwork. For each variable $x \in \mathsf{vars}(\varphi)$, we have an axis with two elements $x = 0$ and $x = 1$. This way, we can identify the positions over these axes with propositional assignments to the corresponding variables.

For each clause $C \in \varphi$, we introduce a tensor T_C with axes corresponding to the variables in C, and $T_C(\alpha) = \mathbf{1}\big[\alpha \models C\big]$ where $\mathbf{1}[\text{condition}]$ is an indicator function that takes the value 1 if the condition is satisfied, and the value 0 otherwise. The full combination T_φ of these tensors corresponds to the CNF, with $T_\varphi(\alpha) = \mathbf{1}\big[\alpha \models \varphi\big]$. Therefore, the full contraction is equal to the propositional model count. We also see this by writing the model count of φ as

$$\sum_{\alpha\in\langle\mathsf{vars}(\varphi)\rangle} \prod_{C\in\varphi} \mathbf{1}\big[\alpha \models C\big]$$

which has exactly the form of a full contraction of a tensor hypernetwork.

3 Tensor-Based Model Counting

We are finally prepared to introduce two new proof systems for model counting: T_{dense} and T_{sparse}. The proof systems are inspired by a tensor-based algorithm for algebraic model counting, that, essentially, is the vanilla algorithm for computing tensor contractions (Dudek, Dueñas-Osorio, and Vardi 2019; Dudek, Phan, and Vardi 2020b). We first present the contraction algorithm before introducing the two new proof systems.

3.1 Model counting by tensor contraction

An algorithm for computing the model count for a CNF $\varphi = \{C_1, \ldots, C_m\}$ with $\mathsf{vars}(\varphi) = \{x_1, \ldots, x_n\}$ is suggested from rewriting its algebraic formulation,

$$\sum_{\alpha\in\langle\mathsf{vars}(\varphi)\rangle} \prod_{j=1}^{m} \mathbf{1}\big[\alpha \models C_j\big] \quad = \quad \sum_{\alpha_1\in\langle\{x_1\}\rangle} \cdots \sum_{\alpha_n\in\langle\{x_n\}\rangle} \prod_{j=1}^{m} \mathbf{1}\big[\bigcup_{i\in[n]} \alpha_i \models C_j\big] \qquad (1)$$

namely, contract the tensors $\mathbf{1}\big[\alpha \models C_j\big]$ over the variables in φ one after the other. Here, $[n]$ denotes $\{1, \ldots, n\}$. The algorithm is parameterised by a permutation π of the variables in $\mathsf{vars}(\varphi)$, which, in the context of tensor hypernetworks, is called a *contraction order*. With the contraction order, we change the order of summation in Equation (1). This ultimately allows us to factor out some parts of the product and save computational effort. To keep the following exposition simple, we set $T_j(\alpha) := \mathbf{1}\big[\alpha \models C_j\big]$. Further, we write $\mathsf{vars}(T_j)$ for the set of variables that represent the axes of T_j and $\mathsf{tensors}_N(x_i)$ for the set of tensors in N that have an axis for x_i, omitting N if it is clear from context.

Pseudocode for counting the models of a given CNF φ via tensor network contraction is given in Algorithm 1.

Algorithm 1: Model counting by tensor contraction

Input: tensors $T_1, \ldots, T_m$ corresponding to the clauses of φ in variables $x_1, \ldots, x_n$, contraction order π
Output: model count for formula φ

$N = \{T_1, \ldots, T_m\}$ {remaining tensors}
for $i = 1, \ldots, n$ **do**
  $x = \pi(x_i)$ {variable to be aggregated}
  $e_x = \mathsf{tensors}_N(x)$ {hyperedge to be contracted}
  $V = \bigcup_{T\in e_x} \mathsf{vars}(T)$ {variables for this step}
  for α assignment to variables in V **do**
    $P(\alpha) = \prod_{T\in e_x} T(\alpha)$ {*Combination*}
  end for
  for α assignment to variables in $V \setminus \{x\}$ **do**
    $T_{m+i}(\alpha) = P(\alpha \cup \{x = 0\}) + P(\alpha \cup \{x = 1\})$ {*Aggregation*}
  end for
  $N = (N \setminus e_x) \cup \{T_{m+i}\}$ {update the set of} {remaining tensors}
end for
return T_{m+n}

The set of hyperedges in the tensor hypernetwork is

$$E = \big\{\mathsf{tensors}(x) \mid x \in \mathsf{vars}(\varphi)\big\}.$$

Contracting over a variable $x \in \mathsf{vars}(\varphi)$ means contracting the hyperedge $e_x = \mathsf{tensors}(x) \in E$, that is, removing all tensors in e_x from the hypergraph and adding the contracted tensor

$$T_{m+i}(\alpha) = \sum_{\alpha_x \in \langle\{x\}\rangle} \prod_{T \in e_x} T(\alpha \cup \alpha_x)$$

for $\alpha \in \left\langle \bigcup_{T \in e_x} \mathsf{vars}(T) \setminus x \right\rangle$ during the i-th contraction, where $i \in [n]$. The edge set E is updated accordingly, that is, e_x is removed from E and in every remaining edge, all tensors in e_x are replaced by the new tensor T_{m+i}. Algorithm 1 contracts the hyperedges e_x in the given contraction order π. Therefore, the efficiency of the contraction algorithm also depends on π. The last tensor T_{m+n} is a scalar, namely, the model count for φ.

So far, we have not discussed the *representation* of the tensors T_i. The *standard (dense) representation* makes use of a multi-dimensional array, so every tensor T_i has size $2^{|\mathsf{vars}(T_i)|}$. If, however, the tensors are *sparse*, i.e. have few non-zero entries, more succinct representations are possible, e.g. as a list of all non-zero entries. With minor changes, the runtime of Algorithm 1 is quadratic in the encoding size of the tensors. We refer to this version for sparse tensors as Algorithm 1 in the sparse encoding.

Algorithm 1 is folklore. The correctness follows from Proposition 3.2 and Theorem 3.3. The algorithm is generic in the sense that for a specific implementation a heuristic for computing π needs to be supplied. Finding the best contraction order for this algorithm is an NP-hard problem (Chi-Chung, Sadayappan, and Wenger 1997).

3.2 Tensor-based proof systems

We introduce two new proof systems for model counting that correspond to Algorithm 1 and its version in the sparse encoding. Both tensor proof systems use tensors as proof lines, denoted T_V^F, where F is a formula and $V \subseteq \mathsf{vars}(F)$. Every such tensor has an axis for every variable in V and hence every position represents an assignment $\alpha \in \langle V \rangle$. In the proof systems we derive tensors by rules such that derived tensors T_V^F satisfy the invariant

$$T_V^F(\alpha) = |\{\beta \in \langle \mathsf{vars}(F) \setminus V \rangle \mid \alpha \cup \beta \models F\}| \qquad (2)$$

for every assignment $\alpha \in \langle V \rangle$, i.e. $T_V^F(\alpha) = |\mathsf{Mod}(F[\alpha])|$. We say that a tensor T_V^F satisfying (2) is *correct*. For a CNF φ, our goal is to derive the correct scalar tensor

$$T_\emptyset^\varphi(\emptyset) = |\{\beta \in \langle \mathsf{vars}(\varphi) \rangle \mid \beta \models \varphi\}|$$

which is exactly the model count of φ.

We derive new tensors by the three rules depicted in Figure 3. With (T-Ax), we transform a clause of the original formula φ into a tensor. With (Comb), we combine two tensors and with (Agg), we remove an axis from an already derived tensor. Intuitively, rules (Comb) and (Agg) correspond to lines 7 and 10 in Algorithm 1.

Tensor axiom. $\dfrac{}{T^{\{C\}}_{\mathsf{vars}(C)}}$ (T-Ax)

- C is a clause from the input formula φ.
- For every $\alpha \in \langle \mathsf{vars}(C) \rangle$ holds

$$T^{\{C\}}_{\mathsf{vars}(C)}(\alpha) = \mathbf{1}\big[\alpha \models C\big].$$

Combination. $\dfrac{T^{F_1}_{V_1} \quad T^{F_2}_{V_2}}{T^{F_1 \cup F_2}_{V_1 \cup V_2}}$ (Comb)

- $\mathsf{vars}(F_1) \cap \mathsf{vars}(F_2) \subseteq V_1 \cap V_2$.
- For every $\alpha \in \langle V_1 \cup V_2 \rangle$ we have

$$T^{F_1 \cup F_2}_{V_1 \cup V_2}(\alpha) = T^{F_1}_{V_1}(\alpha_{|V_1}) \cdot T^{F_2}_{V_2}(\alpha_{|V_2}).$$

Aggregation. $\dfrac{T^F_V}{T^F_{V \setminus \{x\}}}$ (Agg)

- $x \in V$.
- For every $\alpha \in \langle V \setminus \{x\} \rangle$ we have

$$T^F_{V \setminus \{x\}}(\alpha) = T^F_V(\alpha \cup \{x = 0\}) + T^F_V(\alpha \cup \{x = 1\}).$$

Figure 3: Inference rules for T_{dense} and T_{sparse}.

Choosing different encodings – the dense matrix encoding vs the sparse list encoding – for the tensors and hence the proof lines leads to the two proof systems T_{dense} and T_{sparse}.

Definition 3.1. *A T_{dense} or T_{sparse} proof π for a formula φ is a sequence of tensors $\pi = T^{F_1}_{V_1}, \ldots, T^{F_k}_{V_k}$ that are derived with the inference rules from Figure 3 such that $T^{F_k}_{V_k} = T^\varphi_\emptyset$.*

The size of π is defined as $|\pi| = \sum_i |T^{F_i}_{V_i}|$ where the size of a tensor depends on the encoding and is different in T_{dense} and T_{sparse}. In T_{dense}, we use the dense matrix encoding, where $T^{F_i}_{V_i}$ has size $1 + 2^{|V_i|}$. In T_{sparse}, we encode the tensor by listing its non-zero entries, so the size of $T^{F_i}_{V_i}$ is equal to 1 plus the number of non-zero entries in the tensor.

We show that these are indeed proof systems.

Proposition 3.2. *T_{dense} and T_{sparse} are sound and complete proof systems for #SAT.*

As we always derive correct tensors with the inference rules, we only consider correct tensors from now on. Therefore, the notation T^F_V is well-defined.

3.3 Extracting proofs from algorithm runs

We show a tight relationship between Algorithm 1 and the inference rules of the tensor proof systems that makes it straightforward to extract a proof from a trace of the algorithm. We formalise this in the following theorem.

Theorem 3.3. *If a solver S applies Algorithm 1 to compute the model count of a formula φ in time t, then we can extract a T_{dense} proof for φ from the run of $S(\varphi)$ in time $O(t)$. Further, if there is a T_{dense} proof of φ of size s, there exists a*

permutation π such that Algorithm 1 runs in time $O(s \cdot l^2)$ where l is the sum of the clause lengths of φ.

If the solver uses a sparse representation of tensors we can extract a T_{sparse} proof in time $O(t)$.

Proof sketch. We only sketch the proof of the first statement for T_{dense}. Let φ be the input CNF with clauses $C_1, \ldots, C_m$. Let $T_1, \ldots, T_m$ be the corresponding tensors and π be a permutation of $[n]$ where $n = |\mathsf{vars}(\varphi)|$.

We show that for every $i \in [m+n]$, we can translate tensor T_i from Algorithm 1 into a T_{dense} tensor and derive it in T_{dense} efficiently. For this we represent tensors T_i in the algorithm as tensor proof lines $T_{V_i}^{F_i}$ and define V_i as the set of variables from the axes of T_i. The formula F_i contains the clause C_j if tensor T_j was used to compute T_i. We argue that tensors T_i and $T_{V_i}^{F_i}$ are equivalent as functions.

Next, we show by induction that we can derive all these tensors in T_{dense} efficiently. In the base case $i \in [m]$, we obtain T_i by applying (T-Ax) on clause C_i. In the inductive step, let $k > m$ and $T_{V_1}^{F_1}, \ldots, T_{V_{k-1}}^{F_{k-1}}$ have been derived. We inspect the outer for-loop of Algorithm 1 with $i = k - m$. We derive the line corresponding to T by applying (Comb) pairwise to all tensors $T_{V_j}^{F_j}, j \in e_x$. All these tensors are available by inductive hypothesis since all elements of e_x have index less than k. Then we remove variable x by (Agg) and obtain $T_{V_k}^{F_k}$. Since the algorithm performs a tensor contraction, $T_{V_{m+n}}^{F_{m+n}} = T_{\emptyset}^{\varphi}$ and we get a valid T_{dense} proof. □

In fact, the extracted proofs are always treelike. Moreover, using different contraction orders π will result in different proofs. Good contraction orders correspond to short proofs, and lower bounds on proof size provide lower bounds for the algorithm, independent of the contraction order.

4 Simulations Between #SAT Proof Systems

We now analyse how the new systems T_{dense} and T_{sparse} relate to each other and to the existing proof system MICE. Our aim is to show the simulations in Figure 1. As any T_{dense} proof can be rewritten as a T_{sparse} proof without size increase we obtain:

Observation 4.1. *T_{sparse} p-simulates T_{dense}.*

We obtain a similar simulation between MICE and T_{sparse}:

Theorem 4.2. *MICE p-simulates T_{sparse}.*

Proof sketch. We only sketch how to efficiently transform proofs from the weaker system T_{dense} into MICE proofs.

For this, let $T_{V_1}^{F_1}, \ldots, T_{V_n}^{F_n}$ be a T_{dense} proof of a CNF φ. We capture every position of any tensor $T_{V_i}^{F_i}$, $i \in [n]$ as a claim by defining $I_i := \{(F_i, \alpha) \mid \alpha \in \langle V_i \rangle\}$. Let $I_0 = \{(\emptyset, \emptyset)\}$ and I_i' be a list of all entries from I_i in arbitrary order. By case distinction on the rule that derived $T_{V_i}^{F_i}$ we can show that $I_0', I_1', \ldots, I_n'$ is a valid MICE proof of φ. Roughly, applications of (Comb) in T_{dense} are simulated by (Join) in MICE, and (Agg) by (Comp). The most delicate steps are the (Comp) steps which need a resolution proof for the absence of models statement. This is easy to produce here as $\{\overline{\alpha} \mid \alpha \in \langle V_i \rangle\}$ yields a full set of $2^{|V_i|}$ contradictory clauses. □

Theorem 4.2 shows that lower bounds for the size of MICE proofs imply lower bounds for T_{dense} and T_{sparse}. Recently (Beyersdorff, Hoffmann, and Spachmann 2023), XOR-Pairs$_n$ formulas were introduced and it was shown that MICE proofs of these CNFs require $2^{\Omega(n)}$ steps. Consequently we obtain:

Corollary 4.3. *Proofs of XOR-Pairs$_n$ in T_{dense} or T_{sparse} have size $2^{\Omega(n)}$.*

Further, the proof of Theorem 4.2 shows how to convert a T_{sparse} proof into a MICE proof efficiently, i.e. we can use MICE for proof logging for solvers applying tensor contractions:

Corollary 4.4. *If a solver S applies Algorithm 1 to compute the model count of a formula φ in time t, then we can extract a MICE proof for φ in time $O(t)$ from the run of Algorithm 1, and respectively in time $t^{O(1)}$ in its sparse version.*

5 Characterising Proof Size in T_{dense}

It is known from the literature that there is a tight connection between the runtime of Algorithm 1 under the best contraction order and the treewidth of the primal graph of the formula (Markov and Shi 2008; Dudek, Dueñas-Osorio, and Vardi 2019; Dudek 2021).

We use this connection to characterise minimal proof size in T_{dense}. For an introduction to the *treewidth* of a graph we refer to (Robertson and Seymour 1986). We represent a formula φ by its *primal graph*, which has nodes for all variables in φ. Two nodes x, y share an edge if φ has a clause containing x and y (irrespective of the polarities). We can now characterise T_{dense}:

Theorem 5.1. *Let k be the treewidth of the primal graph of a formula φ with c clauses. Then, the size of the smallest T_{dense} proof of φ is in the interval $[2^{k+1}, c \cdot 2^{k+3} + 2 \cdot c]$.*

In particular, Theorem 5.1 yields an easy technique to obtain lower bounds for T_{dense} that we will use in Section 6.

6 Separations Between #SAT Proof Systems

In Section 4 we saw that MICE is at least as strong as T_{sparse} and T_{sparse} is at least as strong as T_{dense}. Formally, MICE p-simulates T_{sparse} which p-simulates T_{dense}. In this section we strengthen these results to exponential separations, implying that the converse simulations do not hold (cf. Figure 1). We show this by exhibiting CNFs requiring exponentially larger proofs in T_{dense} than in T_{sparse} (resp. in T_{sparse} than in MICE).

6.1 Separating T_{sparse} from T_{dense}

In light of Theorem 5.1, to separate T_{sparse} and T_{dense} we need formulas with high treewidth that are easy for T_{sparse}. To get such CNFs, we design the GRID$_n$ formulas with variables for each node in an $n \times n$ grid and clauses ensuring that variables for adjacent nodes take the same value.

Definition 6.1. *The formula GRID$_n$ contains variables $x_{i,j}$ for all $i, j \in [n]$ and clauses encoding*

$x_{i,j} = x_{i+1,j}$ *for* $i \in [n-1]$, $j \in [n]$,
$x_{i,j} = x_{i,j+1}$ *for* $i \in [n]$, $j \in [n-1]$.

To satisfy GRID_n all variables have to take the same value. Therefore, GRID_n has exactly two models. We show that these formulas exponentially separate T_{sparse} from T_{dense}:

Theorem 6.2. *The* GRID_n *formulas have* T_{sparse} *proofs of size* $O(n^2)$*, but require* T_{dense} *proofs of size at least* 2^{n+1}.

Proof sketch. The lower bound for T_{dense} follows from Theorem 5.1 and the fact that an $n \times n$ grid has treewidth n. It is easy to construct T_{sparse} proofs of size $O(n^2)$. □

Note that we can also get this exponential separation of T_{sparse} and T_{dense} on unsatisfiable formulas. For that, we choose an adapted version of GRID_n where we add two unit clauses $(x_{1,1})$ and $(\overline{x}_{n,n})$. We call this formula UNSAT-GRID_n as it has no models. It is easy to see that the proof of Theorem 6.2 also works for UNSAT-GRID_n.

Corollary 6.3. *The* UNSAT-GRID_n *formulas have* T_{sparse} *proofs of size* $O(n^2)$*, but require* T_{dense} *proofs of size at least* 2^{n+1}.

6.2 Separating MICE from T_{sparse}

Next, we investigate the relation between MICE and T_{sparse}. As wide clauses (in n literals) give rise to large tensor representations (implying T_{sparse} proofs of size at least 2^n), we could easily obtain such a separation on CNFs with wide clauses. It therefore appears more interesting to focus our analysis on 3-CNFs, which are also preferable practically. Indeed we can obtain such a stronger separation using the following 3-CNFs.

Definition 6.4. *Let* AGRID_n *be the formula* GRID_n *where we introduce a fresh variable* a *and add it to every clause, i.e.* $\text{AGRID}_n = \{(C \vee a) \mid C \in \text{GRID}_n\}$.

We can satisfy AGRID_n by setting a to true and then assign all n^2 remaining variables arbitrarily, yielding 2^{n^2} models. Additionally, we have the two models of GRID_n if a is false. Therefore, AGRID_n has $2^{n^2}+2$ models. We show that these formulas separate MICE and T_{sparse}.

Theorem 6.5. *There are* MICE *proofs of* AGRID_n *of size* $O(n)$*, but all* T_{sparse} *proofs require size* $2^{\Omega(n)}$.

Proof sketch. In MICE we can consider the two cases with assumptions $a = 0$ and $a = 1$ separately, prove both claims efficiently, and compose them, yielding a short MICE proof.

T_{sparse} however is not flexible enough to allow this case distinction and we can argue for an exponential lower bound by showing that each T_{sparse} proof contains a tensor with exponentially many non-zero entries. For that, we consider an arbitrary T_{sparse} proof π for AGRID_n. The primal graph of AGRID_n contains an $n \times n$ grid and has therefore treewidth at least n. As T_{dense} and T_{sparse} only use different measures for proof size, we can use Theorem 5.1 and conclude that π contains at least one tensor T_V^F with $|V| = \Omega(n)$. We argue that even a sparse representation of this tensor has to be of exponential size. For that we distinguish two cases.

- *Case 1:* $a \in V$. Then, one half of the entries in the matrix correspond to assignments that set a to true. F can only contain clauses from AGRID_n that are all satisfied if a is true. Thus, at least every second entry of T_V^F is not 0.
- *Case 2:* $a \notin V$. We have $a \in \mathsf{vars}(F)$ as F has to contain at least one clause of AGRID_n and every clause contains a. By assigning a to true, we can extend every assignment $\alpha \in \langle V \rangle$ such that the extension satisfies F. Therefore, T_V^F does not contain a single zero entry.

In both cases we get $|T_V^F| = 2^{\Omega(n)}$. Thus, the whole proof π has also size $2^{\Omega(n)}$. □

Consequently, we obtain that MICE is exponentially separated from T_{sparse}. Note that this separation is shown for formulas with many models. Intuitively, T_{sparse} is a good choice when the tensors are sparse, i.e. only have few non-zero tensor entries. Hence, it might appear natural to conjecture that T_{sparse} proofs are short on CNFs with few models. However, we can show that even for unsatisfiable formulas – i.e. formulas with the minimal number of models – T_{sparse} proofs have to be exponentially larger than MICE proofs in some cases.

We demonstrate such a lower bound on the unsatisfiable *pebbling formulas* on pyramidal graphs. These CNFs are prominent in proof complexity and provide an exponential separation between (daglike) resolution and treelike resolution (Bonet et al. 1998; Ben-Sasson, Impagliazzo, and Wigderson 2004). For these CNFs we can show:

Theorem 6.6. *The* PEB_n *formulas have* MICE *proofs of size* $n^{O(1)}$*, but require* T_{sparse} *proofs of size* $2^{\Omega(n)}$.

Hence, even on CNFs with few models, and most extremely on unsatisfiable CNFs, MICE can be exponentially stronger than T_{sparse}.

7 Conclusion

Our theoretical findings presented here are motivated by recent practical work on tensor-network-based model counting solvers. So far, most practical implementations employ dense tensor representations. A recent report shows that sparse tensor network implementation in SQL can outperform dense implementations on #SAT (Blacher et al. 2023). Our theoretical analysis explains these observations and demonstrates that sparse implementations can perform exponentially better than dense ones (Theorem 6.2). We therefore believe that sparse tensor contraction algorithms, which are just starting to see practical usage, hold enormous potential for model counting and other applications. This potential is further illustrated by experiments confirming our theoretical findings (Staudt et al. 2025).

In addition to dense and sparse tensor representations there are further techniques how to efficiently represent tensors, e.g. through algebraic decision diagrams (ADDs) as used experimentally in (Dudek, Phan, and Vardi 2020a; Dudek, Shrotri, and Vardi 2022). It appears interesting to further explore the relative strength of these different tensor representations, both theoretically and practically.

Acknowledgements

The authors are supported by the Carl-Zeiss Foundation in the project Interactive Inference.

References

Aji, S. M.; and McEliece, R. J. 2000. The generalized distributive law. *IEEE Transactions on Information Theory*, 46(2).

Atserias, A.; Fichte, J. K.; and Thurley, M. 2011. Clause-Learning Algorithms with Many Restarts and Bounded-Width Resolution. *J. Artif. Intell. Res.*, 40: 353–373.

Bacchus, F.; Dalmao, S.; and Pitassi, T. 2003. Algorithms and Complexity Results for #SAT and Bayesian Inference. In *FOCS 2003*, 340–351. IEEE Computer Society.

Baluta, T.; Chua, Z. L.; Meel, K. S.; and Saxena, P. 2021. Scalable Quantitative Verification For Deep Neural Networks. In *ICSE 2021*, 312–323. IEEE.

Beame, P.; Kautz, H. A.; and Sabharwal, A. 2004. Towards Understanding and Harnessing the Potential of Clause Learning. *J. Artif. Intell. Res.*, 22: 319–351.

Ben-Sasson, E.; Impagliazzo, R.; and Wigderson, A. 2004. Near Optimal Separation Of Tree-Like And General Resolution. *Comb.*, 24(4): 585–603.

Beyersdorff, O.; Fichte, J. K.; Hecher, M.; Hoffmann, T.; and Kasche, K. 2024. The Relative Strength of #SAT Proof Systems. In *SAT 2024*, volume 305 of *LIPIcs*, 5:1–5:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Beyersdorff, O.; Hoffmann, T.; and Spachmann, L. N. 2023. Proof Complexity of Propositional Model Counting. In *SAT 2023*, volume 271 of *LIPIcs*, 2:1–2:18. Schloss Dagstuhl.

Beyersdorff, O.; Hoffmann, T.; and Spachmann, L. N. 2024. Proof Complexity of Propositional Model Counting. *J. Satisf. Boolean Model. Comput.*, 15(1): 27–59.

Biamonte, J.; and Bergholm, V. 2017. Tensor Networks in a Nutshell. *arXiv*, 1708.00006.

Biamonte, J. D.; Morton, J.; and Turner, J. W. 2015. Tensor Network Contractions for #SAT. *Journal of Statistical Physics*, 160(5): 1389–1404.

Blacher, M.; Klaus, J.; Staudt, C.; Laue, S.; Leis, V.; and Giesen, J. 2023. Efficient and Portable Einstein Summation in SQL. *Proceedings of the ACM on Management of Data*, 1(2): 121:1–121:19.

Bonet, M. L.; Esteban, J. L.; Galesi, N.; and Johannsen, J. 1998. Exponential Separations between Restricted Resolution and Cutting Planes Proof Systems. In *FOCS 1998*, 638–647. IEEE Computer Society.

Bryant, R. E.; Nawrocki, W.; Avigad, J.; and Heule, M. J. H. 2023. Certified Knowledge Compilation with Application to Verified Model Counting. In *SAT 2023*, volume 271 of *LIPIcs*, 6:1–6:20. Schloss Dagstuhl.

Buss, S.; and Nordström, J. 2021. Proof Complexity and SAT Solving. In *Handbook of Satisfiability*, Frontiers in Artificial Intelligence and Applications, 233–350. IOS Press.

Capelli, F. 2017. Understanding the complexity of #SAT using knowledge compilation. In *LICS 2017*, 1–10. IEEE Computer Society.

Capelli, F. 2019. Knowledge Compilation Languages as Proof Systems. In *SAT 2019*, volume 11628, 90–99. Springer.

Capelli, F.; Lagniez, J.; and Marquis, P. 2021. Certifying Top-Down Decision-DNNF Compilers. In *AAAI 2021*, 6244–6253. AAAI Press.

Chede, S.; Chew, L.; and Shukla, A. 2024. Circuits, Proofs and Propositional Model Counting. In *FSTTCS 2024*, volume 323 of *LIPIcs*, 18:1–18:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Chi-Chung, L.; Sadayappan, P.; and Wenger, R. 1997. On optimizing a class of multi-dimensional loops with reduction for parallel execution. *Parallel Processing Letters*, 7(02): 157–168.

Cook, S. A.; and Reckhow, R. A. 1979. The relative efficiency of propositional proof systems. *Journal of Symbolic Logic*, 44(1): 36–50.

Darwiche, A. 1999. Compiling Knowledge into Decomposable Negation Normal Form. In *IJCAI 1999*, 284–289. Morgan Kaufmann.

Darwiche, A.; and Marquis, P. 2002. A Knowledge Compilation Map. *J. Artif. Intell. Res.*, 17: 229–264.

de Beaudrap, N.; Kissinger, A.; and Meichanetzidis, K. 2020. Tensor Network Rewriting Strategies for Satisfiability and Counting. In *QPL 2020*, volume 340 of *EPTCS*, 46–59.

Dudek, J. M. 2021. *Planning and Execution for Discrete Integration*. Ph.D. thesis, Rice University.

Dudek, J. M.; Dueñas-Osorio, L.; and Vardi, M. Y. 2019. Efficient Contraction of Large Tensor Networks for Weighted Model Counting through Graph Decompositions. *CoRR*, abs/1908.04381.

Dudek, J. M.; Phan, V.; and Vardi, M. Y. 2020a. ADDMC: Weighted Model Counting with Algebraic Decision Diagrams. In *AAAI 2020*, 1468–1476. AAAI Press.

Dudek, J. M.; Phan, V. H. N.; and Vardi, M. Y. 2020b. DPMC: Weighted Model Counting by Dynamic Programming on Project-Join Trees. In *CP 2020*, volume 12333, 211–230. Springer.

Dudek, J. M.; Shrotri, A. A.; and Vardi, M. Y. 2022. DPSampler: Exact Weighted Sampling Using Dynamic Programming. In *IJCAI 2022*, 1795–1803. ijcai.org.

Dueñas-Osorio, L.; Meel, K. S.; Paredes, R.; and Vardi, M. Y. 2017. Counting-Based Reliability Estimation for Power-Transmission Grids. In *AAAI 2017*, 4488–4494. AAAI Press.

Fichte, J. K.; Hecher, M.; and Hamiti, F. 2021. The Model Counting Competition 2020. *ACM J. Exp. Algorithmics*, 26: 13:1–13:26.

Fichte, J. K.; Hecher, M.; and Roland, V. 2022. Proofs for Propositional Model Counting. In *SAT 2022*, volume 236 of *LIPIcs*, 30:1–30:24. Schloss Dagstuhl.

Fichte, J. K.; Hecher, M.; Thier, P.; and Woltran, S. 2020. Exploiting Database Management Systems and Treewidth for Counting. In *PADL 2020*, volume 12007, 151–167. Springer.

García-Sáez, A.; and Latorre, J. I. 2012. An exact tensor network for the 3SAT problem. *Quantum Inf. Comput.*, 12(3-4): 283–292.

Goral, A.; Giesen, J.; Blacher, M.; Staudt, C.; and Klaus, J. 2024. Model Counting and Sampling via Semiring Extensions. In *AAAI 2024*.

Heule, M.; Jr., W. A. H.; and Wetzler, N. 2013. Verifying Refutations with Extended Resolution. In *CADE 2013*, 345–359.

Korhonen, T.; and Järvisalo, M. 2023. SharpSAT-TD in Model Counting Competitions 2021-2023. *CoRR*, abs/2308.15819.

Kourtis, S.; Chamon, C.; Mucciolo, E. R.; and Ruckenstein, A. E. 2019. Fast counting with tensor networks. *SciPost Physics*, 7(5): 060.

Lagniez, J.; and Marquis, P. 2017. An Improved Decision-DNNF Compiler. In *IJCAI 2017*, 667–673. ijcai.org.

Latour, A. L. D.; Babaki, B.; Dries, A.; Kimmig, A.; den Broeck, G. V.; and Nijssen, S. 2017. Combining Stochastic Constraint Optimization and Probabilistic Programming - From Knowledge Compilation to Constraint Solving. In *CP 2017*, volume 10416, 495–511. Springer.

Markov, I. L.; and Shi, Y. 2008. Simulating quantum computation by contracting tensor networks. *SIAM Journal on Computing*, 38(3): 963–981.

Marques Silva, J. P.; Lynce, I.; and Malik, S. 2021. Conflict-Driven Clause Learning SAT Solvers. In *Handbook of Satisfiability*, Frontiers in Artificial Intelligence and Applications. IOS Press.

Orús, R. 2019. Tensor networks for complex quantum systems. *Nature Reviews Physics*, 1: 538–550.

Pipatsrisawat, K.; and Darwiche, A. 2011. On the power of clause-learning SAT solvers as resolution engines. *Artif. Intell.*, 175(2): 512–525.

Robertson, N.; and Seymour, P. D. 1986. Graph Minors. II. Algorithmic Aspects of Tree-Width. *J. Algorithms*, 7(3): 309–322.

Shi, W.; Shih, A.; Darwiche, A.; and Choi, A. 2020. On Tractable Representations of Binary Neural Networks. In *KR 2020*, 882–892.

Staudt, C.; Blacher, M.; Hoffmann, T.; Kasche, K.; Beyersdorff, O.; and Giesen, J. 2025. Exploiting Dynamic Sparsity in Einsum. In *NeurIPS 2025*.

Thurley, M. 2006. sharpSAT – Counting Models with Advanced Component Caching and Implicit BCP. In *SAT 2006*, volume 4121, 424–429. Springer.

Toda, S. 1991. PP is as Hard as the Polynomial-Time Hierarchy. *SIAM J. Comput.*, 20(5): 865–877.

Valiant, L. G. 1979. The Complexity of Computing the Permanent. *Theor. Comput. Sci.*, 8: 189–201.

Wetzler, N.; Heule, M.; and Jr., W. A. H. 2014. DRAT-trim: Efficient Checking and Trimming Using Expressive Clausal Proofs. In *SAT 2014*, volume 8561, 422–429. Springer.

Zhai, E.; Chen, A.; Piskac, R.; Balakrishnan, M.; Tian, B.; Song, B.; and Zhang, H. 2020. Check before You Change: Preventing Correlated Failures in Service Updates. In *NSDI 2020*, 575–589. USENIX Association.