MAJORITY-3SAT (and Related Problems) in Polynomial Time

Shyan Akmal
MIT CSAIL & EECS
Cambridge, MA, USA
naysh@mit.edu

Ryan Williams
MIT CSAIL & EECS
Cambridge, MA, USA
rrw@mit.edu

2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS) | 978-1-6654-2055-6/22/$31.00 ©2022 IEEE | DOI: 10.1109/FOCS52979.2021.00103

***Abstract*—Majority-SAT (a.k.a. MAJ-SAT) is the problem of determining whether an input n-variable formula in conjunctive normal form (CNF) has at least 2^(n-1) satisfying assignments. Majority-SAT and related problems have been studied extensively in various AI communities interested in the complexity of probabilistic planning and inference. Although Majority-SAT has been known to be PP-complete for over 40 years, the complexity of a natural variant has remained open: Majority-kSAT, where the input CNF formula is restricted to have clause width at most k.**

We prove that for every k, Majority-kSAT is in P; in fact, the problem can be solved in linear time (whereas the previous best-known algorithm ran in exponential time). More generally, for any positive integer k and constant p in (0,1) with bounded denominator, we give an algorithm that can determine whether a given k-CNF has at least p(2^n) satisfying assignments, in deterministic linear time. We find these results surprising, as many analogous problems which are hard for CNF formulas remain hard when restricted to 3-CNFs. Our algorithms have interesting positive implications for counting complexity and the complexity of inference, significantly reducing the known complexities of related problems such as E-MAJ-kSAT and MAJ-MAJ-kSAT. Our results immediately extend to arbitrary Boolean CSPs with constraints of arity k. At the heart of our approach is an efficient method for solving threshold counting problems by extracting and analyzing various sunflowers found in the corresponding set system of a k-CNF.

Exploring the implications of our results, we find that the tractability of Majority-kSAT is somewhat fragile, in intriguing ways. For the closely related GtMajority-SAT problem (where we ask whether a given formula has *greater than* 2^(n-1) satisfying assignments) which is also known to be PP-complete, we show that GtMajority-kSAT is in P for k at most 3, but becomes NP-complete for k at least 4. We also show that for Majority-SAT on k-CNFs with one additional clause of arbitrary width, the problem is PP-complete for k at least 4, is NP-hard for k=3, and remains in P for k=2. These results are counterintuitive, because the "natural" classifications of these problems would have been PP-completeness, and because there is a stark difference in the complexity of GtMajority-kSAT and Majority-kSAT for all k at least 4.

***Keywords*-satisfiability; majority-SAT; counting complexity; CNF-formulas; sunflowers**

I. Introduction

The complexity of #SAT, the problem of counting satisfying assignments to propositional formulas (a.k.a. "model counting" in the AI and SAT literature), has been intensely studied for decades. The pioneering work of Valiant [1] showed that #SAT is #P-complete already for 2-CNF formulas.

Of course, #SAT (and any other #P problem) is a function problem: up to $n+1$ bits need to be output on a given n-variable formula. A natural question is: how efficiently can output bits of the #SAT function be computed? Obvious choices are the *low-order bit*, which corresponds to the $\oplus$P-complete PARITY-SAT problem, and the *higher-order bits*. For CNF formulas, the highest-order bit of #SAT corresponds to the case where the #SAT value is 2^n, which is trivial for CNF formulas.[1] When the value is less than 2^n and #SAT outputs n bits, the high-order bit corresponds to MAJORITY-SAT, the problem of determining whether $\#\mathrm{SAT}(F) \geq 2^{n-1}$. It is more common to think of it as a probability threshold problem: given a formula F, is $\Pr_a[F(a) = 1] \geq 1/2$? Sometimes MAJORITY-SAT is phrased as determining whether or not $\Pr_a[F(a) = 1] > 1/2$; we will call this version GtMAJORITY-SAT to avoid confusion. Over CNF formulas (and more expressive Boolean representations), there is no essential difference between the two problems.[2]

MAJORITY-SAT (and GtMAJORITY-SAT) are the primary subjects of this paper. Gill [2] and Simon [3] introduced these problems along with the class PP, which consists of decision problems computing "high-order bits" of a #P function. They proved that MAJORITY-SAT on CNF formulas is PP-complete, and that $\#\mathsf{P} \subseteq \mathsf{P}^{\mathsf{PP}}$, showing that determining higher-order bits of a general #P function is as hard as computing the entire function.

The known proofs of PP-hardness for MAJORITY-SAT reduce to CNF formulas having clauses of arbitrarily large width. This raises the very natural question of whether MAJORITY-SAT remains hard over CNFs with fixed-width clauses. Intuition suggests that MAJORITY-kSAT should remain PP-hard for $k \geq 3$, by analogy with the NP-hardness of 3SAT, the PSPACE-hardness of Quantified 3SAT [4], the $\oplus$P-hardness of PARITY-3SAT[3], the Π_2P-hardness of Π_2-

[1]The only n-variable CNFs with 2^n satisfying assignments are those with no clauses. Of course, when the formula is DNF, the high-order bit problem is coNP-complete.

[2]See Section II for a discussion.

[3]This follows from the fact that there is a parsimonious reduction from SAT to 3-SAT [3].

2575-8454/21/$31.00 ©2021 IEEE
DOI 10.1109/FOCS52979.2021.00103

3SAT [5], and so on. Beyond the SAT problem, it is often true that the hardness of a problem can be preserved for "bounded width/degree" versions of the problem: for instance, the NP-hardness of 3-coloring holds even for graphs of degree at most 4 [6], and the #P-hardness of counting perfect matchings in graphs holds even for graphs of degree at most 3 [7]. Indeed, the more general problem: *given a 3-CNF F and an integer $k \geq 0$, determine if $\#SAT(F) \geq 2^k$* can readily be proved PP-complete. However, the same argument cannot be used to show that MAJORITY-SAT is PP-complete for 3-CNF F.[4]

Due to these subtleties, there has been significant confusion in the literature about the complexity of MAJORITY-kSAT, with several works asserting intractability for MAJORITY-3SAT and its variants, while others observing that the complexity of the problem remained open at the time [8]–[25].[5] This is a critical issue, as MAJORITY-SAT and its variants have been at the foundation of many reductions regarding the complexity of probabilistic planning, Bayesian inference, and maximum a posteriori problems in restricted settings, which are of great interest to various communities within AI. The true complexity of MAJORITY-kSAT (and related problems) has remained a central open question for these communities.

A. Our Results

Somewhat surprisingly, we show that MAJORITY-SAT over k-CNFs is in fact **easy**. In fact, for any constant $\rho \in (0,1)$, we can efficiently determine for a given k-CNF F whether $\Pr_{a \in \{0,1\}^n}[F(a) = 1] \geq \rho$ or not. Although we state our main results in this section, the proofs for most of these results are not included here and can instead be found in the full version of the paper [26].

Theorem 1.1: For every constant rational $\rho \in (0,1)$ and every constant $k \geq 2$, there is a deterministic linear-time algorithm that given a k-CNF F determines whether or not $\#\text{SAT}(F) \geq \rho \cdot 2^n$.

Theorem 1.1 is proved in [26, Section 5]. To our knowledge, the previous best-known algorithm for this problem ran in $2^{n-\Theta(n/k)}$ time, by running the best-known algorithm for $\#k$SAT [27], [28]. Of course, Theorem 1.1 does not mean that #P functions can be computed in polynomial time; rather, it shows that lower-order bits of $\#k$SAT are the more difficult ones[6] ([26, Section 5.1] formally describes how to use our algorithm to compute the high-order bits of $\#k$SAT in polynomial time). Even the lowest-order bit of $\#k$SAT is evidently harder: for every $k \geq 2$, PARITY-kSAT is known to be $\oplus$P-complete [29], so (by Toda's theorem [30]) the low-order bit of $\#k$SAT cannot be computed in BPP unless NP = RP.

Implications for Related Inference Problems: Given that MAJORITY-kSAT turns out to be easy, it is worth exploring whether related problems in literature are also easy or hard. In the relevant AI literature on the complexity of Bayesian inference and probabilistic planning, the following two problems are prominent in proving conditional lower bounds:

> E-MAJ-SAT: *Given n, n', and a formula φ over $n+n'$ variables, is there a setting to the first n variables of ϕ such that the majority of assignments to the remaining n' variables are satisfying assignments?*
> MAJ-MAJ-SAT: *Given n, n', and a formula φ over $n+n'$ variables, do a majority of the assignments to the first n variables of φ yield a formula where the majority of assignments to the remaining n' variables are satisfying?*

These problems may seem esoteric, but E-MAJ-SAT and related problems are used extensively in the relevant areas of AI, where an environment has inherently "random" aspects along with variables one can control, and one wants to "plan" the control variables to maximize the chance that a desired property holds (e.g. [31]–[33]). E-MAJ-SAT has also recently been used to study the complexity of verifying differential privacy [34].

Similarly, MAJ-MAJ-SAT applies in the context when one wants to know what is the chance that a random setting of control variables will yield a good chance that a property holds [35], [36]. For general CNF formulas, E-MAJ-SAT is NP^{PP}-complete [31], [37], [38] and MAJ-MAJ-SAT is PP^{PP}-complete [37]–[39]: roughly speaking, these results imply that both problems are essentially intractable, even assuming oracle access to a #SAT solver. There has also been significant confusion about whether E-MAJ-3SAT (the version restricted to 3-CNF) is NP^{PP}-complete or not [11], [14], [21], [22], [24], [25]. We prove that both E-MAJ-SAT and MAJ-MAJ-SAT dramatically decrease in complexity over k-CNF formulas.

Theorem 1.2: E-MAJ-2SAT $\in$ P, and for all $k \geq 3$, E-MAJ-kSAT is NP-complete.

Theorem 1.3: MAJ-MAJ-2SAT $\in$ P.

Theorem 1.2 is proved in [26, Section 6.1] and Theorem 1.3 is proved in [26, Section 6.2].

[4]The proof of PP-completeness (of the more general problem) follows from two facts: (a) the version of the problem for CNF formulas is PP-complete, by Gill and Simon, and (b) the reduction from CNF-SAT to 3SAT preserves the number of solutions. This proof cannot be used to show that determining $\#SAT(F) \geq 2^{n-1}$ is PP-complete for 3-CNF F, because the Cook-Levin reduction introduces many new variables (the variable n increases when going from CNF-SAT to 3SAT, thereby changing the target number of satisfying assignments).

[5]Even the second author is guilty of being confused: see the first comment at https://cstheory.stackexchange.com/questions/36660/status-of-pp-completeness-of-maj3sat.

[6]Indeed in some sense, "middle bits" are PP-hard: Bailey, Dalmau, and Koliatis [10], [12] showed 20 years ago that for all integers $t \geq 2$, determining whether $\#\text{SAT}(F) \geq 2^{n/t}$ (for 3-CNF F) is PP-complete.

Even the NP-completeness of E-MAJ-3SAT is good news, in some sense: Theorem 1.2 suggests that such counting problems could in principle be handled by SAT solvers, rather than needing #SAT solvers.

Greater-Than MAJORITY-SAT: The algorithms behind Theorem 1.1 can efficiently determine if the #SAT value of a k-CNF is *at least* a given fraction of the satisfying assignments. Recall the GtMAJORITY-SAT problem is to determine if the #SAT value is *greater than* a given fraction, and that over CNFs, there is no essential difference between the two problem variants. Another surprise is that, over k-CNFs, there is a difference between these problems for $k \geq 4$: the "greater than" version becomes NP-complete!

Theorem 1.4: For $k \leq 3$, GtMAJORITY-kSAT is in P.

Theorem 1.5: For $k \geq 4$, GtMAJORITY-kSAT is NP-complete.

Both Theorem 1.4 and Theorem 1.5 are proved in [26, Section 7.2].

*Adding One Long Clause Makes MAJORITY-**k**SAT Hard:* Given the surprisingly low complexity of these threshold counting problems over k-CNF formulas, it is natural to investigate what extensions of k-CNFs suffice in order for the problems to become difficult. This direction is also important for the considerable collection of results in AI whose complexity hinges on the difficulty of MAJORITY-SAT and its variants. We show that adding only *one* extra clause of arbitrary width is already enough to make MAJORITY-kSAT difficult, for $k \geq 3$.

Theorem 1.6: Deciding MAJORITY-SAT over k-CNFs with one extra clause of arbitrary width is in P for $k = 2$, NP-hard for $k = 3$, and PP-complete for $k \geq 4$.

This may look preposterous: how could adding only one long clause make MAJORITY-3SAT hard? Couldn't we simply try all $O(n)$ choices for picking a literal from the long clause, and reduce the problem to $O(n)$ calls to MAJORITY-3SAT with no long clauses? Apparently not! Remember that MAJORITY-3SAT *only* decides whether or not the fraction of satisfying assignments is at least $\rho \in (0, 1)$. This information does not help us determine the number of satisfying assignments to $O(n)$ subformulas accurately enough to refute the hardness of MAJORITY-3SAT with no long clauses.

B. Intuition

The ideas behind our algorithms arose from reconsidering the polynomial-time Turing reduction from #SAT to MAJORITY-SAT [2], [3], in the hopes of proving that MAJORITY-3SAT is hard. The key is to reduce the problem

$$\#\text{SATD} := \{(F, s) \mid \#\text{SAT}(F) \geq s\}$$

to MAJORITY-SAT. From there, one can binary search with #SATD to determine $\#\text{SAT}(F)$. The known reductions from #SATD to MAJORITY-SAT require that, given a desired $t \in [0, 2^n]$, we can efficiently construct a formula G_t on n variables with exactly t satisfying assignments.[7] Then, introducing a new variable x_{n+1}, the formula

$$H = (x_{n+1} \vee F) \wedge (\neg x_{n+1} \vee G_t)$$

will have $\#\text{SAT}(H) = \#\text{SAT}(F) + t$, out of 2^{n+1} possible assignments to H. Setting $t = 2^n - s$, it follows that $\#\text{SAT}(F) \geq s$ if and only if $\#\text{SAT}(H) \geq 2^n$, thereby reducing from #SATD to MAJORITY-SAT. Observe we can convert H into k-CNF, provided that both F and G_t are $(k-1)$-CNF.

However, this reduction fails miserably for k-CNF formulas, because for constant k and large n, there are many values $t \in [0, 2^n]$ for which *no* k-CNF formula G_t has exactly t satisfying assignments (observe that every k-CNF with at least one clause has at most $(1 - 1/2^k) \cdot 2^n$ satisfying assignments; therefore no such k-CNF formulas G_t exist, for all $t \in [2^n - 2^{n-k} - 1, 2^n - 1]$). Moreover, every k-CNF containing d disjoint clauses (d clauses sharing no variables) has at most $(1 - 1/2^k)^d \cdot 2^n$ satisfying assignments. But "most" k-CNF formulas (say, from the typical random k-SAT distributions) *will* have *large* disjoint sets of clauses (say, of size $\Omega(n)$). So for "most" formulas, we can quickly determine that $\#\text{SAT}(F) < \rho \cdot 2^n$ for constant $\rho > 0$, by finding a large enough disjoint clause set.

What remains is a rather structured subset of k-CNF formulas. If the maximum possible size of a disjoint clause set is small, then there is a small set of variables that "hit" all other clauses (otherwise, the set would not be maximal). That is, there is a small set of variables that have non-empty intersection with every clause. This kind of small hitting set can be very algorithmically useful for solving #SAT. For example, if $k = 2$, then every assignment to the variables in a small hitting set simplifies the given formula into a 1-CNF. In other words, when there is a small hitting set, we can reduce the computation of #2SAT to a small number of calls to #1SAT, each of which can be solved in polynomial time. This is essentially how our algorithm for MAJORITY-2SAT works.

The situation quickly becomes more technically complicated, as k increases. When $k = 3$, setting all variables in a small hitting set merely simplifies the formula to a 2-CNF, but #SAT is already #P-hard for 2-CNF formulas. To get around this issue, we consider more generally *sunflowers* within the k-CNF: collections of sets which all share the same pairwise intersection (called the core).

Sunflowers in a formula can be useful in bounding the fraction of satisfying assignments. To give a simple example, if the entire formula was a sunflower with a single literal ℓ in its core, then the fraction of satisfying assignments is at least $1/2$ (because setting ℓ true already satisfies the formula).

[7] A standard way to do this is to make a formula G_t which is true if and only if its variable assignment, construed as an integer in $[1, 2^n]$, is at most t. But constructing such a formula requires arbitrary width CNFs.

Our algorithms seek out large sunflowers on disjoint clauses in k-CNF formulas, to get tighter and tighter bounds on the fraction of satisfying assignments. When a formula does not have many such sunflowers, the formula is structured enough that we can find a small hitting set of variables and use the ideas discussed earlier.

Intuition for Theorem 1.1: Here we provide an intuitive idea of how our main algorithm works to determine whether a k-CNF has at least a ρ-fraction of satisfying assignments. Given a Boolean formula Φ on n variables, let $\Pr[\Phi]$ denote the probability a uniform random assignment to the variables of Φ is satisfying. For a given k-CNF φ, we want to decide whether the inequality

$$\Pr[\varphi] \geq \rho$$

holds or not. We will do this by building up a special $(k-2)$-CNF ψ on the same variable set, where each clause of ψ is contained in a clause of φ. We split the probability calculation into

$$\Pr[\varphi] = \Pr[\varphi \wedge \psi] + \Pr[\varphi \wedge \neg\psi]$$

and use the fact that

$$\Pr[\varphi \wedge \psi] \leq \Pr[\varphi] = \Pr[\varphi \wedge \psi] + \Pr[\varphi \wedge \neg\psi]. \quad (1)$$

Intuitively, we will construct ψ in such a way that $\Pr[\varphi \wedge \neg\psi] < \varepsilon_1$ for an *extremely* small $\varepsilon_1 > 0$, so that it is possible to reduce the problem of determining $\Pr[\varphi] \geq \rho$ to the problem of determining $\Pr[\varphi \wedge \psi] \geq \rho$. In other words, we can reduce THR$_\rho$-kSAT on φ to THR$_\rho$-kSAT on $\varphi \wedge \psi$.

This reduction is helpful because the clauses of ψ are subclauses appearing frequently in φ, so the formula $\varphi \wedge \psi$ simplifies to a smaller formula than φ. Additionally, $\varphi \wedge \psi$ has a smaller solution space than φ, so intuitively it becomes easier to check if the resulting formula has fewer than a ρ-fraction of satisfying assignments. More precisely, it follows from (1) that if $\Pr[\varphi \wedge \psi] \geq \rho$ then $\Pr[\varphi] \geq \rho$ as well. The more surprising result is that we can construct ψ so that, if $\Pr[\varphi \wedge \psi] < \rho$, then we can in fact infer that $\Pr[\varphi \wedge \psi] < \rho - \varepsilon_2$ for some $\varepsilon_2 > \varepsilon_1$. Hence by (1) we can deduce that $\Pr[\varphi] < \varepsilon_1 + \rho - \varepsilon_2 < \rho$. We construct the clauses of ψ by taking cores of large sunflowers in φ. Defining what counts as "large" depends on quite a few parameters, so the analysis becomes rather technical.

C. Paper Organization

In Section II, we formally define the problems we are considering, introduce notation, and discuss more related work. In Section III we present a simple algorithm for solving MAJORITY-2SAT in linear time, and in Section IV we extend this algorithm to solve MAJORITY-3SAT in linear time. Proofs of our remaining results can be found in the full version of the paper [26]. We conclude in Section V with a discussion of several intriguing open problems.

II. Preliminaries

We assume basic familiarity with computational complexity, including concepts such as PP and #P [40]. For a formula F on n variables, let #SAT(F) be its number of satisfying assignments as an integer in $[0, 2^n]$.

CNF Formulas. A literal is a Boolean variable or its negation, a *clause* is a disjunction of literals, and a *CNF formula* is a conjunction of clauses. The *width* of a clause is the number of literals it contains. Given an integer w, a w-*clause* is just a clause of width w. Given a positive integer k, we say a formula is a k-CNF if every clause in the formula has width *at most* k. We stress that we allow our k-CNFs to have clauses of length *up to* k: clauses of width $1, \ldots, k$ are allowed. An empty CNF formula evaluates to $\top$, meaning it is always true. An empty clause evaluates to $\bot$, meaning it is always false. Given a CNF formula φ, we let $|\varphi|$ denote the size of the formula, which is just the sums of the widths of all clauses in φ.

We remark that all of the results in this paper that hold for k-CNF formulas also hold for Boolean constraint satisfaction problems (CSPs), over arbitrary constraints of arity at most k. This is because each constraint of such a CSP can be converted into an equivalent k-CNF over the same variables.

GtMAJORITY-SAT vs MAJORITY-SAT. Here we briefly describe how to reduce between these two problems. To reduce from GtMAJORITY-SAT to MAJORITY-SAT given an n-variable formula F, introduce n new variables $y_1, \ldots, y_n$ and map F to $F' := (y_1 \vee \cdots \vee y_n) \wedge F$. Then #SAT($F$) $\geq 2^{n-1}+1$ implies #SAT(F') $\geq (2^n-1)(2^{n-1}+1) = 2^{2n-1} + 2^n - 2^{n-1} - 1 > 2^{2n-1}$ and #SAT(F) $\leq 2^{n-1}$ implies #SAT(F') $\leq (2^n-1)2^{n-1} = 2^{2n-1} - 2^{n-1} < 2^{2n-1}$.

To reduce from MAJORITY-SAT to GtMAJORITY-SAT given an n-variable F, introduce one new variable x_{n+1}, let G be an n-variable formula with precisely $2^n - 2^{n-1} + 1$ satisfying assignments, and set $F' := (\neg x_{n+1} \vee F) \wedge (x_{n+1} \vee G)$. Then #SAT($F'$) = #SAT($F$) + $2^n - 2^{n-1} + 1$. When #SAT(F) $\geq 2^{n-1}$, we have #SAT(F') $\geq 2^n + 1$, and when #SAT(F) $\leq 2^{n-1} - 1$ we have #SAT(F') $\leq 2^n$. (We can increase the gap by increasing the number of additional variables.) Both reductions need unbounded width CNF formulas.

Threshold SAT. We have already defined the MAJORITY-kSAT problem. To discuss problems of detecting fractions of satisfying assignments at other thresholds besides $1/2$, we introduce the following problem.

Definition 2.1 (Threshold SAT): For any positive integer k and threshold $\rho \in (0, 1)$, the THR$_\rho$-kSAT problem is the following task: given a k-CNF formula φ on n variables, determine if the inequality #SAT(φ) $\geq \rho \cdot 2^n$ holds.

In our algorithms, we will often make use of the following structures in CNF formulas.

Definition 2.2 (Consistent Literal Set): Given a set of literals, we say the set is *consistent* if the set does not simultaneously include x and $\neg x$ for any variable x.

Definition 2.3 (Variable Disjoint Set): Given a set S of clauses, we say S is a (variable) *disjoint set* if for every pair C, C' of distinct clauses of S, C and C' share no variables.

We will also utilize the following simple observations about CNFs formulas.

Proposition 2.4: Let F be a CNF formula on n variables, construed as a set of clauses. Suppose there is a $\rho \in (0,1)$ and a subset F' of the clauses of F such that F' contains $r \leq n$ variables and $\#SAT(F') \leq \rho \cdot 2^r$. Then $\#SAT(F) \leq \rho \cdot 2^n$.

Proof: Note that $F = F' \wedge G$, for some formula G. Given a fixed F', the number of satisfying assignments to F is maximized when G is a tautology, having 2^n satisfying assignments. (Since F is over n variables, we can take $G = (x_1 \vee \neg x_1) \wedge \cdots \wedge (x_n \vee \neg x_n)$.) Even in such a case, $\#\text{SAT}(F) \leq \#\text{SAT}(F') \cdot 2^{n-r} \leq \rho \cdot 2^n$. ∎

Proposition 2.5: Given a 1-CNF formula F (i.e. F is a conjunction of literals), the number of satisfying assignments to F can be computed in linear time.

Proof: Let k be the number of 1-clauses (literals) in F. If F contains both a variable and its negation, then F is unsatisfiable, and the number of satisfying assignments is 0. Otherwise, the set of literals in F is consistent, and the number of satisfying assignments is 2^{n-k}. In either case, we can compute the desired quantity by scanning through the clauses in F once. ∎

As a final pieces of notation, we write $A = \text{poly}(B)$ to denote that $A \leq B^c$ for some constant $c > 0$.

A. Comparison With Related Work

Several works [41]–[46] have considered the task of approximately counting satisfying assignments to CNF formulas. In particular, given a constant $\varepsilon \in (0,1)$ and CNF formula φ, we seek to output an estimate that is within ε of the true fraction of assignments of φ which are satisfying.[8] In general, the estimates provided by such algorithms may be strictly more or less than the true fraction of satisfying assignments, so such approximation algorithms cannot be used to solve problems like MAJORITY-kSAT.

However, the starting point of our work, the MAJORITY-2SAT and MAJORITY-3SAT algorithms, uses methods very similar to those of Trevisan [43], who showed that for any fixed integer k one can *approximately* count the fraction of satisfying assignments in a k-CNF formula efficiently, by working with maximal disjoint sets of clauses.[9] Given a desired additive approximation error ε, Trevisan's approach shows that every k-CNF can be *approximated* by a special kind of decision tree of $f(\varepsilon, k) \leq O(1)$ size and depth, where the internal nodes are labeled by variables and the leaves are labeled with 1-CNFs. Computing the exact fraction of satisfying assignments for such a decision tree is simple to do in linear time, and Trevisan uses this count to obtain an ε-additive approximation of the true fraction of satisfying assignments.

In our algorithms, we also implicitly (and for MAJORITY-2SAT, E-MAJ-2SAT, and MAJ-MAJ-2SAT, explicitly) construct such decision tree representations, and we also use the fact that one can count satisfying assignments exactly on such decision tree representations. However, for MAJORITY-kSAT where $k \geq 3$, our algorithms and analysis have to dig further into the problem and take advantage of the structure of the decision tree itself. Informally, we show there are "gaps" in the possible #SAT values of such representations. Very roughly speaking, these gaps are part of what allows us to solve the exact threshold counting problem for k-CNFs in polynomial time, "as if" it were an additive approximation problem. Still, many other cases arise in determining the fraction exactly that are irrelevant in approximations.

More generally, our algorithms rely on extracting sunflowers from various subformulas. Sunflower lemmas have been used previously for obtaining additive approximations to the fraction of satisfying assignments of disjunctive normal form (DNF) formulas and related problems such as DNF sparsification and compression [41], [44]–[46]. These results focus on formulas of super-constant width, whereas our work is specialized to CNFs of constant width. Due to our hardness results, one *cannot* extend our algorithms to 3CNFs with even one unbounded width clause, unless P = NP.

III. Threshold SAT for 2-CNFs in Linear Time

As a warm-up, we begin with a simple linear-time algorithm for MAJORITY-2SAT (even THR_ρ-2SAT, for every $\rho \geq 1/\text{poly}(n)$) that illustrates a few of the ideas.[10]

Theorem 3.1: For every rational $\alpha \in (0,1)$, there is an $m \cdot \text{poly}(1/\alpha)$-time algorithm that, given any 2-CNF formula F on n variables and m clauses, decides whether $\#\text{SAT}(F) \geq \alpha \cdot 2^n$ or not. Furthermore, when $\#\text{SAT}(F) \geq \alpha \cdot 2^n$ is true, the algorithm outputs $\#\text{SAT}(F)$, along with a a decision tree representation for F of $\text{poly}(1/\alpha)$ size. The internal nodes are labeled by variables and leaves are labeled by 1-CNFs.

Proof: For each $\alpha \in (0,1)$, define $c(\alpha) := 1 + \lceil \log_{4/3}(1/\alpha) \rceil$. Note that $c(\alpha) \leq O(\log \frac{1}{\alpha})$.

Given a 2-CNF F, start by finding a *maximal disjoint set* of clauses S. That is, treat the clauses as sets (ignoring literal signs) and find a set S of clauses such that (a) every pair of clauses in S share no variables and (b) all other clauses in F contain at least one variable occurring in S. This can be done by greedily choosing the set S (picking disjoint clauses until we cannot) in time $O(m \cdot |S|)$. We argue that we can stop once $|S|$ exceeds $c(\alpha)$.

[8] One can also consider *multiplicative* approximations to #SAT, but this task is NP-hard. See for example [47], [48].

[9] In fact, the second author devised an algorithm for MAJORITY-2SAT in 2004, inspired by Trevisan's work, but only recently (with the help of the first author) found a way to generalize to MAJORITY-3SAT and beyond.

[10] The second author has known of this result since around 2004; see Section 7 of [49].

Case 1: Suppose $|S| > c(\alpha)$. Then we claim that $\#\text{SAT}(F) < \alpha \cdot 2^n$. Note that each of the clauses in S are over disjoint variables, so each clause in S reduces the total number of satisfying assignments by $3/4$. By our choice of $c(\alpha)$, we have $(3/4)^{c(\alpha)} < \alpha$. Therefore, less than an α-fraction of the possible assignments satisfy the subformula S, and by Proposition 2.4, we can return **NO**.

Case 2: Otherwise, S is a maximal disjoint set of clauses with $|S| \leq c(\alpha)$. Since every clause in F contains at least one variable occurring in S, it follows that, when we plug in any assignment to the variables of S, the remaining formula is a 1-CNF. Therefore, if we try all of the at most

$$3^{c(\alpha)} \leq O(3^{(\log(\frac{1}{\alpha})/\log(4/3))}) \leq O((1/\alpha)^{3.82})$$

satisfying assignments to the clauses of S, and solve #SAT on the remaining 1-CNF formula in $O(m)$ time (Proposition 2.5), we can determine the number of satisfying assignments *exactly* in this case.

Note that **Case 1** of this algorithm only occurs when $\#\text{SAT}(F) < \alpha \cdot 2^n$. Consequently, whenever $\#\text{SAT}(F) \geq \alpha \cdot 2^n$ we fall into **Case 2** and our algorithm reports the exact count of satisfying assignments. The overall run time of this algorithm is $m \cdot \text{poly}(1/\alpha)$. ■

It is interesting to contrast the above result with the result of Leslie Valiant that $\oplus$2SAT is $\oplus$P complete [29]. Valiant's result implies that computing the low-order bit of #2SAT in polynomial time would imply that NP $\subseteq$ BPP. Our result shows that computing the low-order bit of #2SAT looks much more difficult than higher-order bits.

IV. Majority SAT for 3-CNFs in Linear Time

Recall from Theorem 2.1 that given a positive integer k and parameter $\rho \in (0,1)$, we define the "threshold SAT" problem THR$_\rho$-kSAT to be the task of deciding whether at least a ρ-fraction of assignments to a given k-CNF are satisfying. For example, the MAJORITY-3SAT problem discussed previously is equivalent to THR$_{1/2}$-3SAT. In Section IV-A and Section IV-B we show how to extend the ideas from Section III with a subformula detection argument to prove the following result.

Theorem 4.1: For every constant $\rho \in [1/2, 1]$, we can decide in polynomial time if a given 3-CNF on n variables has at least $\rho \cdot 2^n$ satisfying assignments.

Generalizations of this theorem, which prove analogous results for k-CNFs, for fixed positive integers $k \geq 3$, and arbitrary rational thresholds $\rho \in (0,1)$ with constant denominator, are proved in the full version of the paper [26, Section 4.3 & Section 5]. This section in this paper is included only to present arguments which are less technically challenging than the proofs of the more general results, and therefore hopefully more accessible, while still resolving the complexity of MAJORITY-3SAT.

A. Thresholds Greater than One-Half

Building on the MAJORITY-2SAT algorithm of Theorem 3.1, we propose the following natural generalization to 3-CNFs. In the following, a "disjoint set of clauses" refers to a variable disjoint set (see Theorem 2.3).

> **Algorithm A.** *(With two unspecified constants c_1 and c_2.)*
> Given a 3-CNF F, find a maximal disjoint set S of clauses of F. If $|S|$ exceeds a certain constant c_1, then output **NO**.
> For all $7^{|S|}$ SAT assignments A to the clauses in S, let F_A be the 2-CNF induced by assignment A, and search for a maximal disjoint set S_A of 2-clauses in F_A. If $|S_A|$ exceeds a certain constant c_2, then output **NO**. Otherwise, try all SAT assignments A' to the clauses in S_A. For each 1-CNF formula induced by an A', count solutions to the 1-CNF in polynomial time.
> Return **YES** if and only if the total number of solutions counted (over all assignments A) is at least $\rho \cdot 2^n$.

First, we prove that **Algorithm A** correctly decides $\#\text{SAT}(F) \geq \rho \cdot 2^n$ for all fractions $\rho > 1/2$.

Theorem 4.2: For every $\varepsilon \in (0, 1/2]$, we can decide in $\text{poly}(1/\varepsilon, n)$ time if a given 3-CNF on n variables has at least $(1/2+\varepsilon) \cdot 2^n$ satisfying assignments. Moreover, given any 3-CNF with at least $(1/2+\varepsilon) \cdot 2^n$ satisfying assignments, we can report the exact number of satisfying assignments.

The ability to report the exact number of satisfying assignments will (provably) no longer hold when we consider the case of $\varepsilon = 0$, in the next subsection. This is the major reason why we have treated the two cases separately.

To prove Theorem 4.2, we show that by setting c_1, c_2 appropriately in **Algorithm A**, we can decide if there are at least $\rho \cdot 2^n$ SAT assignments for $\rho > 1/2$. We first prove a lemma regarding the sizes of maximal disjoint sets in formulas obtained by assigning variables.

Lemma 4.3: Let $\rho > 1/2$, and let S be a maximal disjoint set of k-clauses in a k-CNF F. Suppose F has at least $\rho \cdot 2^n$ satisfying assignments. For all possible assignments A to the variables of S, and for every induced 2-CNF F_A obtained by assigning A to S, F_A must contain a maximal disjoint set of $(k-1)$-clauses of size less than $2^k |S| \ln(1/(\rho - 1/2))$.

Proof: The proof is by contrapositive. Let $\varepsilon > 0$ be such that $\rho := 1/2 + \varepsilon$, and let S be a maximal disjoint set of k-clauses in a given k-CNF F. Suppose there is an assignment A to the variables of S such that F_A has a maximal disjoint set of $(k-1)$-clauses of size at least $K := 2^k |S| \ln(1/\varepsilon)$. By the pigeonhole principle, there exists some literal $\ell \in \{x, \neg x\}$, coming from a variable x in the maximal disjoint set S, and a set T_ℓ of at least $K/(2|S|) = 2^{k-1} \ln(1/\varepsilon)$ clauses in F of the form

$$(\ell \vee a_{i,1} \vee \cdots \vee a_{i,k-1})$$

where the variables of $a_{i,j}$ are all *distinct* over all $i = 1, \ldots, |T_\ell|$ and $j = 1, \ldots, k-1$. That is, the subformula T_ℓ of F has in total $1 + (k-1)r$ distinct variables, where $r := |T_\ell|$.

Since $r \geq 2^{k-1} \ln(1/\varepsilon)$, the fraction of satisfying assignments in T_ℓ is at most

$$\frac{2^{(k-1)r} + (1 - 1/2^{k-1})^r \cdot 2^{(k-1)r}}{2^{1+(k-1)r}}$$

which simplifies to

$$\frac{1}{2} + \frac{1}{2} \cdot (1 - 1/2^{k-1})^r \leq \frac{1}{2} + \frac{\varepsilon}{2} < \rho,$$

where the $2^{(k-1)r}$ term comes from the case where ℓ is true, and $(1-1/2^{k-1})^r \cdot 2^{(k-1)r}$ term comes from the case where ℓ is false. Therefore, in such a case, F must have less than a ρ fraction of satisfying assignments by Proposition 2.4. ■

We can apply Lemma 4.3 by arguing that, if any subformula F_A of F has a "large" maximal disjoint set of 2-clauses, then we can output NO when $\rho > 1/2$. Otherwise, every F_A has a "small" maximal disjoint set of 2-clauses, and **Algorithm A** works in that case.

Proof of Theorem 4.2: Let $\varepsilon > 0$. We consider **Algorithm A** with constants $c_1 := 10$ and $c_2 := 72 \ln(1/\varepsilon)$.

If $|S| > 10$, then the fraction of satisfying assignments to the subformula S is less than $1/2$, therefore F has less than a $1/2$ fraction by Proposition 2.4. Therefore in step 2 of Algorithm A, we can report **NO**.

Otherwise, $|S| \leq 9$. Suppose we try all possible satisfying assignments to S (there are at most $3^{|S|}$) and suppose there is some induced formula F_A with a maximal disjoint set S_A of at least $72 \cdot \ln(1/\varepsilon)$ clauses. By Lemma 4.3 we can deduce that F has less than an $\rho := 1/2 + \varepsilon$ fraction of satisfying assignments, and can report **NO**.

In the remaining case, every induced formula F_A has a maximal disjoint set S_A of less than $72 \cdot \ln(1/\varepsilon)$ clauses. By trying all possible SAT assignments to each S_A (there are $3^{|S_A|} \leq \text{poly}(1/\varepsilon)$ such assignments) we can count the number of satisfying assignments for each of the remaining 1-CNF formulas in linear time, and determine the exact number of satisfying assignments by taking the sum of all such counts. ■

B. Threshold of One-Half

We now we turn to the case of solving THR_ρ-3SAT for threshold value $\rho = 1/2$.

When $\rho = 1/2$, **Algorithm A** does not work correctly in all cases (regardless of how its parameters are set). Consider a 3-CNF formula F in which every clause contains a common variable x occurring positively. This is trivially a YES-instance for MAJORITY-3SAT. (Note we cannot efficiently compute the number of satisfying assignments exactly in this case, as it would solve the #2SAT problem in polynomial time!) Running **Algorithm A** on F, it will find an S with $|S| = 1$, since x appears in all clauses. When we try all satisfying assignments to S, and x is set true, the formula becomes a tautology. But when x is set false, the formula becomes an arbitrary 2-CNF, with potentially a very large disjoint clause set. Regardless of the size of that clause set, the original F is still a YES instance, even if *all* of the clauses in the remaining 2-CNF are disjoint. So, an algorithm for MAJORITY-3SAT needs to be able to account for this sort of behavior, where a single literal appears in many clauses.

To handle this case, we introduce a check for another type of "bad" subformula.

Lemma 4.4: Let $\ell \in \{x, \neg x\}$ be a literal, and let

$$S = \{(\ell \vee a_1 \vee b_1), \ldots, (\ell \vee a_t \vee b_t), (u \vee v \vee w)\}$$

be a set of clauses with the following properties:

- For all $i, j \in [t]$, a_i and b_j are literals from $2t$ distinct variables, all of which are different from x.
- The literal ℓ does not appear in $(u \vee v \vee w)$. (However, $\neg\ell$ may appear in $(u \vee v \vee w)$.)

Then for all $t \geq 8$, S has less than 2^{r-1} satisfying assignments, where r is the total number of variables occurring in S.

Proof: Let r be the total number of variables in S; note that $r \geq 2t+1$. When ℓ is set to false, the t clauses $(a_i \vee b_i)$ are all disjoint, so the formula S has at most $(3/4)^t \cdot 2^{r-1}$ satisfying assignments over the remaining $r-1$ variables. When ℓ is true, the clause $(u \vee v \vee w)$ remains, so (over the remaining $r-1$ variables) the number of satisfying assignments in this case is at most $(7/8) \cdot 2^{r-1}$. (Note that if the literal $\neg\ell$ appears in $(u \vee v \vee w)$, then the fraction is $3/4$, which is only better for us.) For $t \geq 8$, the total number of satisfying assignments is therefore $((3/4)^t + 7/8) \cdot 2^{r-1} < 2^{r-1}$. ■

For t sufficiently large, Lemma 4.4 can be used to show that S has less than $(7/16 + \varepsilon)2^r$ satisfying assignments for any desired $\varepsilon > 0$.

MAJ3SAT in P*:* We are now ready to give a polynomial-time algorithm for deciding if a 3-CNF has at least 2^{n-1} satisfying assignments. For ease of reading, here we will describe the algorithm alongside its analysis.

Given a 3-CNF F on n variables, we start by checking if there is a common literal ℓ appearing in every clause of F. In this case we output **YES**, as any such formula is satisfied by at least half of its assignments.

After this point, we know:

($\star$) For every literal ℓ there is at least one clause in F that does not contain ℓ.

Next, we find a maximal disjoint set S among the 3-clauses in F. If $|S| \geq 6$ then, since $(7/8)^6 < 0.449 < 1/2$, we can output **NO** by Proposition 2.4.

Otherwise, we know that $|S| \leq 5$. For each of the $7^{|S|}$ satisfying assignments A to the clauses of S, we do the following:

For each 2-CNF formula F_A induced by an assignment A on the variables of S in F, find a maximal disjoint set S_A over the 2-clauses in F_A.

1) We claim that, if there is an assignment A such that $|S_A| \geq 48|S|+2$, then F must contain less than 2^{n-1} satisfying assignments. Hence we can output **NO** in this case.

   This paragraph proves the claim. For each 2-clause $(x \vee y)$ in S_A, select one clause from F that $(x \vee y)$ arose from: such a clause is either of the form $(\ell \vee x \vee y)$ where ℓ is a literal whose variable appears in S, or it is simply $(x \vee y)$ (F may contain 2-clauses itself). Put each such clause from F into a new set S'_A, so that $|S'_A| = |S_A|$. Suppose there are at least three 2-clauses in S'_A. Since these 2-clauses are disjoint and appear in F, the subformula of S'_A restricted to these 2-clauses is a subformula of F and must have at most a $(3/4)^3 < 0.422 < 1/2$ fraction of satisfying assignments. By Proposition 2.4, F has less than $0.422 \cdot 2^n$ satisfying assignments in this case. Otherwise, there are at most two 2-clauses in S'_A. Removing them from S'_A, there are still at least $48|S|$ 3-clauses. As there are $3|S|$ distinct variables appearing in S, and hence $6|S|$ literals whose variable appears in S, there must be a literal ℓ whose variable appears in S such that ℓ appears in at least 8 clauses of S'_A. By property $(\star)$ above, it follows that there is a subformula in F satisfying Lemma 4.4. Therefore F has less than $\rho 2^n$ satisfying assignments for a constant $\rho < 1/2$.
2) Otherwise, for all assignments A, we have $|S_A| < 48|S|$. In this case, we can try all $3^{|S_A|}$ satisfying assignments A' to the 2-clauses in S_A. Since S_A is a maximal disjoint set of 2-clauses in F_A, every formula obtained by plugging in A' is a 1-CNF formula. We solve #SAT on the resulting 1-CNF formula in linear time, and add the number to a running sum (calculated over all choices A and A').

Finally, output **YES** if the total sum of satisfying assignments exceeds 2^{n-1}, otherwise output **NO**. This completes the description of the algorithm, and its analysis.

It is interesting to observe that, no matter what 3-CNF formula is provided, at least one of the following conditions is true at the end of the algorithm:

(a) There are at least 2^{n-1} satisfying assignments (an early **YES** case).
(b) There are at most $\rho 2^n$ satisfying assignments, for a constant $\rho < 1/2$ (an early **NO** case).
(c) The number of satisfying assignments is counted exactly.

Therefore, for any 3-CNF formula in which the #SAT value is strictly between $\rho 2^n$ and 2^{n-1}, the above algorithm actually computes the #SAT value exactly.

Note that the above algorithm runs in linear time, although the constant factor in the worst case (enumeration over partial assignments) is at least $7^5 \cdot 3^{48 \cdot 5-1} > 10^{118}$. Of course, in order to give a succinct proof, we have been extremely loose with the analysis; a smaller constant factor is certainly possible.

V. Conclusion

There are many interesting open issues left to pursue; here are a few.

- **Determine the complexity of MAJ-MAJ-kSAT for $k \geq 3$.** For any fixed integer $k \geq 3$, is the MAJ-MAJ-kSAT problem PP-complete, in PP, or somewhere in between? We conjecture the problem is in P for all constant $k \geq 3$, but have not yet extended our methods to prove this result.
- **Parameter Dependence.** Although our algorithms for THR$_\rho$-kSAT run in linear time for fixed k and ρ, these runtimes grow *extremely* quickly as a function of ρ, even for $k = 3$ (as noted in the full version [26, Proposition 4.10]). Is a better dependence on ρ possible, or can we prove that a significantly better dependence is unlikely to exist? Could there be a poly$(1/\rho)$ dependence, as in the MAJORITY-2SAT algorithm?
- **Threshold Counting Beyond Satisfiability.** Are there other natural problems where the counting problem is known to be hard, but the threshold counting problem turns out to admit a polynomial time algorithm? Our results show this phenomenon holds for the counting and threshold counting versions of kSAT for constant k, but perhaps similar behavior occurs for other problems, such as counting perfect matchings or counting proper k-colorings of graphs.
- **Variants of Weighted Model Counting.** A natural "weighted" extension of the MAJORITY-kSAT problem would be: given $\rho \in (0,1)$ and m degree-k polynomials $p_1(x), \ldots, p_m(x) \in \mathbb{Q}[x_1, \ldots, x_n]$, determine if
$$\sum_{a \in \{0,1\}^n} \prod_{j=1}^{m} p_j(x) \geq \rho \cdot 2^n.$$
What does the complexity of this problem look like? In the special case solved in this paper (k-CNF), our polynomials have the form $1 - C_j$ where C_j is a product of k literals (x_i or $1 - x_i$).
To specialize further (and still fall within the k-CNF case), suppose each p_i takes values in $[0,1]$ over all $a \in \{0,1\}^n$, so their product $\prod_j p_j(a)$ is always in $[0,1]$. For constant $\rho \in (0,1)$, can the above sum-product problem be solvable in polynomial time?
- **Bayesian inference with k-CNFs.** Given two k-CNF formulas F and G over a common variable set, and

given $p \in (0,1)$, the inference problem is to determine whether

$$\Pr_x[F(x) = 1 \mid G(x) = 1] \geq p.$$

By definition, this is equivalent to determining whether

$$\frac{\Pr_x[(F(x) \wedge G(x)) = 1]}{\Pr_x[G(x) = 1]} \geq p.$$

Since determining if the denominator is nonzero is already NP-hard for $k = 3$, the best we can hope for is to put this problem in NP. To sidestep the division-by-zero issue, we can rephrase the inference problem as determining whether

$$\Pr_x[(F(x) \wedge G(x)) = 1] \geq p \cdot \Pr_x[G(x) = 1].$$

Already this problem is interesting for the case where F and G are 2-CNF.

Algorithms: The results of this paper imply that the inference problem is in polynomial time when F is 3-CNF and G is a 1-CNF. When $\#\text{SAT}(F \wedge G) \geq 2^n/\text{poly}(n)$, Theorem 3.1 implies that the inference problem is in P for 2-CNFs regardless of p (because both sides of the inequality can be counted exactly). Also, if $\#\text{SAT}(G) \geq 2^n/\text{poly}(n)$ and $p \geq 1/\text{poly}(n)$, then we can solve the inference problem for 2-CNFs using Theorem 3.1.

Hardness: If G is an arbitrary 3-CNF, and F is a 1-CNF, then the inference problem is already NP-hard. Deciding

$$\Pr_x[(F(x) \wedge G(x)) = 1] \geq p \cdot \Pr_x[G(x) = 1]$$

lets us construct a satisfying assignment for G: try both $F = x_1$ and $F = \neg x_1$ with $p = 1/2$. Observe that

$$\Pr_x[G(x) = 1] = \Pr_x[(x_1 \wedge G(x)) = 1] + \Pr_x[(\neg x_1 \wedge G(x)) = 1],$$

so either $\Pr_x[(x_1 \wedge G(x)) = 1] \geq 1/2 \cdot \Pr_x[G(x) = 1]$ or $\Pr_x[(\neg x_1 \wedge G(x)) = 1] \geq 1/2 \cdot \Pr_x[G(x) = 1]$. By choosing the larger of the two, we can construct a satisfying assignment for G for each variable one at a time.

The above discussion still does not yet settle the case where F is a 2-CNF and G is a 2-CNF, and the fractions involved are smaller than $1/\text{poly}(n)$.

Acknowledgment

The authors were supported by NSF CCF-1909429 and NSF CCF-1741615. Much of this work was performed while the second author was visiting the Simons Institute for the Theory of Computing, participating in the *Theoretical Foundations of Computer Systems* and *Satisfiability: Theory, Practice, and Beyond* programs.

References

[1] L. G. Valiant, "The complexity of enumeration and reliability problems," *SIAM Journal on Computing*, vol. 8, no. 3, pp. 410–421, 1979. [Online]. Available: https://doi.org/10.1137/0208032

[2] J. T. Gill, "Computational complexity of probabilistic Turing machines," in *Proceedings of the Sixth Annual ACM Symposium on Theory of Computing*, ser. STOC '74. New York, NY, USA: Association for Computing Machinery, 1974, p. 91–95. [Online]. Available: https://doi.org/10.1145/800119.803889

[3] J. Simon, "On Some Central Problems in Computational Complexity," Ph.D. dissertation, Cornell University, Jan. 1975. [Online]. Available: https://ecommons.cornell.edu/handle/1813/6975

[4] L. J. Stockmeyer, "The polynomial-time hierarchy," *Theoretical Computer Science*, vol. 3, no. 1, pp. 1–22, 1976. [Online]. Available: https://www.sciencedirect.com/science/article/pii/030439757690061X

[5] L. J. Stockmeyer and A. R. Meyer, "Word problems requiring exponential time(preliminary report)," in *Proceedings of the Fifth Annual ACM Symposium on Theory of Computing*, ser. STOC '73. New York, NY, USA: Association for Computing Machinery, 1973, p. 1–9. [Online]. Available: https://doi.org/10.1145/800125.804029

[6] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[7] P. Dagum and M. Luby, "Approximating the permanent of graphs with large factors," *Theoretical Computer Science*, vol. 102, no. 2, pp. 283–305, 1992. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0304397592902347

[8] M. Mundhenk, "The complexity of optimal small policies," *Mathematics of Operations Research*, vol. 25, no. 1, pp. 118–129, 2000. [Online]. Available: https://doi.org/10.1287/moor.25.1.118.15214

[9] ——, "The complexity of planning with partially-observable markov decision processes," Dartmouth College, USA, Tech. Rep., 2000.

[10] D. D. Bailey, V. Dalmau, and P. G. Kolaitis, "Phase transitions of PP-complete satisfiability problems," in *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001, Seattle, Washington, USA, August 4-10, 2001*. Morgan Kaufmann, 2001, pp. 183–192.

[11] A. Krause and C. Guestrin, "Optimal nonmyopic value of information in graphical models - efficient algorithms and theoretical limits," in *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30 - August 5, 2005*. Professional Book Center, 2005, pp. 1339–1345. [Online]. Available: http://ijcai.org/Proceedings/05/Papers/1154.pdf

[12] D. D. Bailey, V. Dalmau, and P. G. Kolaitis, "Phase transitions of PP-complete satisfiability problems," *Discret. Appl. Math.*, vol. 155, no. 12, pp. 1627–1639, 2007. [Online]. Available: https://doi.org/10.1016/j.dam.2006.09.014

[13] J. Goldsmith, M. Hagen, and M. Mundhenk, "Complexity of dnf minimization and isomorphism testing for monotone formulas," *Information and Computation*, vol. 206, no. 6, pp. 760–775, 2008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0890540108000138

[14] A. Krause and C. Guestrin, "Optimal value of information in graphical models," *Journal of Artificial Intelligence Research*, vol. 35, pp. 557–591, Jul. 2009. [Online]. Available: https://doi.org/10.1613/jair.2737

[15] T. Teige and M. Fränzle, "Resolution for stochastic boolean satisfiability," in *Logic for Programming, Artificial Intelligence, and Reasoning*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 625–639.

[16] A. E. Porreca, A. Leporati, G. Mauri, and C. Zandron, "P systems with elementary active membranes: Beyond np and conp," in *Membrane Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 338–347.

[17] J. Kwisthout, "Most probable explanations in Bayesian networks: Complexity and tractability," *International Journal of Approximate Reasoning*, vol. 52, no. 9, pp. 1452–1469, 2011, handling Incomplete and Fuzzy Information in Data Analysis and Decision Processes. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0888613X11001095

[18] P. Frisco, G. Govan, and A. Leporati, "Asynchronous p systems with active membranes," *Theoretical Computer Science*, vol. 429, pp. 74–86, 2012, magic in Science. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0304397511009923

[19] J. Kwisthout and C. P. de Campos, "Computional complexity of Bayesian networks," July 2015, tutorials of the 31st Conference on Uncertainty in Artificial Intelligence. [Online]. Available: https://www.youtube.com/watch?v=7CU5uo2XwIc

[20] ——, "Lecture notes: Computational complexity of Bayesian networks," July 2015, tutorials of the 31st Conference on Uncertainty in Artificial Intelligence. [Online]. Available: https://auai.org/uai2015/proceedings/slides/UAI2015_Comp_LN.pdf

[21] D. D. Mauá, C. P. De Campos, and F. G. Cozman, "The complexity of MAP inference in Bayesian networks specified through logical languages," in *Proceedings of the 24th International Conference on Artificial Intelligence*, ser. IJCAI'15. AAAI Press, 2015, p. 889–895.

[22] İ. İ. Ceylan, A. Darwiche, and G. V. den Broeck, "Open-world probabilistic databases," in *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR 2016, Cape Town, South Africa, April 25-29, 2016*. AAAI Press, 2016, pp. 339–348. [Online]. Available: http://www.aaai.org/ocs/index.php/KR/KR16/paper/view/12908

[23] F. G. Cozman and D. D. Mauá, "The complexity of Bayesian networks specified by propositional and relational languages," *Artificial Intelligence*, vol. 262, pp. 96–141, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0004370218303047

[24] F. Bridoux, N. Durbec, K. Perrot, and A. Richard, "Complexity of maximum fixed point problem in boolean networks," in *Computing with Foresight and Industry*. Cham: Springer International Publishing, 2019, pp. 132–143.

[25] F. Bridoux, A. Durbec, K. Perrot, and A. Richard, "Complexity of fixed point counting problems in boolean networks," *CoRR*, vol. abs/2012.02513, 2020. [Online]. Available: https://arxiv.org/abs/2012.02513

[26] S. Akmal and R. R. Williams, "MAJORITY-3SAT (and related problems) in polynomial time," *CoRR*, vol. abs/2107.02748, 2021. [Online]. Available: https://arxiv.org/abs/2107.02748

[27] R. Impagliazzo, W. Matthews, and R. Paturi, "A satisfiability algorithm for AC0," in *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*. SIAM, 2012, pp. 961–972. [Online]. Available: https://doi.org/10.1137/1.9781611973099.77

[28] T. M. Chan and R. R. Williams, "Deterministic APSP, Orthogonal Vectors, and more: Quickly derandomizing Razborov-Smolensky," *ACM Trans. Algorithms*, vol. 17, no. 1, pp. 2:1–2:14, 2021. [Online]. Available: https://doi.org/10.1145/3402926

[29] L. G. Valiant, "Accidental algorthims," in *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, 2006, pp. 509–517.

[30] S. Toda, "PP is as hard as the polynomial-time hierarchy," *SIAM J. Comput.*, vol. 20, no. 5, pp. 865–877, 1991. [Online]. Available: https://doi.org/10.1137/0220053

[31] M. L. Littman, J. Goldsmith, and M. Mundhenk, "The computational complexity of probabilistic planning," *Journal of Artificial Intelligence Research*, vol. 9, pp. 1–36, Aug. 1998. [Online]. Available: https://doi.org/10.1613/jair.505

[32] J. D. Park and A. Darwiche, "Complexity results and approximation strategies for MAP explanations," *J. Artif. Intell. Res.*, vol. 21, pp. 101–133, 2004. [Online]. Available: https://doi.org/10.1613/jair.1236

[33] A. Darwiche, *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press, 2009. [Online]. Available: http://www.cambridge.org/uk/catalogue/catalogue.asp?isbn=9780521884389

[34] M. Gaboardi, K. Nissim, and D. Purser, "The complexity of verifying loop-free programs as differentially private," in *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, ser. LIPIcs, A. Czumaj, A. Dawar, and E. Merelli, Eds., vol. 168. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, pp. 129:1–129:17. [Online]. Available: https://doi.org/10.4230/LIPIcs.ICALP.2020.129

[35] A. Choi, Y. Xue, and A. Darwiche, "Same-decision probability: A confidence measure for threshold-based decisions," *International Journal of Approximate Reasoning*, vol. 53, no. 9, pp. 1415–1428, 2012.

[36] U. Oztok, A. Choi, and A. Darwiche, "Solving pp^{pp}-complete problems using knowledge compilation," in *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR 2016, Cape Town, South Africa, April 25-29, 2016*, C. Baral, J. P. Delgrande, and F. Wolter, Eds. AAAI Press, 2016, pp. 94–103. [Online]. Available: http://www.aaai.org/ocs/index.php/KR/KR16/paper/view/12910

[37] K. W. Wagner, "The complexity of combinatorial problems with succinct input representation," *Acta Informatica*, vol. 23, no. 3, pp. 325–356, 1986. [Online]. Available: https://doi.org/10.1007/BF00289117

[38] J. Torán, "Complexity classes defined by counting quantifiers," *J. ACM*, vol. 38, no. 3, pp. 753–774, 1991. [Online]. Available: https://doi.org/10.1145/116825.116858

[39] E. Allender, M. Koucký, D. Ronneburger, S. Roy, and V. Vinay, "Time-space tradeoffs in the counting hierarchy," in *Proceedings of the 16th Annual IEEE Conference on Computational Complexity, Chicago, Illinois, USA, June 18-21, 2001*. IEEE Computer Society, 2001, pp. 295–302. [Online]. Available: https://doi.org/10.1109/CCC.2001.933896

[40] S. Arora and B. Barak, *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009. [Online]. Available: http://www.cambridge.org/catalogue/catalogue.asp?isbn=9780521424264

[41] M. Luby and B. Veličković, "On deterministic approximation of DNF," *Algorithmica*, vol. 16, no. 4-5, pp. 415–433, Oct. 1996. [Online]. Available: https://doi.org/10.1007/bf01940873

[42] E. Hirsch, "A fast deterministic algorithm for formulas that have many satisfying assignments," *Logic Journal of the IGPL*, vol. 6, no. 1, pp. 59–71, 01 1998. [Online]. Available: https://doi.org/10.1093/jigpal/6.1.59

[43] L. Trevisan, "A note on approximate counting for k-DNF," in *Proceedings of the 7th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2004, and 8th International Workshop on Randomization and Computation, RANDOM 2004*, ser. Lecture Notes in Computer Science, vol. 3122. Springer, 2004, pp. 417–426. [Online]. Available: https://doi.org/10.1007/978-3-540-27821-4_37

[44] P. Gopalan, R. Meka, and O. Reingold, "DNF sparsification and a faster deterministic counting algorithm," *computational complexity*, vol. 22, no. 2, pp. 275–310, May 2013. [Online]. Available: https://doi.org/10.1007/s00037-013-0068-6

[45] S. Lovett and J. Zhang, "Dnf sparsification beyond sunflowers," in *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, ser. STOC 2019. New York, NY, USA: Association for Computing Machinery, 2019, p. 454–460. [Online]. Available: https://doi.org/10.1145/3313276.3316323

[46] S. Lovett, N. Solomon, and J. Zhang, "From dnf compression to sunflower theorems via regularity," in *Proceedings of the 34th Computational Complexity Conference*, ser. CCC '19. Dagstuhl, DEU: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019. [Online]. Available: https://doi.org/10.4230/LIPIcs.CCC.2019.5

[47] L. J. Stockmeyer, "On approximation algorithms for #p," *SIAM J. Comput.*, vol. 14, no. 4, pp. 849–861, 1985. [Online]. Available: https://doi.org/10.1137/0214060

[48] H. Dell and J. Lapinskas, "Fine-grained reductions from approximate counting to decision," in *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*. ACM, 2018, pp. 281–288.

[49] R. Williams, "Defying hardness with a hybrid approach," Carnegie Mellon University, CMU-CS-04-159, School of Computer Science, Tech. Rep., 2004.