Distribution-Aware Sampling and Weighted Model Counting for SAT

Supratik Chakraborty  
Indian Institute of Technology, Bombay

Kuldeep S. Meel  
Department of Computer Science, Rice University

Moshe Y. Vardi  
Department of Computer Science, Rice University

Daniel J. Fremont  
University of California, Berkeley

Sanjit A. Seshia  
University of California, Berkeley

Abstract

Given a CNF formula and a weight for each assignment of values to variables, two natural problems are weighted model counting and distribution-aware sampling of satisfying assignments. Both problems have a wide variety of important applications. Due to the inherent complexity of the exact versions of the problems, interest has focused on solving them approximately. Prior work in this area scaled only to small problems in practice, or failed to provide strong theoretical guarantees, or employed a computationally-expensive maximum a posteriori probability (MAP) oracle that assumes prior knowledge of a factored representation of the weight distribution. We present a novel approach that works with a black-box oracle for weights of assignments and requires only an NP-oracle (in practice, a SAT-solver) to solve both the counting and sampling problems. Our approach works under mild assumptions on the distribution of weights of satisfying assignments, provides strong theoretical guarantees, and scales to problems involving several thousand variables. We also show that the assumptions can be significantly relaxed while improving computational efficiency if a factored representation of the weights is known.

1 Introduction

Given a set of weighted elements, computing the cumulative weight of all elements that satisfy a set of constraints is a fundamental problem that arises in many contexts. Known variously as weighted model counting, discrete integration and partition function computation, this problem has applications in machine learning, probabilistic reasoning, statistics, planning and combinatorics, among other areas. Unfortunately, the exact versions of both problems are computationally hard. Weighted model counting can be used to count the number of satisfying assignments of a CNF formula; hence it is \#P-hard. It is also known that an efficient algorithm for weighted sampling would yield a fully polynomial randomized approximation scheme (FPRAS) for \#P-complete inference problems – a possibility that lacks any evidence so far. Fortunately, approximate solutions to weighted model counting and weighted sampling are good enough for most applications. Consequently, there has been significant interest in designing practical approximate algorithms for these problems.

Since constraints arising from a large class of real-world problems can be modeled as propositional CNF (henceforth CNF) formulas, we focus on CNF and assume that the weights of truth assignments are given by a weight function \( w(\cdot) \) defined on the set of truth assignments. Roth showed that approximately counting the models of a CNF formula is \#P-hard even when the structure of the formula is severely restricted. By a result of Jerrum, Valiant and Vazirani, we also know that approximate model counting and almost uniform sampling (a special case of approximate weighted sampling) are polynomially inter-reducible. Therefore, it is unlikely that there exist polynomial-time algorithms for either approximate weighted model counting or approximate weighted sampling. Recently, a new class of algorithms that use pairwise independent random parity constraints and a MAP (maximum a posteriori probability)-oracle have been proposed for solving both problems. These algorithms provide strong theoretical guarantees (FPRAS relative to the MAP oracle), and have been shown to scale to medium-sized problems in practice. While this represents a significant step in our quest for practically efficient algorithms with strong guarantees for approximate weighted model counting and approximate weighted sampling, the use of MAP-queries presents issues that need to be addressed in practice. First, the use of MAP-queries along with parity constraints poses scalability hurdles. Second, existing MAP-query solvers work best when the distribution
of weights is represented by a graphical model with small tree-width—a restriction that is violated in several real-life problems. While this does not pose problems in practical applications where an approximation of the optimal MAP solution without guarantees of the approximation factor suffices, it presents significant challenges when we demand the optimal MAP solution. This motivates us to ask if we can design approximate algorithms for weighted model counting and weighted sampling that do not invoke MAP-oracles at all, and do not assume any specific representation of the weight distribution.

Our primary contribution is an affirmative answer to the above question under mild assumptions on the distribution of weights. Specifically, we show that two recently-proposed algorithms for approximate (unweighted) model counting (Chakraborty, Meel, and Vardi 2013a) and near-uniform (unweighted) sampling (Chakraborty, Meel, and Vardi 2013b) can be adapted to work in the setting of weighted assignments, using only a SAT solver (NP-oracle) and a black-box weight function \( w(\cdot) \). For the algorithm to work well in practice, we require that \( \text{tilt} \) of the weight function, which is the ratio of the maximum weight of a satisfying assignment to the minimum weight of a satisfying assignment, is small. We present arguments why this is a reasonable assumption in some important classes of problems. We also present an adaptation of our algorithm for problem instances where the tilt is large. The adapted algorithm requires a pseudo-Boolean SAT solver instead of a (regular) SAT solver as an oracle.

2 Notation and Preliminaries

Let \( F \) be a Boolean formula in conjunctive normal form (CNF), and let \( X \) be the set of variables appearing in \( F \). The set \( X \) is called the \textit{support} of \( F \). Given a set of variables \( S \subseteq X \) and an assignment \( \sigma \) of truth values to the variables in \( S \), we write \( \sigma|_S \) for the projection of \( \sigma \) onto \( S \). A \textit{satisfying assignment} or \textit{witness} of \( F \) is an assignment that makes \( F \) evaluate to true. We denote the set of all witnesses of \( F \) by \( R_F \). For notational convenience, whenever the formula \( F \) is clear from the context, we omit mentioning it. Let \( D \subseteq X \) be a subset of the support such that there are no two satisfying assignments that differ only in the truth values of variables in \( D \). In other words, in every satisfying assignment, the truth values of variables in \( X \setminus D \) uniquely determine the truth value of every variable in \( D \). The set \( D \) is called a \textit{dependent} support of \( F \), and \( X \setminus D \) is called an \textit{independent} support. Note that there may be more than one independent support: \((a \lor \neg b) \land (\neg a \lor b)\) has three, namely \( \{a\} \), \( \{b\} \) and \( \{a, b\} \). Clearly, if \( I \) is an independent support of \( F \), so is every superset of \( I \).

Let \( w(\cdot) \) be a function that takes as input an assignment \( \sigma \) and yields a real number \( w(\sigma) \in (0, 1) \) called the \textit{weight} of \( \sigma \). Given a set \( Y \) of assignments, we use \( w(Y) \) to denote \( \Sigma_{\sigma \in Y} w(\sigma) \). Our main algorithms (see Section 3) make no assumptions about the nature of the weight function, treating it as a black-box function. In particular, we do not assume that the weight of an assignment can be factored into the weights of projections of the assignment on specific subsets of variables. The exception to this is Section 5, where we consider possible improvements when the weights are given by a known function, or “white-box”.

Three important quantities derived from the weight function are \( w_{\max} = \max_{\sigma \in R_F} w(\sigma), w_{\min} = \min_{\sigma \in R_F} w(\sigma), \) and the \textit{tilt} \( \rho = w_{\max}/w_{\min} \). Our algorithms require an upper bound on the tilt, denoted \( r \), which is provided by the user. As tight a bound as possible is desirable to maximize the efficiency of the algorithms. We define MAP (\textit{maximum a posteriori probability}) for our distribution of weights to be \( w_{\max}/w(R_F) \).

We write \( Pr[X : P] \) for the probability of outcome \( X \) when sampling from a probability space \( P \). For brevity, we omit \( P \) when it is clear from the context. The expected value of the outcome \( X \) is denoted \( E[X] \).

A special class of hash functions, called \textit{k-wise independent} hash functions, play a crucial role in our work (Bellare, Goldreich, and Petrank 1998). Let \( n, m, k \) be positive integers, and let \( H(n, m, k) \) denote a family of \( k \)-wise independent hash functions mapping \( \{0, 1\}^n \) to \( \{0, 1\}^m \). We use \( h \in H(n, m, k) \) to denote the probability space obtained by choosing a hash function \( h \) uniformly at random from \( H(n, m, k) \). The property of \( k \)-wise independence guarantees that for all \( \alpha_1, \ldots, \alpha_k \in \{0, 1\}^m \) and for all distinct \( y_1, \ldots, y_k \in \{0, 1\}^n \), \( Pr[h(y_i) = \alpha_i : h \in H(n, m, k)] = 2^{-mk} \). For every \( \alpha \in \{0, 1\}^m \) and \( h \in H(n, m, k) \), let \( h^{-1}(\alpha) \) denote the set \( \{y \in \{0, 1\}^n \mid h(y) = \alpha\} \). Given \( R_F \subseteq \{0, 1\}^n \) and \( h \in H(n, m, k) \), we use \( R_{F, h, \alpha} \) to denote the set \( R_F \cap h^{-1}(\alpha) \).

Our work uses an efficient family of hash functions, denoted as \( H_{swr}(n, m, 3) \). Let \( h : \{0, 1\}^n \rightarrow \{0, 1\}^m \) be a hash function in the family, and let \( y \) be a vector in \( \{0, 1\}^n \). Let \( h(y)[i] \) denote the \( i \)-th component of the vector obtained by applying \( h \) to \( y \). The family of hash functions of interest is defined as \( \{h(y) \mid h(y)[i] = a_{i,0} \oplus (\bigoplus_{i=1}^n a_{i,j} \cdot y[i])\} \), \( a_{i,j} \in \{0, 1\}, 1 \leq i \leq m, 0 \leq j \leq n \), where \( \oplus \) denotes the xor operation. By choosing values of \( a_{i,j} \) randomly and independently, we can effectively choose a random hash function from the family. It has been shown in (Gomes, Sabharwal, and Selman 2007) that this family of hash functions is \( 3 \)-independent.

Given a CNF formula \( F \), an \textit{exact weighted model counter} returns \( w(R_F) \). An \textit{approximate weighted model counter} relaxes this requirement to some extent: given tolerance \( \epsilon > 0 \) and confidence \( 1 - \delta \in (0, 1] \), the value \( v \) returned by the counter satisfies \( Pr[w(R_F) \in (1 + \epsilon)w(R_F)] \geq 1 - \delta \). A related type of algorithm is a \textit{weighted-uniform probabilistic generator}, which outputs a witness \( w \in R_F \) such that \( Pr[w = y] = w(y)/w(R_F) \) for every \( y \in R_F \). An \textit{almost weighted-uniform generator} relaxes this requirement, ensuring that for all \( y \in R_F \), we have \( w(y)/(1 + \epsilon)w(R_F) \).
compute approximate weighted model counts and also expected sizes, Ermon et al showed that they can provably achieve approximate weighted sampling. The performance of Ermon et al’s algorithms depend crucially on the ability to efficiently answer MAP queries. Complexity-wise, MAP is significantly harder than CNF satisfiability, and is known to be NP$^{PP}$-complete (Park 2002). The problem is further compounded by the fact that the MAP queries generated by Ermon et al’s algorithms have random parity constraints built into them. Existing MAP-solving techniques work efficiently when the weight distribution of assignments is specified by a graphical model, and the underlying graph has specific structural properties. With random parity constraints, these structural properties are likely to be violated very often. In (Ermon et al. 2013c), it has been argued that a MAP-oracle-based weighted model-counting algorithm proposed in (Ermon et al. 2013a) is unlikely to scale well to large problem instances. Since MAP solving is also crucial in the weighted sampling algorithm of (Ermon et al. 2013b), the same criticism applies to that algorithm as well. Several relaxations of the MAP-oracle-based algorithm proposed in (Ermon et al. 2013a), were therefore discussed in (Ermon et al. 2013c). While these relaxations help reduce the burden of MAP solving, they also significantly weaken the theoretical guarantees.

In later work (Ermon et al. 2014), Ermon et al showed how the average size of parity constraints in their weighted model counting and weighted sampling algorithms can be reduced using a new class of hash functions. This work, however, still stays within the same paradigm as their earlier work – i.e, it uses MAP-oracles and XOR constraints. Although Ermon et al’s algorithms provide a 16-factor approximation in theory, in actual experiments, they use relaxations and timeouts of the MAP solver to get upper and lower bounds of the optimal MAP solution. Unfortunately, these bounds do not come with any guarantees on the factor of approximation. Running the MAP solver to obtain the optimal value is likely to take significantly longer, and is not attempted in Ermon et al’s work.

The algorithms developed in this paper are closely related to two algorithms proposed recently by Chakraborty, Meel and Vardi (Chakraborty, Meel, and Vardi 2013a) (Chakraborty, Meel, and Vardi 2013b). The first of these (Chakraborty, Meel, and Vardi 2013a) computes the approximate (unweighted) model-count of a CNF formula, while the second algorithm (Chakraborty, Meel, and Vardi 2013b) performs near-uniform (unweighted) sampling. Like Ermon et al’s algorithms, these algorithms make use of parity constraints as pair-wise independent hash functions, and can benefit from the new class of hash functions proposed in (Ermon et al. 2014). Unlike, however, Ermon et al’s algorithms, Chakraborty et al. use a SAT solver (NP-oracle) specifically engineered to handle parity constraints efficiently.

### 4 Algorithm

We now present algorithms for approximate weighted model counting and approximate weighted sampling, assuming a small bounded tilt and a black-box weight function.
Recalling that the tilt concerns weights of only satisfying assignments, our assumption about it being bounded by a small number is reasonable in several practical situations. For example, when solving probabilistic inference with evidence by reduction to weighted model counting (Chavira and Darwiche 2008), every satisfying assignment of the CNF formula corresponds to an assignment of values to variables in the underlying probabilistic graphical model that is consistent with the evidence. Furthermore, the weight of a satisfying assignment is the joint probability of the corresponding assignment of variables in the probabilistic graphical model. A large tilt would therefore mean existence of two assignments that are consistent with the evidence, but one of which is overwhelmingly more likely than the other. In several real-world problems (see, e.g. Sec 8.3 of [Diaz and Druzdzel 2006], this is considered unlikely given that numerical conditional probability values are often obtained from human experts providing qualitative and rough quantitative data.

Our weighted model counting algorithm, called WeightMC, is best viewed as an adaptation of the ApproxMC algorithm proposed by Chakraborty, Meel and Vardi (Chakraborty, Meel, and Vardi 2013a) for approximate unweighted model counting. Similarly, our weighted sampling algorithm, called WeightGen, can be viewed as an adaptation of the the UniWit algorithm (Chakraborty, Meel, and Vardi 2013b), originally proposed for near-uniform unweighted sampling. The key idea in both ApproxMC and UniWit is to partition the set of satisfying assignments into “cells” containing roughly equal numbers of satisfying assignments, using a random hash function from the family $H_{xor}(n, m, 3)$. A random cell is then chosen and inspected to see if the number of satisfying assignments in it is smaller than a pre-computed threshold. The threshold, in turn, depends on the desired approximation factor or tolerance $\varepsilon$. If the chosen cell is small enough, UniGen samples uniformly from the chosen small cell to obtain a near-uniformly generated satisfying assignment. ApproxMC multiplies the number of satisfying assignments in the cell by a suitable scaling factor to obtain an estimate of the model count. ApproxMC is then repeated a number of times (depending on the desired confidence: $1 - \delta$) and the statistical median of computed counts taken to give the final approximate model count. For weighted model counting and sampling, the primary modification that needs to be done to ApproxMC and UniGen is that instead of requiring “cells” to have roughly equal numbers of satisfying assignments, we now require them to have roughly equal weights of satisfying assignments. To ensure that all weights lie in $[0, 1]$, we scale weights by a factor of $\frac{1}{w_{\text{max}}}$. Unlike earlier works (Ermon et al. 2013a; Ermon et al. 2013c), however, we do not require a MAP-oracle to get $w_{\text{max}}$; instead we estimate $w_{\text{max}}$ online without incurring any additional performance cost.

A randomly chosen hash function from $H_{xor}(n, m, 3)$ consists of $m$ XOR constraints, each of which has expected size $n/2$. Although ApproxMC and UniWit were shown to scale for few thousands of variables, the performance erodes rapidly after a few thousand variables. It has recently been shown in (Chakraborty, Meel, and Vardi 2014) that by using random parity constraints on the independent support of a formula (which can be orders of magnitude smaller than the complete support), we can significantly reduce the size of XOR constraints. We use this idea in our work. For all our benchmark problems, obtaining the independent support of CNF formulae has been easy, once we examine the domain from which the problem originated.

Both WeightMC and WeightGen assume access to a subroutine called BoundedWeightSAT that takes a CNF formula $F$, a “pivot”, an upper bound $r$ of the tilt and an upper bound $w_{\text{max}}$ of the maximum weight of a satisfying assignment in the independent support set $S$. It returns a set of satisfying assignments of $F$ such that the total weight of the returned assignments scaled by $1/w_{\text{max}}$ exceeds pivot. It also updates the minimum weight of a satisfying assignment seen so far and returns the same. BoundedWeightSAT accesses a subroutine AddBlockClause that takes as inputs a formula $F$ and a projected assignment $\sigma|_{S}$, computes a blocking clause for $\sigma|_{S}$, and returns the formula $F'$ obtained by conjoining $F$ with the blocking clause thus obtained. Both algorithms also accept as input a positive real-valued parameter $r$ which is an upper bound on $\rho$. Finally, the algorithms assume access to an NP-oracle, which in particular can decide SAT.

4.1 WeightMC Algorithm

The pseudocode for WeightMC is shown in Algorithm 1. The algorithm takes a CNF formula $F$, tolerance $\varepsilon \in (0, 1)$, confidence parameter $\delta \in (0, 1)$, independent support $S$, and tilt upper bound $r$, and returns an approximate weighted model count. WeightMC invokes an auxiliary procedure WeightMCCore that computes an approximate weighted model count by randomly partitioning the space of satisfying assignments using hash functions from the family $H_{xor}(|S|, m, 3)$, where $S$ denotes an independent support of $F$. After invoking WeightMCCore sufficiently many times, WeightMC returns the median of the non-$\perp$ counts returned by WeightMCCore.

**Theorem 1.** Given a propositional formula $F$, $\varepsilon \in (0, 1)$, $\delta \in (0, 1)$, independent support $S$, and tilt bound $r$, suppose WeightMC($F, \varepsilon, \delta, S, r$) returns $c$. Then $\Pr[(1 + \varepsilon)^{-1} \cdot w(R_F) \leq c \leq (1 + \varepsilon) \cdot w(R_F)] \geq 1 - \delta$.

**Theorem 2.** Given an oracle for SAT, WeightMC($F, \varepsilon, \delta, S, r$) runs in time polynomial in $\log_2(1/\delta), r, |F|$ and $1/\varepsilon$ relative to the oracle.

We defer all proofs to the supplementary material for lack of space.

4.2 WeightGen Algorithm

The pseudocode for WeightGen is presented in Algorithm 2. WeightGen takes in a CNF formula $F$, tolerance $\varepsilon > 1.71$, tilt upper bound $r$, and independent support $S$ and returns a random (approximately weighted-uniform) satisfying assignment. WeightGen first computes $\kappa$ and pivot and uses them to compute $\text{hiThresh}$ and $\text{loThresh}$, which quantify the size of a “small” cell. The easy case of the weighted
count being less than hiThresh is handled in lines 6–9. Otherwise, WeightMC is called to estimate the weighted model count, which is used to estimate the range of candidate values for \( m \). The choice of parameters for WeightMC is motivated by technical reasons. The loop in 13–19 terminates when a small cell is found and a sample is picked weighted-uniformly at random. Otherwise, the algorithm reports a failure.

**Theorem 3.** Given a CNF formula \( F \), tolerance \( \varepsilon > 1.71 \), tilt bound \( r \), and independent support \( S \), for every \( y \in R_F \) we have 
\[
\Pr[\text{WeightGen}(F, \varepsilon, r, X) = y] \leq (1 + \varepsilon) w(R_F) \leq (1 + \varepsilon) w(R_F).
\]
Also, WeightGen succeeds (i.e. does not return \( \perp \)) with probability at least 0.62.

**Theorem 4.** Given an oracle for SAT, WeightGen\((F, \varepsilon, r, S)\) runs in time polynomial in \(|F|\) and \(1/\varepsilon\) relative to the oracle.

### 4.3 Implementation Details

In our implementations of WeightGen and WeightMC, BoundedWeightSAT is implemented using CryptoMiniSAT [Cry], a SAT solver that handles xor clauses efficiently. CryptoMiniSAT uses blocking clauses to prevent already generated witnesses from being generated again. Since the independent support of \( F \) determines every satisfying assignment of \( F \), blocking clauses can be restricted to only variables in the set \( S \). We implemented this optimization in CryptoMiniSAT, leading to significant improvements in performance. We used “random device” implemented in C++11 as source of pseudo-random numbers to make random choices in WeightGen and WeightMC.

**Algorithm 1 WeightMC\((F, \varepsilon, \delta, S, r)\)**

1: counter ← 0; \( C \leftarrow \text{emptyList}; \) \( w_{\text{max}} \leftarrow 1; \)
2: pivot ← \( 2 \times \left( \varepsilon \right)^{2/2} \left( 1 + \frac{\delta}{3} \right); \)
3: \( t \leftarrow \left\lceil 35 \log_2(3/\delta) \right\rceil; \)
4: repeat
5: \( (c, w_{\text{max}}) \leftarrow \text{WeightMCCore}(F, S, \text{pivot}, r, w_{\text{max}}); \)
6: \( \text{counter} \leftarrow \text{counter} + 1; \)
7: if \( c \neq \perp \) then
8: \( \text{AddToList}(C, c \cdot w_{\text{max}}); \)
9: until \( \text{counter} < t \)
10: \( \text{finalCount} \leftarrow \text{FindMedian}(C); \)
11: return \( \text{finalCount}; \)

### 5 Experimental Results

To evaluate the performance of WeightGen and WeightMC, we built prototype implementations and conducted an extensive set of experiments. The suite of benchmarks was made up of problems arising from various practical domains as well as problems of theoretical interest. Specifically, we used bit-level unweighted versions of constraints arising from grid networks, plan recognition, DQMR networks, bounded model checking of circuits, bitblasted versions of SMT-LIB (SMT) benchmarks, and IS-CAS89 (Brglez, Bryan, and Kozminski 1989) circuits with parity conditions on randomly chosen subsets of outputs and next-state variables (Sang, Bearne, and Kautz 2005 John and Chakraborty 2011). While our algorithm is agnostic to the weight oracle, other tools that we used for comparison require the weight of an assignment to be the product of the weights of its literals. Consequently, to create weighted problems with tilt at most some bound \( r \), we randomly selected \( m = \max(15, n/100) \) of the variables and assigned them the weight \( w \) such that \((w/(1-w))^{m} = r\), their negation the weight \( 1-w \), and all other literals the weight 1. Unless mentioned otherwise, our experiments for WeightMC used \( r = 3 \), \( \epsilon = 0.8 \), and \( \delta = 0.2 \), while our experiments for WeightGen used \( r = 3 \) and \( \epsilon = 5 \).

To facilitate performing multiple experiments in parallel, we used a high performance cluster, each experiment running on its own core. Each node of the cluster had two quad-core Intel Xeon processors with 4GB of main memory. We used 2500 seconds as the timeout of each invocation of BoundedWeightSAT and 20 hours as the overall timeout for WeightGen and WeightMC. If an invocation of BoundedWeightSAT timed out in line 10 (WeightMC) and line 17 (WeightGen), we repeated the execution of the corresponding loops without incrementing the variable \( i \) (in both algorithms). With this setup, WeightMC and WeightGen were able to successfully return weighted counts and generate weighted random instances for formulas with close to 64,000 variables.

We compared the performance of WeightMC with the SDD Package [sdd], a state-of-the-art tool which
Algorithm 4 WeightGen\((F, \varepsilon, r, S)\)

\(\text{Assume } \varepsilon > 1.71 \forall i\)

1: \(w_{\text{max}} \leftarrow 1\); Samples \(\{\}\);
2: \((k, \text{pivot}) \leftarrow \text{ComputeKappaPivot}(\varepsilon)\);
3: hiThresh \leftarrow \(1 + (1 + \kappa)\) pivot;
4: loThresh \leftarrow \(1 + \kappa\) pivot;
5: \((Y, w_{\text{max}}) \leftarrow \text{BoundedWeightSAT}(F, \text{hiThresh}, r, w_{\text{max}}, S)\);
6: if \((w(Y) / w_{\text{max}}) \leq \text{hiThresh}) then
7: \(\text{Choose } y \text{ weighted-uniformly at random from } Y\);
8: \(\text{return } y\);
9: else
10: \((C, w_{\text{max}}) \leftarrow \text{WeightMC}(F, 0.8, 0.2)\);
11: \(q \leftarrow [\log C - \log w_{\text{max}} + \log 1.8 - \log \text{pivot}]\);
12: \(i \leftarrow q - 4\);
13: repeat
14: \(i \leftarrow i + 1\);
15: \(\text{Choose } h \text{ at random from } H_{\text{sort}}(|S|, i, 3)\);
16: \(\text{Choose } \alpha \text{ at random from } \{0, 1\}^*\);
17: \((Y, w_{\text{max}}) \leftarrow \text{BoundedWeightSAT}(F \land (h(x_1, \ldots, x_{|S|}) = \langle \alpha \rangle, \text{hiThresh}, r, w_{\text{max}}, S)\);
18: \(W \leftarrow w(Y) / w_{\text{max}}\);
19: until \((\text{loThresh} \leq W \leq \text{hiThresh}) \lor (i = g)\);
20: if \((W > \text{hiThresh}) \lor (W < \text{loThresh})\) then \(\text{return } y\);
21: \(\text{else } \text{Choose } y \text{ weighted-uniformly at random from } Y\); \(\text{return } y\);

Algorithm 5 ComputeKappaPivot\((\varepsilon)\)

1: \(\text{Find } \kappa \in \{0, 1\} \text{ such that } \varepsilon = (1 + \kappa)(2.36 + \frac{0.51}{(1 - \kappa)^2}) - 1\);
2: \(\text{pivot} \leftarrow [\varepsilon^{3/2} \left(1 + \frac{1}{\kappa}\right)^2]; \text{return } (\kappa, \text{pivot})\)

---

Table 1: WeightMC, SDD, and WeightGen runtimes in seconds.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>vars</th>
<th>#clus</th>
<th>WeightMC</th>
<th>SDD</th>
<th>WeightGen</th>
</tr>
</thead>
<tbody>
<tr>
<td>w50</td>
<td>100</td>
<td>266</td>
<td>15</td>
<td>0.38</td>
<td>0.14</td>
</tr>
<tr>
<td>w70</td>
<td>140</td>
<td>374</td>
<td>771</td>
<td>0.83</td>
<td>13.37</td>
</tr>
<tr>
<td>s526a2</td>
<td>365</td>
<td>944</td>
<td>62</td>
<td>29.54</td>
<td>0.85</td>
</tr>
<tr>
<td>s526a2t</td>
<td>366</td>
<td>944</td>
<td>81</td>
<td>12.16</td>
<td>1.1</td>
</tr>
<tr>
<td>s953a2</td>
<td>515</td>
<td>1297</td>
<td>11978</td>
<td>355.7</td>
<td>21.14</td>
</tr>
<tr>
<td>s1238a3t</td>
<td>704</td>
<td>1926</td>
<td>3519</td>
<td>19.52</td>
<td>mem</td>
</tr>
<tr>
<td>s1196a15t</td>
<td>777</td>
<td>2165</td>
<td>3087</td>
<td>2275</td>
<td>19.59</td>
</tr>
<tr>
<td>Sorting</td>
<td>1434</td>
<td>5028</td>
<td>34942</td>
<td>mem</td>
<td>110.37</td>
</tr>
<tr>
<td>Squaring7</td>
<td>1628</td>
<td>5837</td>
<td>39367</td>
<td>mem</td>
<td>113.12</td>
</tr>
<tr>
<td>ProcessBean</td>
<td>4768</td>
<td>14458</td>
<td>53746</td>
<td>mem</td>
<td>418.29</td>
</tr>
<tr>
<td>LoginService2</td>
<td>11511</td>
<td>41411</td>
<td>322</td>
<td>mem</td>
<td>3.45</td>
</tr>
<tr>
<td>Sort</td>
<td>12125</td>
<td>49613</td>
<td>19303</td>
<td>T</td>
<td>140.19</td>
</tr>
<tr>
<td>EnqueueSeq</td>
<td>16466</td>
<td>58515</td>
<td>8620</td>
<td>mem</td>
<td>165.64</td>
</tr>
<tr>
<td>Karatsuba</td>
<td>19594</td>
<td>82417</td>
<td>4962</td>
<td>mem</td>
<td>193.11</td>
</tr>
<tr>
<td>TreeMax</td>
<td>24859</td>
<td>103762</td>
<td>34</td>
<td>T</td>
<td>2.0</td>
</tr>
<tr>
<td>LLReverse</td>
<td>63797</td>
<td>257657</td>
<td>1496</td>
<td>mem</td>
<td>88.0</td>
</tr>
</tbody>
</table>

---

Figure 1: Quality of counts computed by WeightMC. The benchmarks are arranged in increasing order of weighted model counts.

---

Figure 2: Uniformity comparison for case110
To measure the accuracy of WeightGen, we implemented an Ideal Sampler, henceforth called IS, and compared the distributions generated by WeightGen and IS for a representative benchmark. Given a CNF formula $F$, IS first generates all the satisfying assignments, then computes their weights and uses these to sample the ideal distribution. We then generated a large number $N (= 6 \times 10^5)$ of sample witnesses using both IS and WeightGen. In each case, the number of times various witnesses were generated was recorded, yielding a distribution of the counts. Figure 2 shows the distributions generated by WeightGen and IS for one of our benchmarks (case 110) with 16,384 solutions. The almost perfect match between the distribution generated by IS and WeightGen held also for other benchmarks. Thus, as was the case for WeightMC, the accuracy of WeightGen is better in practice than that established by Theorem 3.

6 White-Box Weight Functions

As noted above, the runtime of WeightMC is proportional to the tilt of the weight function, which means that the algorithm becomes impractical when the tilt is large. If the assignment weights are given by a known polynomial-time-computable function instead of an oracle, we can do better. We abuse notation slightly and denote this weight function by $w(X)$, where $X$ is the set of support variables of the Boolean formula $F$. The essential idea is to partition the set of satisfying assignments into regions within which the tilt is small. Defining $R_F(a, b) = \{ \sigma \in R_F | a < w(\sigma) \leq b \}$, we have $w(R_F) = w(R_F(w_{min}, w_{max}))$.

If we use a partition of the form $R_F(w_{min}, w_{max}) = R_F(w_{max}/2, w_{max}) \cup R_F(w_{max}/4, w_{max}/2) \cup \cdots \cup R_F(w_{max}/2^{N}, w_{max}/2^{N-1})$, where $w_{max}/2^{N} \leq w_{min}$, then in each partition region the tilt is at most 2. Note that we do not need to know the actual values of $w_{min}$ and $w_{max}$: any bounds $L$ and $H$ such that $0 < L \leq w_{min}$ and $w_{max} \leq H$ will do (although if the bounds are too loose, we may partition $R_F$ into more regions than necessary). If assignment weights are poly-time computable, we can add to $F$ a constraint that eliminates all assignments not in a particular region. So we can run WeightMC on each region in turn, passing 2 as the upper bound on the tilt, and sum the results to get $w(R_F)$. This idea is implemented in PartitionedWeightMC (Algorithm 6).

The correctness and runtime of PartitionedWeightMC are established by the following theorems, whose proof is deferred to Appendix.

Algorithm 6 PartitionedWeightMC($F, \varepsilon, \delta, S, L, H$)

1: $N \leftarrow \lceil \log_2 H/L \rceil + 1; \delta' \leftarrow \delta/N; c \leftarrow 0$
2: for all $1 \leq m \leq N$ do
3: $G \leftarrow F \land (H/2^m < w(X) \leq H/2^{m-1})$
4: $d \leftarrow \text{WeightMC}(G, \varepsilon, \delta', S, 2)$
5: if $(d = \bot)$ then return $\bot$
6: $c \leftarrow c + d$
7: return $c$

Theorem 5. If PartitionedWeightMC($F, \varepsilon, \delta, S, L, H$) returns $c$ (and all arguments are in the required ranges), then

$$\Pr[c \neq \bot \land (1 + \varepsilon)^{-1}w(R_F) \leq c \leq (1 + \varepsilon)w(R_F)] \geq 1 - \delta.$$  

Theorem 6. With access to an NP oracle, the runtime of PartitionedWeightMC($F, \varepsilon, \delta, S, L, H$) is polynomial in $|F|$, $1/\varepsilon$, $\log(1/\delta)$, and $\log r = \log(H/L)$.

The reduction of the runtime’s dependence on the tilt bound $r$ from linear to logarithmic can be a substantial saving. If the assignment weights are products of literal weights, as is the case in many applications, the best $a$ priori bound on the tilt $\rho$ given only the literal weights is exponential in $n$. Thus, unless the structure of the problem allows a better bound on $\rho$ to be used, WeightMC will not be practical. In this situation PartitionedWeightMC can be used to maintain polynomial runtime.

When implementing PartitionedWeightMC in practice the handling of the weight constraint $H/2^m < w(X) \leq H/2^{m-1}$ is critical to efficiency. If assignment weights are sums of literal weights, or equivalently products of literal weights (we just take logarithms), then the weight constraint is a pseudo-Boolean constraint. In this case we may replace the SAT-solver used by WeightMC with a pseudo-Boolean satisfiability (PBS) solver. While a number of PBS-solvers exist, none have the specialized handling of XOR clauses that is critical in making WeightMC practical. The design of such solvers is a clear direction for future work. We also note that the choice of 2 as the tilt bound for each region is arbitrary, and the value may be adjusted depending on the application: larger values will decrease the number of regions, but increase the difficulty of counting within each region. Finally, note that the same partitioning idea can be used to reduce WeightGen’s dependence on $r$ to be logarithmic.

7 Conclusion

In this paper, we considered approximate approaches to the twin problems of distribution-aware sampling and weighted model counting for SAT. For approximation techniques that provide strong theoretical two-way bounds, a major limitation is the reliance on potentially-expensive maximum a posteriori (MAP) queries. We showed how to remove this reliance on MAP queries, while retaining strong theoretical guarantees. First, we provided model counting and sampling algorithms that work with a black-box model of giving weights to assignments, requiring access only to an NP-oracle, which is efficient for small tilt values. Experimental results demonstrate the effectiveness of this approach in practice. Second, we provide an alternative approach that
promises to be efficient for tilt value, requiring, however, a white-box model of weighting and access to a pseudo-Boolean solver. As a next step, we plan to empirically evaluate this latter approach using pseudo-Boolean solvers designed to handle parity constraints efficiently.
References


APPENDIX

Using notation introduced in Section 2, let \( w(y) \) denote the weight of solution \( y \) and \( R_F \) denote the set of witnesses of the Boolean formula \( F \). We denote the weight of the set \( R_F \) by \( w(R_F) \). For brevity, we write \( \forall V(y) \) for the expression \( w(y)/w_{\text{max}} \) (where \( w_{\text{max}} \) is the variable appearing in several of our algorithms).

Recall that WeightMC is a probabilistic algorithm that takes as inputs a Boolean CNF formula \( F \), a tolerance \( \varepsilon \), confidence parameter \( \delta \), a subset \( S \) of the support of \( F \), and an upper bound \( r \) on the ratio \( \rho \). We extend the results in (Chakraborty, Meel, and Vardi 2014) and show that if \( X \) is the support of \( F \), and if \( S \subseteq X \) is an independent support of \( F \), then WeightMC\( (F, \varepsilon, \delta, S, r) \) behaves identically (in a probabilistic sense) to WeightMC\( (F, \varepsilon, \delta, X, r) \). Once this is established, the remainder of the proof proceeds by making the simplifying assumption \( S = X \). The proofs of Lemmas 1 and 2 extend the earlier results by Chakraborty, Meel, and Vardi 2014 for unweighted sampling space.

Clearly, the above claim holds trivially if \( X = S \). Therefore, we focus only on the case when \( S \subsetneq X \). For notational convenience, we assume \( X = \{x_1, \ldots, x_n\} \), \( 0 \leq k < n \), \( S = \{x_1, \ldots, x_k\} \) and \( D = \{x_{k+1}, \ldots, x_n\} \) in all the statements and proofs in this section. We also use \( \bar{X} \) to denote the vector \((x_1, \ldots, x_n)\), and similarly for \( \bar{S} \) and \( \bar{D} \).

**Lemma 1.** Let \( F(\bar{X}) \) be a Boolean function, and \( S \) an independent support of \( F \). Then there exist Boolean functions \( g_0, g_1, \ldots, g_{n-k} \) each with support \( S \) such that

\[
F(\bar{X}) \leftrightarrow \left( g_0(\bar{S}) \land \bigwedge_{j=1}^{n-k} (x_{k+j} \leftrightarrow g_j(\bar{S})) \right)
\]

**Proof.** Since \( S \) is an independent support of \( F \), the set \( D = X \setminus S \) is a dependent support of \( F \). From the definition of a dependent support, there exist Boolean functions \( g_1, \ldots, g_k \) each with support \( S \), such that \( F(\bar{X}) \rightarrow \bigwedge_{j=1}^{n-k} (x_{k+j} \leftrightarrow g_j(\bar{S})) \).

Let \( g_0(\bar{S}) \) be the characteristic function of the projection of \( R_F \) on \( S \). More formally, \( g_0(\bar{S}) \equiv \bigvee_{(x_{k+1}, \ldots, x_n) \in \{0,1\}^{n-k}} F(\bar{X}) \). It follows that \( F(\bar{X}) \rightarrow g_0(\bar{S}) \).

Combining this with the result from the previous paragraph, we get the implication \( F(\bar{X}) \rightarrow g_0(\bar{S}) \land \bigwedge_{j=1}^{n-k} (x_{k+j} \leftrightarrow g_j(\bar{S})) \).

From the definition of \( g_0(\bar{S}) \) given above, we have \( g_0(\bar{S}) \rightarrow F(\bar{S}, x_{k+1}, \ldots, x_n) \), for some values of \( x_{k+1}, \ldots, x_n \). However, we also know that \( F(\bar{X}) \rightarrow \bigwedge_{j=1}^{n-k} (x_{k+j} \leftrightarrow g_j(\bar{S})) \). It follows that

\[
(g_0(\bar{S}) \land \bigwedge_{j=1}^{n-k} (x_{k+j} \leftrightarrow g_j(\bar{S}))) \rightarrow F(\bar{X}).
\]

Referring to the pseudocode of WeightMC in Section 3, we observe that the only steps that depend directly on \( S \) are those in line 3, where \( h \) is randomly chosen from \( H_{\text{xor}}(|S|, i, 3) \), and line 10, where the set \( Y \) is computed by calling \( \text{BoundedWeightSAT}(F \land (h(x_1, \ldots, x|S|) = \alpha), \text{pivot}, r, w_{\text{max}}) \). Since all subsequent steps of the algorithm depend only on \( Y \), it suffices to show that if \( S \) is an independent support of \( F \), the probability distribution of \( Y \) obtained at line 10 is identical to what we would obtain if \( S \) was set equal to the entire support \( X \).

The following lemma formalizes the above statement. As before, we assume \( X = \{x_1, \ldots, x_n\} \) and \( S = \{x_1, \ldots, x_k\} \).

**Lemma 2.** Let \( S \) be an independent support of \( F(\bar{X}) \). Let \( h \) and \( h' \) be hash functions chosen uniformly at random from \( H_{\text{xor}}(k, i, 3) \) and \( H_{\text{xor}}(n, i, 3) \), respectively. Let \( \alpha \) and \( \alpha' \) be tuples chosen uniformly at random from \( \{0,1\}^i \). Then, for every \( Y \in \{0,1\}^n \), \( \text{pivot} > 0 \), \( r \geq 1 \), and \( w_{\text{max}} \geq 1 \), we have

\[
\Pr \left[ \text{BoundedWeightSAT} \left( F(\bar{X}) \land (h(\bar{S}) = \alpha), \text{pivot}, r, w_{\text{max}} \right) = Y \right] = \Pr \left[ \text{BoundedWeightSAT} \left( F(\bar{X}) \land (h'(\bar{X}) = \alpha'), \text{pivot}, r, w_{\text{max}} \right) = Y \right]
\]

**Proof.** Since \( h' \) is chosen uniformly at random from \( H_{\text{xor}}(n, i, 3) \), recalling the definition of the latter we have \( F(\bar{X}) \land (h'(\bar{X}) = \alpha') \equiv F(\bar{X}) \land \bigwedge_{i=1}^{n} \left( (a_{i,0} \oplus \bigoplus_{j=1}^{n} a_{i,j} \cdot x[j]) \leftrightarrow \alpha'[l] \right) \), where the coefficients \( a_{i,j} \) are chosen i.i.d. uniformly from \( \{0,1\} \).

Since \( S \) is an independent support of \( F \), from Lemma 1 there exist Boolean functions \( g_1, \ldots, g_{n-k} \), each with support \( S \), such that \( F(\bar{X}) \rightarrow \bigwedge_{j=1}^{n-k} (x_{k+j} \leftrightarrow g_j(\bar{S})) \). Therefore, \( F(\bar{X}) \land (h'(\bar{X}) = \alpha') \) holds iff \( F(\bar{X}) \land \bigwedge_{j=1}^{n} \left( (a_{i,0} \oplus \bigoplus_{j=1}^{n} a_{i,j} \cdot x[j] + B) \leftrightarrow \alpha'[l] \right) \) does,

where \( B \equiv \bigoplus_{j=k+1}^{n} a_{i,j} \cdot g_{j-k}(\bar{S}) \). Rearranging terms, we get \( F(\bar{X}) \land \bigwedge_{l=1}^{i} \left( (a_{i,0} \oplus \bigoplus_{j=1}^{k} a_{i,j} \cdot x[j]) \leftrightarrow \alpha'[l] \oplus B \right) \).

Now since \( \alpha' \) is chosen uniformly at random from \( \{0,1\}^i \) and since \( B \) is independent of \( \alpha' \), we have that \( \alpha'[l] \oplus B \) is a random binary variable with equal probability of being \( 0 \) and \( 1 \). So \( \alpha'[l] \oplus B \) has the same distribution as \( \alpha[l] \), and the result follows.}

Lemma 2 allows us to continue with the remainder of the proof assuming \( S = X \). It has already been shown in (Gomes, Sabharwal, and Selman 2007) that
variables $\gamma, \eta$ of $h$

Therefore, using Lemma 3 with $\beta$, Lemma 4.

Lemma 5. Given $W(R_F) > pivot$, the probability that an invocation of WeightMCCore from WeightMC returns non-$\perp$ with $i \leq m$, is at least $1 - e^{-3/2}$.

Proof. Let $p_i (0 \leq i \leq n)$ denote the conditional probability that WeightMCCore terminates in iteration $i$ of the repeat-until loop (lines 6-11 of the pseudocode) with $0 < W(R_{F,h,\alpha}) \leq pivot$, given $W(R_F) > pivot$.

Since the choice of $h$ and $\alpha$ in each iteration of the loop are independent of those in previous iterations, the conditional probability that WeightMCCore returns non-$\perp$ with $i \leq m$, given $W(R_F) > pivot$, is $p_0 + (1-p_0)p_1 + \cdots + (1-p_0)^i(1-p_1)\cdots(1-p_{m-1})p_m$. Let us denote this sum by $P$. Thus, $P = p_0 + \prod_{i=1}^{m-1} (1-p_i)p_i \geq p_0 + \prod_{i=1}^{m-1} (1-p_i)p_i \geq \prod_{i=1}^{m-1} (1-p_i)p_i = p_m$.

The lemma is now proved by showing that $p_m \geq 1 - e^{-3/2}$.

It was shown in the proof of Lemma 4 that $Pr [(1+\varepsilon)^{-1} \cdot W(R_F) \leq 2W(R_{F,h,\alpha}) \leq (1+\varepsilon) \cdot W(R_F)] \geq 1 - e^{-3/2}$ for every $i \in \{0, \ldots, m\}$, $h \in H(n, i, 3)$ and $\alpha \in \{0, 1\}^3$. Substituting $m$ for $i$, re-arranging terms and noting that the definition of $m$ implies $2^{-m}W(R_F) = pivot/2$, we get $Pr [(1+\varepsilon)^{-1} \cdot W(R_F) \leq 2W(R_{F,h,\alpha}) \leq (1+\varepsilon) \cdot W(R_F)] \geq 1 - e^{-3/2}$. Since $0 < \varepsilon \leq 1$ and pivot $> 4$, it follows that $Pr [0 < W(R_{F,h,\alpha}) \leq pivot] \geq 1 - e^{-3/2}$. Hence, $p_m \geq 1 - e^{-3/2}$.

Lemma 6. Let an invocation of WeightMCCore from WeightMC return $c$. Then $Pr [c \neq \perp \land (1+\varepsilon)^{-1} \cdot w(R_F) \leq c \cdot w_{\text{max}} \leq (1+\varepsilon) \cdot w(R_F)] \geq (1 - e^{-3/2})^2 > 0.6$.

Proof. It is easy to see that the required probability is at least as large as $Pr [c \neq \perp \land i \leq m \land (1+\varepsilon)^{-1} \cdot w(R_F) \leq c \cdot w_{\text{max}} \leq (1+\varepsilon) \cdot w(R_F)]$. Dividing by $w_{\text{max}}$ and applying Lemmas 3 and 4 this probability is $\geq (1 - e^{-3/2})^2$.

We now turn to proving that the confidence can be raised to at least $1 - \delta$ for $\delta \in \{0, 1\}$ by invoking WeightMCCore $O(\log_2(1/\delta))$ times, and by using the median of the non-$\perp$ counts thus returned. For convenience of exposition, we use $\eta(t, m, p)$ in the following discussion to denote the probability of at least $m$ heads in $t$ independent tosses of a biased coin with $Pr[\text{heads}] = p$. Clearly, $\eta(t, m, p) = \sum_{k=m}^{\infty} \binom{t}{k} p^k (1-p)^{t-k}$.

Theorem 1. Given a propositional formula $F$ and parameters $\varepsilon (0 < \varepsilon \leq 1)$ and $\delta (0 < \delta \leq \frac{3}{2}$.
Throughout this proof, we assume that WeightMC is easy to verify that \( p \)-wise, we set it to
\[ \text{WeightMCCore} = \sum (1 + \varepsilon)^{-1} \cdot w(R_F) \leq c \leq (1 + \varepsilon) \cdot w(R_F) \geq 1 - \delta. \]

Proof. Throughout this proof, we assume that WeightMC is invoked \( t \) times from WeightMC, where \( t = \lceil 35 \log_2(3/\delta) \rceil \) (see pseudocode for ComputeMCCount in Section ??). Referring to the pseudocode of WeightMC, the final count returned is the median of the non-\( \perp \) counts obtained from the \( t \) invocations of WeightMCCore. Let \( \text{Err} \) denote the event that the median is not in \([ (1+\varepsilon)^{-1} \cdot W(R_F), (1+\varepsilon) \cdot W(R_F) \]). Let “\( \#\non \perp = q \)” denote the event that \( q \) (out of \( t \)) values returned by WeightMCCore are \( \non \perp \). Then, \( \Pr[\text{Err}] = \sum_{q=0}^{t \cdot \text{Pr}[\#\non \perp = q] \cdot \text{Pr}[\#\non \perp = q]. \)

In order to obtain \( \Pr[\text{Err} | \#\non \perp = q] \), we define a \( 0 - 1 \) random variable \( Z_i \), for \( 1 \leq i \leq t \), as follows. If the \( i \)-th invocation of WeightMCCore returns \( c \), and if \( c \) is either \( \perp \) or a non-\( \perp \) does not lie in the interval \( [(1+\varepsilon)^{-1} \cdot W(R_F), (1+\varepsilon) \cdot W(R_F)] \), we set \( Z_i = 1 \); otherwise, we set it to 0. From Lemma 6, \( \Pr[Z_i = 1] = p < 0.4 \).

If \( Z \) denotes \( \sum_{i=1}^{t} Z_i \), a necessary (but not sufficient) condition for event \( \text{Err} \) to occur, given that \( q \) \( \non \perp \)s were returned by WeightMCCore, is \( Z \geq (t - q + \lceil q/2 \rceil) \). To see why this is so, note that \( t - q \) invocations of WeightMCCore must return \( \perp \). In addition, at least \( \lceil q/2 \rceil \) of the remaining \( q \) invocations must return values outside the desired interval. To simplify the exposition, let \( q \) be an even integer. A more careful analysis removes this restriction and results in an additional constant scaling factor for \( \Pr[\text{Err}] \). With our simplifying assumption, \( \Pr[\text{Err} | \#\non \perp = q] \leq \Pr[Z \geq (t - q + \lceil q/2 \rceil)] = \Pr[Z \geq (t - q - 2) \geq 0.4 \cdot p \), which is a decreasing function of \( m \) and since \( q/2 \leq t - q/2 \leq t/2 \), we have \( \Pr[\text{Err} | \#\non \perp = q] \leq \Pr[Z \geq (t - q - 2) \geq 0.4 \cdot p \). In our case, \( p < 0.4 \); hence, \( \Pr[\text{Err} | \#\non \perp = q] \leq \eta(t, t/2, 0.4) \).

It follows from the above that \( \Pr[\text{Err}] = \sum_{q=0}^{t} \Pr[\text{Err} | \#\non \perp = q] \cdot \Pr[\#\non \perp = q] \leq \eta(t, t/2, 0.4) \cdot \sum_{q=0}^{t} \Pr[\#\non \perp = q] = \eta(t, t/2, 0.4) \cdot \sum_{k=0}^{t} \binom{t}{k} \cdot 0.4^k \cdot 0.6^{t-k} \leq \binom{t}{t/2} \sum_{k=0}^{t} \binom{t}{k} \cdot 0.4^k \cdot 0.6^{t-k} \leq 2^t \cdot 3 \cdot 0.6^{0.4} \cdot 0.6^{t} \leq 3 \cdot 0.98^t \). Since \( t = \lceil 35 \log_2(3/\delta) \rceil \), it follows that \( \Pr[\text{Err}] \leq \delta \).

**Theorem 2.** Given an oracle for \( \text{BoundedWeightSAT} \), \( \text{WeightMC}(F, \varepsilon, \delta, S, r) \) runs in time polynomial in \( \log_2(1/\delta), r, |F| \) and \( 1/\varepsilon \) relative to the oracle.

Proof. Referring to the pseudocode for WeightMC, lines 0 take \( O(1) \) time. The repeat-until loop in lines 0 is repeated \( t = \lceil 35 \log_2(3/\delta) \rceil \) times. The time taken for each iteration is dominated by the time taken by WeightMCCore. Finally, computing the median in line 0 takes time linear in \( t \). The proof is therefore completed by showing that WeightMCCore takes time polynomial in \( |F|, r \) and \( 1/\varepsilon \) relative to the SAT oracle.

Referring to the pseudocode for WeightMCCore, we find that BoundedWeightSAT is called \( O(|F|) \) times. Observe that when the loop in BoundedWeightSAT terminates, \( w_{\text{min}} \) is such that each \( y \in R_F \) whose weight was added to \( w_{\text{total}} \) has weight at least \( w_{\text{min}} \). Thus since the loop terminates when \( w_{\text{total}} \geq r \cdot \text{pivot} \), it has iterated at most \( (r \cdot \text{pivot} + 1) \) calls to the SAT oracle, and takes time polynomial in \( |F|, r \), and \( \text{pivot} \) relative to the oracle. Since \( \text{pivot} \) is in \( O(1/\varepsilon^2) \), the number of calls to the SAT oracle, and the total time taken by all calls to BoundedWeightSAT in each invocation of WeightMCCore is polynomial in \( |F|, r \) and \( 1/\varepsilon \) relative to the oracle. The random choices in lines 0 and 0 of WeightMC can be implemented in time polynomial in \( n \) (hence, in \( |F| \)) if we have access to a source of random bits. Constructing \( F \land \bigwedge(z_1, \ldots, z_n) = \alpha \) in line 0 can also be done in time polynomial in \( |F| \). \( \square \)

### B Analysis of WeightGen

For convenience of analysis, we assume that \( \log(W(R_F) - 1) - \log \text{pivot} \) is an integer, where \( \text{pivot} \) is the quantity computed by algorithm ComputeKappaPivot (see Section ??). A more careful analysis removes this assumption by scaling the probabilities by constant factors. Let us denote \( \log(W(R_F) - 1) - \log \text{pivot} \) by \( m \). The expression used for computing \( \text{pivot} \) in algorithm ComputeKappaPivot ensures that \( \text{pivot} \geq 17 \). Therefore, if an invocation of WeightGen does not return from line 0 of the pseudocode, then \( W(R_F) \geq 18 \). Note also that the expression for computing \( \kappa \) in algorithm ComputeKappaPivot requires \( \varepsilon \geq 1.71 \) in order to ensure that \( \kappa \in [0, 1) \) can always be found.

In the case where \( W(R_F) \leq 1 + (1 + \kappa)\text{pivot} \), BoundedWeightSAT returns all witnesses of \( F \) and WeightGen returns a perfect weighted-uniform sample on line 0. So we restrict our attention in the lemmas below to the other case, where as noted above we have \( W(R_F) \geq 18 \). The following lemma shows that \( q \), computed in line 0 of the pseudocode, is a good estimator of \( m \).

**Lemma 7.** \( \Pr[q - 3 \leq m \leq q] \geq 0.8 \)

Proof. Recall that in line 0 of the pseudocode, an approximate weighted model counter is invoked to obtain an estimate, \( C \), of \( w(R_F) \) with tolerance 0.8 and confidence 0.8. By the definition of approximate weighted model counting, we have \( \Pr \left[ \frac{C}{1.8} \leq w(R_F) \leq (1.8)C \right] \geq 0.8 \).
Defining \( c = C/w_{\text{max}} \), we have \( \Pr[|c - \log(1.8)| \leq \log W(R_F) - \log(1.8) \leq \log c + \log(1.8)] \geq 0.8 \). It follows that
\[
\Pr[|c - \log(1.8) - \log pivot - \log(1/W(R_F))| \leq \log(1/W(R_F) - 1 - \log c - \log pivot + \log(1.8) - \log(1/W(R_F)))] \geq 0.8.
\]
Substituting \( q = \log C - \log w_{\text{max}} + \log 1.8 - \log pivot \) and using the bounds \( w_{\text{max}} \leq 1 \), \( \log 1.8 \leq 0.85 \), and \( \log(1/W(R_F)) \leq 0.12 \) (since \( W(R_F) \geq 18 \) at line 10 of the pseudocode, as noted above), we have \( \Pr[q - 3 \leq m \leq q] \geq 0.8 \).

The next lemma provides a lower bound on the probability of generation of a witness. Let \( w_i,y,\alpha \) denote the probability that we will return a witness \( y \) with a particular value of \( i \) and \( \alpha \in (0,1)^1 \) is the value chosen on line 19. Then, \( \Pr[U] = \sum_{i=0}^{q} \frac{W(y)}{W(Y)} p_{i,y} \prod_{j=q-1}^{i-1} (1 - p_{j,y}) \), where \( Y \) is the set returned by BoundedWeightSAT on line 17. From Lemma 7, we know that \( f_m \geq 0.8 \). From line 20, we also know that \( \frac{1}{1 + \kappa \cdot pivot} \leq W(Y) \leq 1 + (1 + \kappa) \cdot pivot \). Therefore, \( \Pr[U] \geq \frac{W(y)}{1 + (1 + \kappa) \cdot pivot} \cdot p_{m,y} \cdot f_m \).

The proof is now completed by showing \( p_{m,y} \geq \frac{1}{2m} (1 - e^{-3/2}) \), as then we have \( \Pr[U] \geq \frac{0.8(1 - e^{-3/2})}{(1 + (1 + \kappa) \cdot pivot) 2^m} \geq 0.8 \cdot (1 + (1 + \kappa) \cdot pivot) 2^m \). The last inequality uses the observation that \( 1/\log pivot \leq 0.06 \).

To calculate \( p_{m,y} \), we first note that since \( y \in R_F \), the requirement \( \{y \in R_{F,h,\alpha} \} \) reduces to \( h^{-1}(\alpha) \). For \( \alpha \in (0,1)^n \), we define \( w_{m,y,\alpha} = \Pr[pivot = \frac{1}{1 + \kappa}] \leq W(R_F,\alpha) \leq 1 + (1 + \kappa) pivot \wedge h(y) = \alpha - h^{-1}(\alpha) \). Then we have \( p_{m,y} = \sum_{\alpha \in (0,1)^n} w_{m,y,\alpha} \cdot 2^{-m} \). To prove the desired bound on \( p_{m,y} \) it suffices to show that \( w_{m,y,\alpha} \geq (1 - e^{-3/2})/2^m \) for every \( \alpha \in (0,1)^m \) and \( y \in \{0,1\}^n \).

Towards this end, let us first fix a random \( y \). Now we define an indicator variable \( \gamma_{z,\alpha} \) for every \( z \in R_F \backslash \{y\} \) such that \( \gamma_{z,\alpha} = W(z) \) if \( h(z) = \alpha \), and \( \gamma_{z,\alpha} = 0 \) otherwise. Let us fix \( y \) and choose \( h \) uniformly at random from \( H_{xor}(n,m,3) \). The random choice of \( h \) induces a probability distribution on \( \gamma_{z,\alpha} \) such that \( \Pr[\gamma_{z,\alpha} = 1] = W(z) \cdot \Pr[h(z) = \alpha] = W(z)/2^m \). Since we have fixed \( y \) and since hash functions chosen from \( H_{xor}(n,m,3) \) are 3-wise independent, it follows that for every distinct \( z_a, z_b \in R_F \backslash \{y\} \), the random variables \( \gamma_{z_a,\alpha}, \gamma_{z_b,\alpha} \) are 2-wise independent. Let \( \Gamma_{\alpha} = \sum_{z \in R_F \backslash \{y\}} \gamma_{z,\alpha} \) and \( \mu_{\alpha} = \sum_{z \in W(R_F,\alpha)} \gamma_{z,\alpha} = (W(R_F) - W(y))/2^m \). Since \( pivot = \log(1/W(R_F)) \), \( pivot = (W(R_F) - 1)/2^m \leq (W(R_F) - W(y))/2^m \), we have \( \Pr[pivot \leq 1 + (1 + \kappa) pivot] \leq \Pr[W(R_F,\alpha) \leq 1 + (1 + \kappa) pivot] \leq W(R_F,\alpha) - W(y) / 2^m \). Since \( pivot = \log(1/W(R_F)) \) and the variables \( \gamma_{z,\alpha} \) are 2-wise independent and in the range \([0,1] \), we may apply Lemma 3 with \( \beta = \kappa / (1 + \kappa) \) to obtain \( \Pr[pivot \leq 1 + (1 + \kappa) pivot] \geq 1 - e^{-3/2} \). Since \( h \) is chosen at random from \( H_{xor}(n,m,3) \), we also have \( \Pr[h(y) = \alpha] = 1/2^m \). It follows that \( w_{m,y,\alpha} \geq (1 - e^{-3/2})/2^m \).

The next lemma provides an upper bound of \( w_{i,y,\alpha} \) and \( p_{i,y} \).

**Lemma 9.** For \( i < m \), both \( w_{i,y,\alpha} \) and \( p_{i,y} \) are bounded above by
\[
1/1 + \frac{1}{(1 + \kappa) W(R_F) - 1 - \log 2m^2}.
\]

**Proof.** We will use the terminology introduced in the proof of Lemma 8. Clearly, \( \mu_{\alpha} = W(R_F,\alpha) - W(y) \). Since each \( \gamma_{z,\alpha} \) takes values in \([0,1] \), \( V[\gamma_{z,\alpha}] \leq 2\epsilon \). Therefore, \( \sigma_{z,\alpha}^2 \leq \sum_{z \in R_F} \epsilon[\gamma_{z,\alpha}] \leq \sum_{z \in R_F} \epsilon[\gamma_{z,\alpha}] = \epsilon[\Gamma_{\alpha}] \leq 2^{-m}(W(R_F) - W(y)) \). So \( \Pr[pivot \leq 1 + (1 + \kappa) pivot] \leq \Pr[W(R_F,\alpha) - W(y) \leq (1 + \kappa) pivot] \). From Chebyshev’s inequality, we know that \( \Pr[|\Gamma_{\alpha} - \mu_{\alpha}| \geq \lambda \sigma_{z,\alpha}] \leq 1/\lambda^2 \) for every \( \lambda > 0 \). Then \( \Pr[W(R_F,\alpha) - W(y) \leq (1 + \kappa) W(R_F) - W(y)] \leq \Pr[pivot \leq 1 + (1 + \kappa) pivot] \leq 1/\lambda^2 \) for every \( \lambda > 0 \).
\[ \geq (1 - \frac{1 + \kappa}{2m - r}) \frac{W(R_F) - W(y)}{W(R_F)} \leq \frac{1}{(1 + \kappa)^2}. \]

Since \( h \) is chosen at random from \( H_{\text{cor}}(n, m, 3) \), we also have \( \Pr[h(y) = \alpha] = \frac{1}{2^i} \).

It follows that \( w_{i,y,\alpha} \leq \frac{W(R_F) - (1 - \frac{1 + \kappa}{2m})}{(1 - \frac{1 + \kappa}{2m - r})^2}. \)

The bound for \( p_{i,y} \) is easily obtained by noting that \( p_{i,y} = \Sigma_{\alpha \in \{0,1\}} (w_{i,y,\alpha} \cdot 2^{-i}) \). □

**Lemma 10.** For every witness \( y \in R_F \), \( \Pr[y \text{ is output}] \leq \frac{1 + \kappa}{2m} \left( \frac{W(y)}{W(R_F)} \right) - 1 \left( \frac{2^{2i} - 1}{(1 - \kappa)^2} \right). \)

**Proof.** We will use the terminology introduced in the proof of Lemma 8. Using \( \frac{p_{i,y}}{1 + \kappa} \leq W(y) \), we have

\[ \Pr[U] = \sum_{i=q-3}^{q} \frac{\Pr[U_{i,y}]}{W(y)} \left( 1 - p_{3,i,y} \right) \leq \frac{1 + \kappa}{2m} W(y) \sum_{i=q-3}^{q} p_{i,y}. \]

Now we subdivide the calculation of \( \Pr[U] \) into three cases depending on the value of \( m \).

**Case 1:** \( q - 3 \leq m \leq q \).

Now there are four values that \( m \) can take.

1. \( m = q - 3 \). We know that \( p_{i,y} \leq \Pr[h(y) = \alpha] = \frac{1}{2^i} \), so \( \Pr[U_{i,y} \leq \frac{1 + \kappa}{2m} W(y) \left( \frac{1}{W(R_F)} - 1 \right) \left( 1 - \frac{1 + \kappa}{2m} \right)^2. \)

Substituting the values of \( pivot \) and \( m \) gives \( \Pr[U_{i,y} = q - 3] \leq \frac{1 + \kappa}{2m} W(y) \left( \frac{1}{W(R_F)} - 1 \right) \left( 1 - \frac{1 + \kappa}{2m} \right)^2. \)

2. \( m = q - 2 \). For \( i \in [q - 2, q] \), we get

\[ \Pr[h(y) = \alpha] = \frac{1}{2^i} \] Using Lemma 9 we get

\[ \Pr[U_{i,y} \leq \frac{1 + \kappa}{2m} W(y) \left( \frac{1}{W(R_F)} - 1 \right) \left( 1 - \frac{1 + \kappa}{2m} \right)^2. \]

Substituting the value of \( pivot \) and maximizing \( m - q + 3 \), we get

\[ \Pr[U_{i,y} \leq \frac{1 + \kappa}{2m} W(y) \left( \frac{1}{W(R_F)} - 1 \right) \left( 1 - \frac{1 + \kappa}{2m} \right)^2. \]

3. \( m = q - 1 \). For \( i \in [q - 1, q] \), we get

Using Lemma 9 we get

\[ \Pr[U_{i,y} \leq \frac{1 + \kappa}{2m} W(y) \left( \frac{1}{W(R_F)} - 1 \right) \left( 1 - \frac{1 + \kappa}{2m} \right)^2. \]

Therefore, \( \Pr[U_{i,y} = q - 1] \leq \frac{1 + \kappa}{2m} W(y) \left( \frac{1}{W(R_F)} - 1 \right) \left( 1 - \frac{1 + \kappa}{2m} \right)^2. \)

Since \( pivot = \frac{W(R_F) - 1}{2m} > 10 \) and \( \kappa \leq 1 \), we have

\[ \Pr[U_{i,y} = q - 1] \leq \frac{1 + \kappa}{2m} W(y) \left( \frac{1}{W(R_F)} - 1 \right) \left( 1 - \frac{1 + \kappa}{2m} \right)^2. \]

4. \( m = q \). We have \( p_{q,y} \leq \Pr[h(y) = \alpha] = \frac{1}{2^q} \), and using Lemma 9 we get

\[ \Pr[U_{i,y} \leq \frac{1 + \kappa}{2m} W(y) \left( \frac{1}{W(R_F)} - 1 \right) \left( 1 - \frac{1 + \kappa}{2m} \right)^2. \]

Substituting the value of \( pivot \) and maximizing \( m - q + 3 \), we get

\[ \Pr[U_{i,y} \leq \frac{1 + \kappa}{2m} W(y) \left( \frac{1}{W(R_F)} - 1 \right) \left( 1 - \frac{1 + \kappa}{2m} \right)^2. \]

Since \( \Pr[U_{i,y} = q - 3 \leq m \leq q] = \max_{q - 3 \leq m \leq q} \left( \Pr[U_{i,y} = q] \right) \), we have

\[ \Pr[U_{i,y} = q - 3 \leq m \leq q] \leq \frac{1 + \kappa}{2m} W(y) \left( \frac{1}{W(R_F)} - 1 \right) \left( 1 - \frac{1 + \kappa}{2m} \right)^2. \]

The R.H.S. is maximized when \( m = q + 1 \). Hence \( \Pr[U_{i,y} = q] \leq \frac{1 + \kappa}{2m} W(y) \left( \frac{1}{W(R_F)} - 1 \right) \sum_{i=q-3}^{q} 1 \left( 1 - \frac{1 + \kappa}{2m} \right)^2. \)

Noting that \( pivot = \frac{W(R_F) - 1}{2m} \geq 10 \) and expanding the above summation we have

\[ \Pr[U_{i,y} = q] \leq \frac{1 + \kappa}{2m} W(y) \left( \frac{1}{W(R_F)} - 1 \right) \left( 1 - \frac{1 + \kappa}{2m} \right)^2. \]

Using \( \kappa \leq 1 \) for the first three summation terms, we obtain

\[ \Pr[U_{i,y} = q] \leq \frac{1 + \kappa}{2m} W(y) \left( \frac{1}{W(R_F)} - 1 \right) \left( 1 - \frac{1 + \kappa}{2m} \right)^2. \]

Summing up all the above cases, \( \Pr[U_{i,y} = q - 3 \leq m \leq q] \), we have

\[ \Pr[U_{i,y} = q - 3 \leq m \leq q] \leq \frac{1 + \kappa}{2m} W(y) \left( \frac{1}{W(R_F)} - 1 \right) \left( 1 - \frac{1 + \kappa}{2m} \right)^2. \]

Therefore, we have

\[ \Pr[U_{i,y} = q - 1] \leq \frac{1 + \kappa}{2m} W(y) \left( \frac{1}{W(R_F)} - 1 \right) \left( 1 - \frac{1 + \kappa}{2m} \right)^2. \]

Since \( pivot = \frac{W(R_F) - 1}{2m} > 10 \) and \( \kappa \leq 1 \), we have

\[ \Pr[U_{i,y} = q - 1] \leq \frac{1 + \kappa}{2m} W(y) \left( \frac{1}{W(R_F)} - 1 \right) \left( 1 - \frac{1 + \kappa}{2m} \right)^2. \]

□
Combining Lemmas\textsuperscript{8} and \textsuperscript{10}, the following lemma is obtained.

**Lemma 11.** For every witness $y \in R_F$, if $\varepsilon > 1.71$, then
\[
\frac{w(y)}{(1 + \varepsilon)w(R_F)} \leq \Pr[\text{BoundedWeightSAT}(F, \varepsilon, r, X) = y] \leq (1 + \varepsilon) \frac{w(y)}{w(R_F)}.
\]

Proof. In the case where $w(R_F) \leq 1 + (1 + \kappa)\text{pivot}$, the result holds because \text{BoundedWeightSAT} returns a perfect weighted-uniform sample. Otherwise, using Lemmas\textsuperscript{8} and \textsuperscript{10} and substituting $(1 + \varepsilon) = (1 + \kappa)(2.36 + \frac{0.51}{1 - \kappa^2}) = \frac{18}{17}(1 + \kappa)(2.23 + \frac{0.48}{1 - \kappa^2})$, via the inequality
\[
\frac{1.06 + \kappa}{0.8(1 - e^{-3/2})} \leq \frac{18}{17}(1 + \kappa)(2.23 + \frac{0.48}{1 - \kappa^2})
\]
we have the bounds
\[
\Pr[\text{WeightGen}(F, \varepsilon, r, X) = y] \leq \frac{18}{17}(1 + \varepsilon)\frac{W(R_F)}{W(y)} - \frac{W(R_F) - 1}{W(y)}.
\]
Using $W(R_F) \geq 18$, we obtain the desired result. \hfill $\square$

**Lemma 12.** Algorithm \text{WeightGen} succeeds (i.e. does not return $\perp$) with probability at least 0.62.

Proof. If $W(R_F) \leq 1 + (1 + \kappa)\text{pivot}$, the theorem holds trivially. Suppose $W(R_F) > 1 + (1 + \kappa)\text{pivot}$ and let $P_{\text{succ}}$ denote the probability that a run of the algorithm succeeds. Let $p_i$ with $q - 3 \leq i \leq q$ denote the conditional probability that \text{WeightGen} $(F, \varepsilon, r, X)$ terminates in iteration $i$ of the repeat-until loop (lines\textsuperscript{13}\textsuperscript{19}) with $\text{pivot} = \frac{1 + \kappa}{1 + \kappa}$.

\[
W(R_F, h, \alpha) \leq 1 + (1 + \kappa)\text{pivot}, \quad \text{given that } W(R_F) > 1 + (1 + \kappa)\text{pivot}.
\]

Then $P_{\text{succ}} = \prod_{i=q-3}^{q} p_i$. Letting $f_m = \Pr[q - 3 \leq m \leq q]$, by Lemma\textsuperscript{7} we have $P_{\text{succ}} \geq pm_f$. The theorem is now proved by using Lemma\textsuperscript{3} to show that $p_m \geq 1 - e^{-3/2} \geq 0.776$.

For every $y \in \{0, 1\}^n$ and $\alpha \in \{0, 1\}^m$, define an indicator variable $\nu_{y, \alpha}$ as follows: $\nu_{y, \alpha} = W(y)$ if $h(y) = \alpha$, and $\nu_{y, \alpha} = 0$ otherwise. Let us fix $\alpha$ and $y$ and choose $h$ uniformly at random from $H_{\text{xor}}(n, m, 3)$. The random choice of $h$ induces a probability distribution on $\nu_{y, \alpha}$, such that $\Pr[\nu_{y, \alpha} = W(y)] = \Pr[h(y) = \alpha] = 2^{-m}$ and $E[\nu_{y, \alpha}] = W(y)\Pr[\nu_{y, \alpha} = 1] = 2^{-m}W(y)$. In addition 3-wise independence of hash functions chosen from $H_{\text{xor}}(n, m, 3)$ implies that for every distinct $y_0, y_1, y_2 \in R_F$, the random variables $\nu_{y_0, \alpha}, \nu_{y_1, \alpha}$, and $\nu_{y_2, \alpha}$ are 3-wise independent.

Let $\Gamma_{\alpha} = \sum_{y \in R_F} \nu_{y, \alpha}$ and $\mu_{\alpha} = E[\Gamma_{\alpha}]$. Clearly,
\[
\Gamma_{\alpha} = W(R_F, h, \alpha) \quad \text{and} \quad \mu_{\alpha} = \sum_{y \in R_F} E[\nu_{y, \alpha}] = 2^{-m}W(R_F). \quad \text{Since } \text{pivot} = \left\lceil e^{3/2}(1 + 1/\varepsilon)^2 \right\rceil,
\]
we have $2^{-m}W(R_F) \geq e^{3/2}(1 + 1/\varepsilon)^2$, and so using Lemma\textsuperscript{3} with $\beta = \kappa/(1 + \kappa)$ we obtain
\[
\Pr\left[\frac{W(R_F)}{2^m} \left(1 - \frac{\kappa}{1 + \kappa}\right) \leq W(R_F, h, \alpha)\right]
\]
\[
\leq (1 + \kappa)\frac{W(R_F)}{2^m} > 1 - e^{-3/2}. \quad \text{Simplifying and noting that } \frac{1 + \kappa}{1 + \kappa} < \kappa \text{ for all } \kappa > 0,
\]
we have $\Pr\left[(1 + \kappa)^{-1} \frac{W(R_F)}{2^m} \leq W(R_F, h, \alpha)\right] \leq (1 + \kappa)\frac{W(R_F)}{2^m}$ and $1 + (1 + \kappa)\text{pivot} = 1 + \frac{1 + (1 + \kappa)(W(R_F) - 1)}{2}(1 + (1 + \kappa)W(R_F))$.

Therefore, $p_m = \Pr\left[pivot \leq \frac{W(R_F)}{2^m}\right] \leq 1 + (1 + \kappa)\text{pivot} \geq \Pr\left[(1 + \kappa)^{-1} \frac{W(R_F)}{2^m} \leq \frac{W(R_F)}{2^m}\right] \geq 1 - e^{-3/2}$. \hfill $\square$

By combining Lemmas\textsuperscript{11} and\textsuperscript{12} we get the following:

**Theorem 3.** Given a CNF formula $F$, tolerance $\varepsilon > 1.71$, lift bound $r$, and independent support $S$, for every $y \in R_F$ we have $\frac{w(y)}{(1 + \varepsilon)w(R_F)} \leq \Pr[\text{WeightGen}(F, \varepsilon, r, X) = y] \leq (1 + \varepsilon)\frac{w(y)}{w(R_F)}$.

Also, \text{WeightGen} succeeds (i.e. does not return $\perp$) with probability at least 0.62.

**Theorem 4.** Given an oracle for SAT, \text{WeightGen}(F, \varepsilon, r, S) runs in time polynomial in $r, |F|$ and $1/\varepsilon$ relative to the oracle.

Proof. Referring to the pseudocode for \text{WeightGen}, the runtime of the algorithm is bounded by the runtime of the constant number (at most 5) of calls to BoundedWeightSAT and one call to WeightMC (with parameters $\delta = 0.2, \varepsilon = 0.8$). As shown in Theorem\textsuperscript{1}, the call to WeightMC can be done in time polynomial in $|F|$ and $r$ relative to the oracle. Every invocation of BoundedWeightSAT can be implemented by at most $(r \cdot \text{pivot} + 1)$ calls to a SAT oracle (as in the proof of Theorem\textsuperscript{2}), and the total time taken by all calls to BoundedWeightSAT is polynomial in $|F|, r$ and pivot relative to the oracle. Since pivot = $O(1/e^2)$, the runtime of WeightGen is polynomial in $r, |F|$ and $1/e$ relative to the oracle. \hfill $\square$

**C Analysis of Partitioned WeightMC**

**Theorem 5.** If PartitionedWeightMC($F, \varepsilon, \delta, S, L, H$) returns $c$ (and all arguments are in the required ranges), then
\[
\Pr \left[ c \neq \perp \land (1 + \varepsilon)^{-1}w(R_F) \leq c \leq (1 + \varepsilon)w(R_F) \right] \geq 1 - \delta.
\]

Proof. For future reference note that since $N \geq 1$ and $\delta < 1$, we have $(1 - \delta)^N = (1 - \delta/N)^N \geq 1 - \delta.$
Define $G_m = F \wedge (H/2^m < w(X) \leq H/2^{m-1})$, the formula passed to WeightMC in iteration $m$. Clearly, we have $w(R_F) = \sum_{m=1}^{N} w(R_{G_m})$. Since $w(\cdot)$ is poly-time computable, the NP oracle used in WeightMC can decide the satisfiability of $G_m$, and so WeightMC will return a value $d_m$. Now since $H/2^m$ and $H/2^{m-1}$ are lower and upper bounds respectively on the weights of any solution to $G_m$, by Theorem 1 we have

$$\Pr\left[d_m \neq \bot \land (1 + \varepsilon)^{-1}w(R_{G_m}) \leq d_m \leq (1 + \varepsilon)w(R_{G_m})\right] \geq 1 - \delta'$$

for every $m$, and so

$$\Pr\left[c \neq \bot \land (1 + \varepsilon)^{-1}w(R_F) \leq c \leq (1 + \varepsilon)w(R_F)\right]$$

$$= \Pr\left[c \neq \bot \land (1 + \varepsilon)^{-1} \sum_{m} w(R_{G_m}) \leq c \leq (1 + \varepsilon) \sum_{m} w(R_{G_m})\right]$$

$$\geq (1 - \delta')^N \geq 1 - \delta$$

as desired.

**Theorem 6.** With access to an NP oracle, the runtime of PartitionedWeightMC($F, \varepsilon, \delta, S, L, H$) is polynomial in $|F|, 1/\varepsilon, \log(1/\delta'), \log r = \log(H/L)$.

**Proof.** Put $r = H/L$. By Theorem 2 each call to WeightMC runs in time polynomial in $|G|, 1/\varepsilon$ and $\log(1/\delta')$ (the tilt bound is constant). Clearly $|G|$ is polynomial in $|F|$. Since $\delta' = \delta/N$ we have $\log(1/\delta') = \log(N/\delta) = O(\log((\log r)/\delta)) = O(\log \log r + \log(1/\delta))$. Therefore each call to WeightMC runs in time polynomial in $|F|, 1/\varepsilon, \log(1/\delta'), \log \log r$. Since there are $N = O(\log r)$ calls, the result follows.