AAAI Technical Report FS-12-06
Machine Aggregation of Human Judgment

Generalized Weighted Model Counting: An Efficient Monte-Carlo Meta-Algorithm

Lirong Xia
SEAS, Harvard University, USA,
lxia@seas.harvard.edu

Abstract

In this paper, we focus on computing the prices of securities represented by logical formulas in combinatorial prediction markets when the price function is represented by a Bayesian network. This problem turns out to be a natural extension of the weighted model counting (WMC) problem (Sang, Bearne, and Kautz 2005), which we call *generalized weighted model counting (GWMC)* problem. In GWMC, we are given a logical formula F and a polynomial-time computable weight function. We are asked to compute the total weight of the valuations that satisfy F.

Based on importance sampling, we propose a Monte-Carlo meta-algorithm that has a good theoretical guarantee for formulas in disjunctive normal form (DNF). The meta-algorithm queries an oracle algorithm that computes marginal probabilities in Bayesian networks, and has the following theoretical guarantee. When the weight function can be approximately represented by a Bayesian network for which the oracle algorithm runs in polynomial time, our meta-algorithm becomes a *fully polynomial-time randomized approximation scheme (FPRAS)*.

Introduction

Prediction markets are a type of financial markets that aggregates agents' probabilistic information about some random event. The *Iowa Electronic Market* and *Intrade* are two examples of real-world prediction markets with a long history of tested results (Berg et al. 2008; Berg, Nelson, and Rietz 2008). See (Chen and Pennock 2010) for a recent survey.

In many real-life situations, the number of outcomes of the random event is exponentially large and has a combinatorial structure. Such situations are called *combinatorial prediction markets* (Fortnow et al. 2004; Hanson 2003; 2007; Chen et al. 2008; Chen, Goel, and Pennock 2008). For example, in the NCAA men's basketball tournament, there are 64 teams and therefore 63 matches in total to predict. Each match can be seen as a binary variable. It follows that the prediction market for this tournament has $2^{63} \approx 9.2 \times 10^{18}$ outcomes. In such situations, even though usually the price of a security corresponding to a single outcome is computationally easy, it is not surprising to see that computing and updating the prices by directly using the cost function becomes computationally intractable. In fact, pricing LMSR-based combinatorial prediction markets is #P-hard (Chen et al. 2008).

To overcome this computational intractability, one promising idea is to use a *Bayesian network* to represent the prices of the securities corresponding to disjoint and exhaustive outcomes. This idea was first explored by (Chen, Goel, and Pennock 2008) for a class of LMSR-based combinatorial prediction markets for tournaments. They modeled the market price distribution by a Bayesian network whose graph is a balanced binary tree, and identified two types of *structure-preserving* securities: after any shares of any such securities are sold, the updated market price distribution can still be represented by a Bayesian network with the same structure. A followup work obtained a full characterization for structure-preserving securities for any network structure (Pennock and Xia 2011).

However, the two papers mentioned above focused on price-updating after trading some shares of a security rather than the #P-hard problem of computing the price of a given security when the market prices are represented by a Bayesian network. In this paper, we tackle this problem by designing a sample-based Monte-Carlo algorithm. Mathematically, the problem corresponds to computing the total weight (prices) of valuations (outcomes) that satisfy a given logical formula F, which is a natural generalization of the *model counting problem (a.k.a. #SAT)*, one of the most important problems in artificial intelligence. Our main technical contribution is a Monte-Carlo meta-algorithm (Algorithm 1) based on importance sampling that uses two subprocedures. One procedure computes the marginal probabilities in Bayesian networks (we call this the oracle algorithm), and the other computes the weight of a given valuation.[1] Our algorithm has the following theoretical guarantee: Suppose the weights can be approximately represented by a Bayesian network π^* (we will formally define this in Definition 2). For any DNF formula F and any error rate ϵ, Algorithm 1 is an unbiased estimator for the total weight of the outcomes that satisfies F, and with probability larger than $3/4$ the multiplicative error is no more than ϵ. Moreover, the algorithm runs in polynomial time plus polynomial calls to the oracle algorithm, where the running time (respectively

[1]The second procedure is usually given as part of the input.

the number of calls) is polynomial in the input size and $1/\epsilon$.

Of course the performance of the meta-algorithm depends on both π^* and the oracle algorithm that computes the marginal probabilities. One important corollary is that if there exits a polynomial time algorithm that computes marginal probabilities in π^*, then Algorithm 1 is a *fully polynomial-time randomized approximation scheme (FPRAS)*. For example, when π^* can be represented by a Bayesian network whose structure is a polytree,[2] then we can use the celebrated *belief-propagation algorithm* (Pearl 1988) to obtain an FPRAS. Therefore, we automatically obtain another FPRAS for pricing LMSR-based combinatorial prediction markets for tournaments (Xia and Pennock 2011).

Related Work. Our work is closest to the following two lines of research. Conceptually, it is closely related to and is a natural generalization of the weighted model counting problems (Sang, Bearne, and Kautz 2005). We will see that WMC is a very special case of GWMC where the weight function can be represented by a Bayesian network without edges (Example 1). Therefore, our main algorithm is an FPRAS for WMC. We are not aware of previous work showing an FPRAS for WMC. On the technical level, the main application of WMC is to solve Bayesian inference by reducing a Bayesian inference problem to a WMC problem. Our meta-algorithm, on the other hand, aims at exploring the power of Bayesian inference algorithms to compute GWMC for various applications, including pricing combinatorial prediction markets.

Technically, our work extends the idea of the FPRAS algorithm for model counting by Karp, Luby, and Madras (Karp, Luby, and Madras 1989) (KLM for short), and the FPRAS algorithm for pricing combinatorial prediction markets for tournaments in our previous work (Xia and Pennock 2011). However, the KLM algorithm only dealts with (unweighted) model counting problems. While the framework of our meta-algorithm is similar to Algorithm 1 in (Xia and Pennock 2011), our meta-algorithm uses a Bayesian inference algorithm as a oracle, and the algorithm in (Xia and Pennock 2011) heavily depends on the assumption that π^* can be represented by a Bayesian network whose structure is a balanced binary tree.

Other related work including designing combinatorial prediction markets based on convex optimization (Abernethy, Chen, and Vaughan 2012; Dudik, Lahaie, and Pennock 2012), which are technically quite different from the setup of this work.

Comments on Significance and Limitations. Admittedly, both the conceptual extension of WMC to GWMC and the technical extension of the KLM algorithm and the FPRAS in (Xia and Pennock 2011) to our meta-algorithm are quite natural. However, we feel that their combination is interesting because our meta-algorithm shows that there is some hope to develop efficient algorithms for a much more general problem than WMC. Methodologically, our paper introduces the idea of employing Bayesian network inference algorithms to handle GWMC problems, while previous work focused on how to develop techniques to facilitate inference in Bayesian networks. As we will see in Example 2, GWMC has found applications in pricing general combinatorial prediction markets. Therefore, we feel that designing computationally tractable algorithms for GWMC is a promising direction for future research, which bridges different important research directions including Bayesian inference, SAT and model counting, and electronic commerce.

Our meta-algorithm has mainly two limitations. Most importantly, being an FPRAS is a good theoretical guarantee, but it is not clear yet how well our meta-algorithm works in practice. On the technical level, the meta-algorithm only work well for DNFs. It is important to develop practical algorithms for GWMC for CNFs. One natural idea is to convert CNFs to DNFs, and then apply our meta-algorithm. This approach will us additive error bounds. More generally, how to design algorithms for GWMC for CNF is an interesting future direction.

[2]A polytree is a directed graph that does not contain undirected cycles.

Preliminaries

Combinatorial Prediction Markets

In *combinatorial prediction markets* (Chen et al. 2008), the set of outcomes Ω has a combinatorial structure. That is, each outcome is characterized by the values of a set of variables $\mathcal{V} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$, where for each $k \leq n$, $\mathbf{x}_k$ takes a value in a domain $\Omega_k = \{0, 1\}$. It follows that $\Omega = \Omega_1 \times \cdots \times \Omega_n$. In this paper, a security is represented by a logical formula F over $\mathcal{V}$ in *conjunctive normal form (CNF)*. That is, $F = C_1 \wedge \cdots \wedge C_K$, , where for any $j \leq k$, $C_j = l^j_1 \wedge \cdots \wedge l^j_{s_j}$, and l^j_i is either $\mathbf{x}$ or $\neg\mathbf{x}$ for some variable $\mathbf{x}$. C_j is called a *clause* and l^j_i is called a *literal*. We assume that no clause contains both $\mathbf{x}$ and $\neg\mathbf{x}$ for any variable $\mathbf{x}$. If F is satisfied under the eventual true outcome (which is a valuation over $\mathcal{V}$), then the market maker should pay the agent \$1 for each share of F the agent holds; otherwise the agent receives nothing for holding F.

At any point, we let $p(\cdot)$ denote the price function. By definition, the instantaneous price of F is the sum of the prices of the securities that correspond to the valuations under which F is satisfied. That is, $p(F) = \sum_{\vec{x}:F(\vec{x})=1} p(\vec{x})$.

General Importance Sampling

Importance sampling is a general technique for Monte-Carlo methods that reduces the variance of estimation, which in turn improves the convergence rate. Suppose we want to evaluate the expectation of a function $f : \{1, \dots, N\} \to \mathbb{R}$ when the variable is chosen from a probability distribution π over $\{1, \dots, N\}$. That is, we want to evaluate the expectation of f w.r.t. π, denoted by $E[f; \pi]$. The most straightforward Monte-Carlo method is to generate Z samples $X_1, \dots, X_Z$ i.i.d. according to π, and use $\frac{1}{Z}\sum_{i=1}^{Z} f(X_i)$ as an unbiased estimator for $E[f; \pi]$. The convergence rate is guaranteed by the following lemma, which follows directly from Chebyshev's inequality.

Lemma 1 (Follows from Chebyshev's inequality) *Let $H_1, \dots, H_Z$ be i.i.d. random variables with $\mu = E[H_i]$ and* *variance σ^2. If $Z \geq 4\sigma^2/(\epsilon^2\mu^2)$, then,*

$$Pr(|\frac{1}{Z}\sum_{i=1}^{Z} H_i - \mu| < \epsilon\mu) \geq 3/4$$

This lemma illustrates that for a fixed ϵ, the smaller σ^2/μ^2, the faster this sampling method converges. Importance sampling reduces the variance by generating the outcomes that have higher f values more often. Suppose we have another distribution π^* such that for every outcome i, $[\pi^*(i) = 0] \implies [f(i)\pi(i) = 0]$. We can then use π^* to provide an unbiased estimator for $E[f;\pi]$ as follows. Let H denote the random variable that takes $\frac{f(i)\pi(i)}{\pi^*(i)}$ with probability $\pi^*(i)$. We generate Z i.i.d. samples of H, denoted by $H_1, \ldots, H_Z$, and use $\frac{1}{Z}\sum_{i=1}^{Z} H_i$ as an estimator for $E[f;\pi]$. It is easy to check that this estimator is unbiased, and $\text{Var}(H)/E[H]^2$ might be significantly smaller than $\text{Var}(f)/E[f;\pi]^2$.

A good π^* can greatly reduce the ratio of variance over square expectation, therefore in turn improving the performance of the Monte-Carlo method. The best scenario is that for any outcome i, $\pi^*(i)$ is proportional to $f(i)\pi(i)$. Then, the variance becomes 0 and we only need 1 sample to calculate the expectation. In practice, sometimes we would like to choose π^* that is a good approximation to $f(i)\pi(i)$ and is much easier to handle, as we will see in our meta-algorithm.

An FPRAS for # DNF

An algorithm A is a *fully polynomial-time randomized approximation scheme (FPRAS)* for a function f, if for any input x and any error rate ϵ, (1) the output of the algorithm A is in $[(1-\epsilon)f(x), (1+\epsilon)f(x)]$ with probability at least $3/4$,[3] (2) the runtime of A is polynomial in $1/\epsilon$ and the size of x.

To better present our algorithm, we recall an FPRAS for the #DNF problem by Karp, Luby, and Madras (Karp, Luby, and Madras 1989) (KLM for short). The #DNF problem has been proven to be #P-complete (Valiant 1979). In a #DNF instance, we are given a DNF formula $F = C_1 \vee \cdots \vee C_k$ over $\{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$, and we are asked to compute the number of valuations under which $F = 1$. Let π_u denote the uniform distribution over all valuations. The #DNF problem is equivalent to computing $2^m \cdot E[F;\pi_u]$.

One naïve Monte-Carlo method is to generate Z valuations i.i.d. uniformly at random, and counts how many times F is satisfied, denoted by X^Z. Clearly $2^m \cdot X^Z/Z$ is an unbiased estimator for the solution to the #DNF instance. However, when the solution is small, $\text{Var}(X^Z/Z)/E[X^Z/Z]^2$ can be exponentially large. Consequently, the naïve Monte-Carlo method might have a slow convergence rate (Lemma 1). For example, if there is only one valuation that satisfies F, then the variance of X^Z/Z is approximately $1/2^m$, and the expectation of X^Z/Z is $1/2^m$, which means that $\text{Var}(X^Z/Z)/E[X^Z/Z]^2$ is approximately 2^m. Therefore, the convergence rate might be very slow.

[3]By using the *median of means* method, for any $\delta < 1$, the success rate of an FPRAS can be increased to $1-\delta$, at the cost of increasing the runtime by a multiplicative factor of $\ln(\delta^{-1})$ (cf. Exercise 28.1 in (Vazirani 2001)).

The KLM algorithm reduces the variance of by only generating valuations that satisfies F. We next recall a slight variant of the KLM algorithm in (Xia and Pennock 2011), which uses the uniform distribution π_u for better presentation. For any clause C_j and any probability distribution π, we let $\pi(C_j)$ denote the marginal probability of the partial valuation that corresponds to the literals in C_j. For example, if $C_j = \mathbf{x}_1 \wedge \neg x_2$, then $\pi(C_j) = \pi(\mathbf{x}_1 = 1, \mathbf{x}_2 = 0)$. The KLM algorithm is composed of the following three steps in each iteration.

(1) Let $G = \sum_{j'} \pi_u(C_{j'})$. Choose a clause C_j with probability $\pi_u(C_j)/G$;

(2) then choose a valuation $\vec{x}$ that satisfy C_j with probability $\pi_u(\vec{x}|C_j)$;

(3) finally, compute the number of clauses $\vec{x}$ satisfies, denoted by $n(\vec{x})$, and add $\dfrac{F(\vec{x})G}{2^m\pi_u(\vec{x})n(\vec{x})}$ to a counter K.

Given a error rate $\epsilon > 0$, let $Z = 4k^4/\epsilon^2$. After Z iterations, the algorithm outputs $2^m K/Z$. Let H denote the random variable corresponding to $\dfrac{F(\vec{x})G}{2^m\pi_u(\vec{x})n(\vec{x})}$. Since $n(\vec{x}) \leq k$, it can be checked that $\text{Var}(H)/E[H]^2 \leq k^4$ and H is an unbiased estimator to $F \cdot \pi_u$. It follows from Lemma 1 that the KLM algorithm is an FPRAS for #DNF.

Generalized Weighted Model Counting

In this section, we formally model the computational problem of pricing in combinatorial prediction markets as a generalized weighted model counting problem.

Definition 1 *In the* generalized weighted model counting (GWMC) *problem, we are given a logical formula F and a polynomial-time computable weight function w defined over all valuations of $\mathcal{X}$. We are asked to compute the total weight of the valuations that satisfy F, denoted by $W(F)$. Formally,*

$$W(F) = \textstyle\sum_{\vec{x}} w(\vec{x})F(\vec{x}) = \sum_{\vec{x}:F(\vec{x})=1} w(\vec{x})$$

We note that w can be represented by a probability distribution π_w over $\mathcal{X}$ and a total weight W, where $W = \sum_{\vec{x}} w(\vec{x})$ and $\pi_w(\vec{x}) = w(\vec{x})/W$. Therefore, if π_w can be represented by a compact Bayesian network, then w is polynomial-time computable. However, inference under simple Bayesian network might still be NP-hard (Cooper 1990). Our algorithm will use a Bayesian network π^* that approximately represents w. We next define the notion of such approximation.

Definition 2 *For any $c > 1$, we say that a probability distribution π^* is a c*-approximation *to π_w, if for any outcome $\vec{x}$, $\frac{1}{c}\pi_w(\vec{x}) \leq \pi^*(\vec{x}) \leq c\pi_w(\vec{x})$.*

Of course we can always set $\pi^* = \pi_w$, but a good π^* may greatly reduce computational complexity. Such an approximation π^* may come from expert knowledge. We next show that GWMC is an extension of weighted model counting (WMC) (Chavira and Darwiche 2008) with a very simple Bayesian network, and can be used to compute prices in combinatorial prediction markets (Xia and Pennock 2011).

In WMC, we are given a propositional logical formula F and a weight $\bar{w}(\vec{x})$ for each valuation $\vec{x}$. We are asked to compute $\sum_{\vec{x}:F(\vec{x})=1} \bar{w}(\vec{x})$, that is, the total weight of the valuations that satisfy F. The weight function is represented compactly in the following way: Each literal contributes a multiplicative factor to the weight of a valuation. More precisely, for each variable $\mathbf{x}$ there are two weights $w^0_{\mathbf{x}}$ and $w^1_{\mathbf{x}}$. For any valuation x_i of $\mathbf{x}_i$, let $\bar{w}(x_i) = w^0_{\mathbf{x}}$ if and only if $x_i = 0$ and let $\bar{w}(x_i) = w^1_{\mathbf{x}}$ if and only if $x_i = 1$. For any valuation $\vec{x}$ of all variables, let $\bar{w}(\vec{x}) = \prod_i \bar{w}(x_i)$.

Example 1 *WMC is an special case of GWMC. To see this, we show how to model $\bar{w}$ as a simple Bayesian network. Suppose in the BN there are no edges, and for any variable $\mathbf{x}$, let $\pi_w(\mathbf{x} = 1) = w^1_{\mathbf{x}}$ and $\pi_w(\mathbf{x} = 0) = w^0_{\mathbf{x}}$. Let π_w denote the probability distribution and $W = \sum_{\vec{x}} \bar{w}(\vec{x})$. It follows that for any valuation $\vec{x}$, $\bar{w}(\vec{x}) = W\pi_w(\vec{x})$.*

Example 2 *In combinatorial prediction market, for each valuation $\vec{x}$, the price for its corresponding security, denoted by $p(\vec{x})$ and usually can be computed in polynomial time. When a security is represented by a logical formula F, the price of which is defined to be $p(F) = \sum_{\vec{x}:F(\vec{x})=1} p(\vec{x})$. Computing $p(F)$ is a GWMC problem, where we let $w = p$.*

The Meta Algorithm

The meta-algorithm we present in this section contains two procedures. (1) Procedure CompMargin is the oracle algorithm that computes the marginal probability in a Bayesian network. (2) Procedure QueryWeight returns the weight of a given valuation. The second procedure is usually given as part of the input. First, we do not specify Procedure CompMargin, and will evaluate the computational complexity of Algorithm 1 by number of calls to Procedure CompMargin. The input of our meta-algorithm consists in F, w, ϵ, and π^* that is a c-approximation to π_w for some constant c.

```
Procedure CompMargin($\pi^*$,$C_j$)
  Input: $\pi^*$ and a conjunction of literals $C_j$.
  Output: $\pi^*(C_j)$.
```

```
Procedure QueryWeight($\cdot$)
  Input: A valuation $\vec{x}$.
  Output: $w(\vec{x})$.
```

The meta-algorithm is similar to the KLM algorithm and the Algorithm 1 in (Xia and Pennock 2011). It contains the following three steps in each iteration.

(1) Compute $G = \sum_{j'} \pi^*(C_{j'})$ by calling Procedure CompMargin for k times. Choose a clause C_j with probability $\pi^*(C_j)/G$.

(2) Choose a valuation $\vec{x}$ with probability $\pi^*(\vec{x}|C_j)$ from all valuations that satisfy C_j by calling Procedure GenerateValuation, which calls Procedure CompMargin for no more than $2m$ times.

(3) Finally, compute the number of clauses $\vec{x}$ satisfies, denoted by $n(\vec{x})$, and add $\dfrac{w(\vec{x})G}{\pi^*(\vec{x})n(\vec{x})}$ to a counter N by calling Procedure QueryWeight once.

Procedure QueryWeight generates values of variables in $\mathcal{X}$ sequentially in a way similarly to the Monte-Carlo sampling algorithm in (Gogate and Dechter 2008). More precisely, given j, w.l.o.g. let $C_j = \mathbf{x}_1 \wedge \mathbf{x}_2 \wedge \cdots \wedge \mathbf{x}_{m'}$. Let $x_1 = \cdots = x_{m'} = 1$. Suppose the values of $\mathbf{x}_1, \ldots, \mathbf{x}_t$ has been determined for some $m' \leq t < m$, such that for every $i \leq t$, $\mathbf{x}_i = x_i$. Let $p_1 = \pi^*(\mathbf{x}_1 = x_1, \ldots, \mathbf{x}_t = x_t, \mathbf{x}_{t+1} = 1)$ and $p_0 = \pi^*(\mathbf{x}_1 = x_1, \ldots, \mathbf{x}_t = x_t, \mathbf{x}_{t+1} = 0)$. p_1 and p_2 can be computed by Procedure CompMargin. We let $\mathbf{x}_{t+1} = \begin{cases} 1 & \text{with probability } p_1/(p_1+p_0) \\ 0 & \text{with probability } p_0/(p_1+p_0) \end{cases}$ and let $t \leftarrow t + 1$.

```
Algorithm 1: GWMC
  Input: $w$, $\pi^*$, $\epsilon$, and a DNF formula $F = C_1 \vee \cdots \vee C_k$.
  Output: An estimation for $W(F)$.
1 Compute $G = \sum_{j' \leq k}$ CompMargin$(C_{j'})$.
2 for $i = 1$ to $Z = 4c^4k^4/\epsilon^2$ do
3     Choose an index $j$ with probability $\frac{\text{CompMargin}(C_j)}{G}$.
4     Call GenerateValuation$(\pi^*, C_j)$ to choose a valuation $\vec{x}$ with probability $\pi^*(\vec{x}|C_j)$ from all valuations that satisfy $C_j$.
5     Compute $n(\vec{x}) = |\{j' : C_{j'}(\vec{x}) = 1\}|$.
6     Let $N \leftarrow N + \frac{\text{QueryWeight}(\vec{x})G}{\pi^*(\vec{x})n(\vec{x})}$.
7 end
8 return $N/Z$.
```

```
Procedure GenerateValuation($\pi^*$,$C_j$)
  Input: $\pi^*$ and a conjunction of literals $C_j$.
  Output: A valuation $\vec{x}$ that satisfies $C_j$ w.p. $\pi^*(\vec{x}|S_j)$.
1 for any variable $\mathbf{x}$ whose literals appear in $C_j$ do
2     If $C_j$ contains $\mathbf{x}$, then let $\mathbf{x} = 1$; otherwise let $\mathbf{x} = 0$.
3 end
4 Let $C = C_j$.
5 while there exists a variable $\mathbf{x}$ such that $C$ does not contain $\mathbf{x}$ or $\neg\mathbf{x}$ do
6     Let $p_1 =$ CompMargin$(\pi^*, C \wedge \mathbf{x})$, let $p_0 =$ CompMargin$(\pi^*, C \wedge \neg\mathbf{x})$.
7     Choose $\mathbf{x} = \begin{cases} 1 & \text{w.p. } \frac{p_1}{p_1+p_0}\text{, and then let } C \leftarrow C \wedge \mathbf{x} \\ 0 & \text{w.p. } \frac{p_0}{p_1+p_0}\text{, and then let } C \leftarrow C \wedge \neg\mathbf{x} \end{cases}$
8 end
9 return $\vec{x}$.
```

Theorem 1 *If π^* is a c-approximation to π_w for some constant c, then given any $\epsilon > 1$, Algorithm 1 is an unbiased estimator for $W(F)$, and*

$$Pr\left((1-\epsilon)W(F) < Output < (1+\epsilon)W(F)\right) > \frac{3}{4}\,. \quad (1)$$

The running time of Algorithm 1 is a polynomial function in $1/\epsilon$ and the input size, plus polynomial number of calls to Procedure CompMargin.

Proof of Theorem 1: We first prove that Algorithm 1 is an unbiased estimator for $W(F)$. Let X_i (for all $1 \leq i \leq Z$) denote the random variable that corresponds to the ith sample added to N in Step 6. Then, $E(X_i) = \sum_{\vec{x}:F(\vec{x})=1} \frac{w(\vec{x})G}{\pi^*(\vec{x})n(\vec{x})} \times \frac{\pi^*(\vec{x})n(\vec{x})}{G} = \sum_{\vec{x}:F(\vec{x})=1} w(\vec{x}) = W(F)$. Therefore, the output of Algorithm 1 is an unbiased estimator for $W(F)$.

To prove Inequality (1), we note that $X_i = \frac{w(\vec{x})G}{\pi^*(\vec{x})n(\vec{x})} = W(F)G\frac{\pi_w(\vec{x})}{\pi^*(\vec{x})n(\vec{x})}$. Because π^* is a c-approximation to π_w and $1 \leq n(\vec{x}) \leq k$, we have $\frac{1}{ck}E(X_i) \leq X_i \leq ckE(X_i)$, which means that $Var(X_i)/(E(X_i)^2) \leq (ck)^4$. Inequality (1) follows after Lemma 1.

Algorithm 1 calls Procedure CompMargin for k times in Step 1 and calls Procedure GenerateValuation for $4c^4k^4/\epsilon^2$ times, in each of which Procedure CompMargin is called for no more than $2m$ times. Therefore, the total number of times Algorithm 1 calls Procedure CompMargin is polynomial in $1/\epsilon$ and the input size. □

Remark: the power of Theorem 1 is that even if π_w cannot be represented by a Bayesian network where inference is computationally easy, it might still be possible to find a good approximation π^* where inference is computationally easy. Since inference for polytree-structured Bayesian networks can be done in polynomial time by the belief-propagation algorithm (Pearl 1988), we immediately have the following corollary.

Corollary 1 *If π^* is a c-approximation to π_w for some constant c and π^* can be represented by a polytree-structured Bayesian network, then Algorithm 1 is an unbiased FPRAS for $W(F)$ where the oracle algorithm is the belief-propagation algorithm.*

We have shown in Example 1 that WMCs are GWMCs where π_w can be represented by a Bayesian network without edges. Also, it is assumed in (Xia and Pennock 2011) that for combinatorial prediction markets for tournaments, there exists a Bayesian network whose structure is a tree and is a c-approximation to the price function. Since both structures are polytrees, it follows immediately after Corollary 1 that Algorithm 1 is an FPRAS for these two problems if we use the belief-propagation algorithm as Procedure CompMargin.

Summary and Future Work

In this paper, we propose an efficient Monte-Carlo meta-algorithm to compute GWMC for DNF formulas, motivated by applications in pricing combinatorial prediction markets. Our algorithm is an FPRAS if the weight function can be approximately represented by a polytree-structured Bayesian network. There are many directions for future research. Technically, it would be interesting to develop techniques for GWMC for CNF formulas, for example, we can study how to extend techniques developed for WMC to GWMC. Another important direction is to find more applications of GWMC and test the performance of the algorithms for real-world applications.

Acknowledgments

The author is supported by NSF under Grant #1136996 to the Computing Research Association for the CIFellows Project. The author thanks Carla Gomes, David Pennock, and Toby Walsh for helpful comments and suggestions. The author also thanks an anonymous IJCAI-11 reviewer of (Xia and Pennock 2011) for a very insightful comment that inspired part of this work.

References

Abernethy, J.; Chen, Y.; and Vaughan, J. W. 2012. Efficient market making via convex optimization, and a connection to online learning. *ACM Transactions on Economics and Computation*.

Berg, J. E.; Forsythe, R.; Nelson, F. D.; and Rietz, T. A. 2008. Results from a dozen years of election futures markets research. *The Handbook of Experimental Economics Results* 1:742–751.

Berg, J.; Nelson, F.; and Rietz, T. 2008. Prediction market accuracy in the long run. *International Journal of Forecasting* 24:285–300.

Chavira, M., and Darwiche, A. 2008. On probabilistic inference by weighted model counting. *Artificial Intelligence* 172(6–7):772–799.

Chen, Y., and Pennock, D. M. 2010. Designing markets for prediction. *AI Magazine* 31:42–52.

Chen, Y., and Vaughan, J. W. 2010. A new understanding of prediction markets via no-regret learning. In *Proceedings of the 11th ACM conference on Electronic commerce*, EC '10, 189–198.

Chen, Y.; Fortnow, L.; Lambert, N.; Pennock, D. M.; and Wortman, J. 2008. Complexity of combinatorial market makers. In *Proceedings of the ACM Conference on Electronic Commerce (EC)*, 190–199.

Chen, Y.; Goel, S.; and Pennock, D. M. 2008. Pricing combinatorial markets for tournaments. In *STOC '08: Proceedings of the 40th annual ACM Symposium on Theory of Computing*, 305–314.

Cooper, G. F. 1990. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence* 42(2–3):393–405.

Dudik, M.; Lahaie, S.; and Pennock, D. M. 2012. A tractable combinatorial market maker using constraint generation. In *Proceedings of the ACM Conference on Electronic Commerce (EC)*.

Fortnow, L.; Kilian, J.; Pennock, D. M.; and Wellman, M. P. 2004. Betting Boolean-style: a framework for trading in securities based on logical formulas. *Decision Support Systems* 39(1):87–104.

Gogate, V., and Dechter, R. 2008. Studies in solution sampling. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 271–276.

Hanson, R. 2003. Combinatorial information market design. *Information Systems Frontiers* 5(1):107–119.

Hanson, R. 2007. Logarithmic market scoring rules for modular combinatorial information aggregation. *Journal of Prediction Markets* 1:3–15.

Karp, R. M.; Luby, M.; and Madras, N. 1989. Monte-Carlo approximation algorithms for enumeration problems. *J. Algorithms* 10:429–448.

Pearl, J. 1988. *Probabilistic reasoning in intelligent systems: Networks of plausible inference*. Morgan Kaufmann Publishers Inc.

Pennock, D. M., and Xia, L. 2011. Price updating in combinatorial prediction markets with bayesian networks. In *Proceedings of the 27th Annual Conference on Uncertainty in Artificial Intelligence (UAI)*.

Sang, T.; Bearne, P.; and Kautz, H. 2005. Performing bayesian inference by weighted model counting. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 475–481.

Valiant, L. 1979. The complexity of enumeration and reliability problems. *SIAM J. Computing* 8(3):410–421.

Vazirani, V. 2001. *Approximation Algorithms*. Springer Verlag.

Xia, L., and Pennock, D. M. 2011. An Efficient Monte-Carlo Algorithm for Pricing Combinatorial Prediction Markets for Tournaments. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI)*.