An Efficient Monte-Carlo Algorithm for Pricing Combinatorial Prediction Markets for Tournaments

Lirong Xia[*]
Department of Computer Science
Duke University
Durham, NC 27708, USA
lxia@cs.duke.edu

David M. Pennock
Yahoo! Research New York
111 West 40th Street, 17th floor
New York, NY 10018
pennockd@yahoo-inc.com

Abstract

Computing the market maker price of a security in a combinatorial prediction market is #P-hard. We devise a fully polynomial randomized approximation scheme (FPRAS) that computes the price of any security in disjunctive normal form (DNF) within an ϵ multiplicative error factor in time polynomial in $1/\epsilon$ and the size of the input, with high probability and under reasonable assumptions. Our algorithm is a Monte-Carlo technique based on importance sampling. The algorithm can also approximately price securities represented in conjunctive normal form (CNF) with additive error bounds. To illustrate the applicability of our algorithm, we show that many securities in Yahoo!'s popular combinatorial prediction market game called *Predictalot* can be represented by DNF formulas of polynomial size.

1 Introduction

A prediction market turns a random variable into a tradable financial security of the form "\$1 if event E happens". If E does happen, then agents get \$1 for every share of the security they own; if E doesn't happen, they get nothing. The price of the security reflects the aggregation of agents' beliefs about the random event. The main goal of a prediction market is to extract an informative price for the security and thus an informative probability for the event. The *Iowa Electronic Market* and *Intrade* are two examples of real prediction markets with a long history of tested results [1; 2]. Chen and Pennock [7] discuss objectives for designing good prediction mechanisms and survey a number of proposed and fielded mechanisms.

In this paper, we focus on prediction markets with a central *market maker* that determines prices algorithmically based on a *cost function* [6]. At any time, the market maker will quote a price for any security; agents can decide to buy or sell shares at that price, or do nothing ("take it or leave it"). After each (infinitessimal) trade, the market maker updates the prices. For example, suppose there is a prediction market on a Duke basketball game, and the current price for the security "Duke wins" is \$ 0.8. If a risk-neutral agent believes that Duke will win with probability 0.9, then she has an incentive to buy some shares of the security, because her expected profit per share is $0.9 - 0.8 = 0.1$. If she buys some shares of the security, then its price will go up; if she sells some shares (equivalent to buying shares in Duke's opponent), then its price will go down. See Section 2 for more details.

A common cost function is Hanson's *logarithmic market scoring rule (LMSR)*, studied extensively in the literature [4; 5; 8; 13; 14], and used in many practical deployments including at Microsoft, Yahoo!, InklingMarkets, and Consensus Point. Pricing securities in LMSR-based prediction markets takes time that is polynomial in the number of outcomes. Therefore, it works well if the number of outcomes is not too large. However, in many situations the number of outcomes is exponentially large and has a combinatorial structure [4; 5; 10; 13; 14]. 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. Hence, the prediction market for this tournament has $2^{63} \approx 9.2 \times 10^{18}$ outcomes so computing the prices by directly using the cost function is infeasible. Pricing LMSR-based *combinatorial prediction markets* is #P-hard [4]. Chen et al. [5] show that using a Bayesian network to represent prices in a compact way, they can compute and update the prices for a restricted class of securities like "team A advances to round k".

Our contribution. In this paper, we take a Monte-Carlo approach to pricing LMSR-based combinatorial prediction markets for tournaments. Suppose a security is represented by a DNF formula F. Our main contribution is a Monte-Carlo algorithm (Algorithm 1) that is a fully polynomial randomized approximation scheme (FPRAS) for pricing F, under a reasonable assumption. Given any error rate $\epsilon > 0$, our algorithm outputs an estimation $\hat{I}$ of the market price $I(F)$ of F with the following guarantees: (1) $(1-\epsilon)I(F) \leq \hat{I} \leq (1+\epsilon)I(F)$ with at least $3/4$ probability; (2) the runtime of the algorithm is polynomial in $1/\epsilon$ and the size of F. Our algorithm is based on *importance sampling*, a well-known variance-reduction technique for Monte-Carlo methods [19]. As far as we know, our algorithm is the first Monte-Carlo algorithm for pricing prediction markets with a good theoretical guarantee.

Compared to Chen et al.'s [5] approach, ours works for a much larger class of securities, as we will show in Section 7. The tradeoff is the following two constraints. First, our algorithm returns an approximation of the price, and its runtime

[*]Part of this work was conducted at Yahoo! Research.

is determined by the error rate of the outcome and the size of the input. Second, the algorithm is an FPRAS only if we have a distribution that is a not-too-bad estimation for the true prices, and under this distribution it is easy to generate true valuations of the variables. Fortunately, for LMSR-based prediction markets for tournaments we can use the pairwise win rates between the teams to provide a reasonable estimation, as discussed in Section 4.

Other related work in AI. Pricing LMSR-based combinatorial prediction markets (see Section 2.2 for definitions) is a special case of a general version of the *weighted model counting problem*. In such a problem, we are given a logical formula F (not necessarily in DNF) and a weight $w(\vec{x})$ for each valuation $\vec{x}$. We are asked to compute $\sum_{\vec{x}:F(\vec{x})=1} w(\vec{x})$. However, in most weighted model counting problems, it is assumed that the weight function can be represented by the product of the individual weight functions, one for each variable [3; 17]. This is not the case in LMSR-based combinatorial prediction markets.

Another related problem is the *solution sampling problem* [9; 11; 12; 20], where the objective is to generate a valuation uniformly or nearly-uniformly from satisfying valuations. Our algorithm, on the other hand, generates a valuation according to a (not necessarily uniform or nearly-uniform) distribution. More importantly, we are not aware of any previous work in weighted model counting or solution sampling that is an FPRAS.

2 Preliminaries

2.1 LMSR-based Prediction Market

Let $\{1, \ldots, N\}$ denote the set of outcomes of a random variable X. A security "X will be i" means that holding each share of the security, the agent will receive \$1 from the market maker, if X turns out to be i. In this paper, we use a vector $\vec{q} \in \mathbb{R}^N$ to represent how many shares the market maker has sold for each security. That is, for every $i \leq N$, the market maker has sold $\vec{q}(i)$ shares of "X will be i". A cost function based prediction market is characterized by a *cost function* $C : \mathbb{R}^N \to \mathbb{R}$ and an initial quantity $\vec{q}_0$. The price for ϵ share of "X will be i" is the marginal cost of incrementing $\vec{q}$ by $\epsilon\vec{e}_i$ in C, where $\vec{e}_i$ is the N-dimensional vector whose ith component is 1 and the other components are 0. That is, if the agent wants to buy ϵ share of "X will be i", she must pay $C(\vec{q} + \epsilon\vec{e}_i) - C(\vec{q})$ to the market maker. The instantaneous price as $\epsilon \to 0$ is therefore $\partial C(\vec{q})/\partial\vec{q}(i)$.

In this paper, we study pricing prediction market with the cost function $C(\vec{q}) = b \log \sum_{i=1}^{N} e^{\vec{q}(i)/b}$, where the parameter b is called the *liquidity* of the market. This particular cost function corresponds to the logarithm market scoring rule, and we call this type of prediction markets *LMSR-based prediction markets*. The next equation computes the instantaneous price $I_{\vec{q}}(i)$ for the security "X will be i".

$$I_{\vec{q}}(i) = \frac{\partial C(\vec{q})}{\partial q_i} = \frac{e^{\vec{q}(i)/b}}{\sum_{j=1}^{N} e^{\vec{q}(j)/b}} \qquad (1)$$

2.2 Combinatorial Prediction Markets for Tournaments

A tournament of 2^m teams is represented by a binary tree of 2^m leaves, defined as follows. We note that in this paper the leaves are on the top of the tree (see Figure 1.)

Definition 1 *The tournament of 2^m teams is modeled by a binary tree composed of $2^m - 1$ binary variables as follows. For any $1 \leq j \leq m$, let $R_j = \{\mathbf{x}_1^j, \ldots, \mathbf{x}_{2^{m-j}}^j\}$ represent matches in round j. A variable $\mathbf{x}_i^1$ in R_1 represents the match between team T_{2i-1} and T_{2i}. For any i, j, the parents of $\mathbf{x}_i^j$ are $\mathbf{x}_{2i-1}^{j-1}$ and $\mathbf{x}_{2i}^{j-1}$. $\mathbf{x}_i^j$ takes 0 (respectively, 1) means that the $\mathbf{x}_{2i-1}^{j-1}$ (respectively, $\mathbf{x}_{2i}^{j-1}$) branch winner wins the match $\mathbf{x}_i^j$. The set of all variables is $X_m = R_1 \cup \ldots \cup R_m$. An outcome is uniquely characterized by a valuation of X_m.*[1]

In this paper, a security is represented by a logic formula F over X_m in *disjunctive normal form (DNF)*. That is, $F = C_1 \vee \cdots \vee C_k$, where for any $j \leq k$, $C_j = l_1^j \wedge \cdots l_{s_j}^j$, and l_i^j is either $\mathbf{x}$ or $\neg\mathbf{x}$ for some variable $\mathbf{x} \in X_M$. C_j is called a *clause* and l_i^j is called a *literal*. If F is satisfied under the outcome of the tournament (i.e., a valuation over X_m), then the market maker should pay the agent \$1 for each share of F the agent owns; otherwise the agent receives nothing.

Example 1 Figure 1 illustrates a tournament of four teams. $\mathbf{x}_1^1 = 0$ if T_1 beats T_2 in the first round; $\mathbf{x}_1^2 = 1$ if the winner of the match $\mathbf{x}_2^1$ beats the winner of the match $\mathbf{x}_1^1$. The security "T_2 is the champion" can be represented by the DNF formula $(\mathbf{x}_1^1 \wedge \neg\mathbf{x}_1^2)$. The valuation $(0_1^1, 1_2^1, 0_1^2)$ corresponds to the outcome where $\mathbf{x}_1^1 = 0$, $\mathbf{x}_2^1 = 1$, and $\mathbf{x}_1^2 = 0$, where T_1 is the champion. □

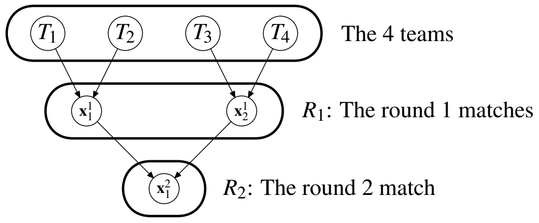

Figure 1: A tournament of four teams.

By definition, the price of F is the sum of the prices of the securities that correspond to the valuations under which F is satisfied. That is, $I_{\vec{q}}(F) = \sum_{\vec{v}:F(\vec{v})=1} I_{\vec{q}}(\vec{v}) = (\sum_{\vec{v}:F(\vec{v})=1} e^{\vec{q}(\vec{v})/b})/(\sum_{\vec{y}} e^{\vec{q}(\vec{y})/b})$. Let $N_{\vec{q}}(F) = \sum_{\vec{v}:F(\vec{v})=1} e^{\vec{q}(\vec{v})/b}$ and $D_{\vec{q}}(F) = \sum_{\vec{y}} e^{\vec{q}(\vec{y})/b}$. That is, $I_{\vec{q}}(F) = N_{\vec{q}}(F)/D_{\vec{q}}(F)$.

2.3 Importance Sampling

Importance sampling is a general variance-reduction technique for Monte-Carlo methods. Suppose we want to evaluate the expectation of a function $f : \{1, \ldots, N\} \to \mathbb{R}$ when the variable is chosen from a probability distribution π over $\{1, \ldots, N\}$. 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 $X_1, \ldots, X_Z$ 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.

[1]We note that there are $2^m - 1$ variables, so that the input size is polynomial in 2^m.

Lemma 1 (Follows from Chebyshev's inequality) *Let $H_1, \ldots, H_Z$ 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,*

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

Importance sampling reduces the variance by generating the outcomes that have higher f values more often. Suppose we have another distribution $\overline{\pi}$ such that for every outcome i, $\overline{\pi}(i) = 0 \implies f(i)\pi(i) = 0$. We can then use $\overline{\pi}$ to provide an unbiased estimator for $E[f;\pi]$ as follows. Let H denote the random variable that takes $\frac{f(i)\pi(i)}{\overline{\pi}(i)}$ with probability $\overline{\pi}(i)$. 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 also unbiased, and $\text{Var}(H)/E[H]^2$ might be significantly smaller than $\text{Var}(f)/E[f;\pi]^2$.

A good $\overline{\pi}$ can greatly reduce the variance, therefore in turn boost the Monte-Carlo method. The best scenario is that for any outcome i, $\overline{\pi}(i)$ is proportional to $f(i)\pi(i)$. Then, the variance becomes 0 and we only need 1 sample. For any pair of functions f and g defined over the same domain D, let $f \cdot g$ denote the function such that for any value $x \in D$, $(f \cdot g)(x) = f(x) \times g(x)$.

Definition 2 *For any $c > 0$, we say that a probability distribution $\overline{\pi}$ is a* c-approximation *to $f \cdot \pi$, if there exists a constant d such that for any outcome i, $d\frac{1}{c}f(i)\pi(i) \leq \overline{\pi}(i) \leq dcf(i)\pi(i)$.*

That is, $\overline{\pi}$ is a c-approximation to $f \cdot \pi$ if for every $i \leq N$, $\overline{\pi}(i)$ is approximately proportional to $f(i)\pi(i)$, up to a multiplicative factor c. In this case it is easy to check that $\text{Var}(H)/(E[H])^2 \leq c^4$, where H is the random variable that takes $\frac{f(i)\pi(i)}{\overline{\pi}(i)}$ with probability $\overline{\pi}(i)$. Hence, we have the following lemma, which follows directly from Lemma 1.

Lemma 2 *Suppose $\overline{\pi}$ is a c-approximation to $f \cdot \pi$. Let H^Z denote the estimator calculated by applying importance sampling to $f \cdot \pi$ using $\overline{\pi}$ for Z iterations. If $Z \geq 4c^4/\epsilon^2$, then,*

$$Pr(|H^Z - \mu| < \epsilon\mu) \geq 3/4$$

2.4 An FPRAS for # DNF

An algorithm A is an 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 $3/4$,[2] (2) the runtime of A is polynomial in $1/\epsilon$ and the size of x.

To motivate our algorithm, we recall an FPRAS for the #DNF problem by Karp, Luby, and Madras [15] (KLM for short). The #DNF problem has been proven to be #P-complete [18]. In a #DNF instance, we are given a DNF formula $F = C_1 \vee \cdots \vee C_k$ over $\{\mathbf{x}_1, \ldots, \mathbf{x}_t\}$, 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 $2^t \cdot E[F;\pi_u]$.

Let us first explain why a naïve Monte-Carlo method does not work. The naïve Monte-Carlo method generates Z valuations i.i.d. uniformly at random, and counts how many times F is satisfied, denoted by X^Z. Clearly $2^t \cdot X^Z/Z$ 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, it might take a long time for the naïve Monte-Carlo method to converge (Lemma 1). For example, if there is only one valuation that satisfies F, then the variance of X^Z/Z is approximately $1/2^t$, and the expectation of X^Z/Z is $1/2^t$, which means that $\text{Var}(X^Z/Z)/E[X^Z/Z]^2$ is approximately 2^t.

In the KLM algorithm, only the valuations under which $F = 1$ are generated. We next show a slight variant of the KLM algorithm using the uniform distribution π_u, in order to better explain its importance sampling nature under the c-approximation argument, as well as to show its connection with our algorithm. For any clause C_j, let S_j denote the set of valuations that satisfy C_j. The algorithm has three steps in each iteration. (1) Choose a clause C_j with probability $\pi_u(S_j)/(\sum_{j'}\pi_u(S_{j'}))$; (2) then choose a valuation $\vec{v}$ from S_j with probability $\pi_u(\vec{v}|S_j)$; (3) finally, compute the number of clauses $\vec{v}$ satisfies, denoted by $n(\vec{v})$, then add $(F(\vec{v})\sum_j \pi_u(S_j))/(2^t\pi_u(\vec{v})n(\vec{v}))$ to a counter K. Given a error rate $\epsilon > 0$, let $Z = 4k^4/\epsilon^2$. After Z iterations, the algorithm outputs $2^tK/Z$. Let π' denote the distribution induced by this sampling process. That is, for any valuation $\vec{v}$ with $F(\vec{v}) = 1$, $\pi'(\vec{v}) = n(\vec{v})/(\sum_{j'}|S_{j'}|)$. We note that for any $\vec{v}$ with $F(\vec{v}) = 1$, $1/k \leq 1/n(\vec{v}) \leq 1$. Therefore, π' is a k-approximation to $F \cdot \pi_u$. By Lemma 2, this algorithm is an FPRAS for the #DNF problem.

3 Overview of Our Algorithm

In this section, we explain the main ideas behind our algorithm (Algorithm 1). Details of the sub-procedures (steps 1,4, and 7) will be discussed in later sections.

Let F denote a DNF formula over X_m. For the same reason as in the #DNF problems, the naïve sampling approach (that generates valuations i.i.d. uniformly at random) might not work well for either $N_{\vec{q}}(F)$ or $D_{\vec{q}}(F)$. Therefore, we employ Monte-Carlo techniques for $D_{\vec{q}}(F)$ and $N_{\vec{q}}(F)$ respectively. For now, suppose we have a distribution $\overline{\pi}$ (which will be specified in Section 4) such that (1) $\overline{\pi}$ is a c-approximation to $f_{\vec{q}} \cdot \pi_u$, where c is a constant,[3] and (2) there is a polynomial-time algorithm that samples a valuation according to $\overline{\pi}$.

Our Monte-Carlo technique for $D_{\vec{q}}(F)$ is straightforward. Given $\vec{q}$, let $f_{\vec{q}}$ be such that for any valuation $\vec{v}$, $f_{\vec{q}}(\vec{v}) = e^{\vec{q}(\vec{v})/b}$. Then, $D_{\vec{q}}(F) = 2^{2^m} \cdot E[f_{\vec{q}};\pi_u]$ (we recall that π_u is the uniform distribution). We adopt the standard importance sampling technique to estimate $E[f_{\vec{q}};\pi]$ using $\overline{\pi}$.

Our Monte-Carlo technique for $N_{\vec{q}}(F)$ is more complicated. We note that $N_{\vec{q}}(F) = 2^{2^m}E[F \cdot f_{\vec{q}};\pi_u]$. Therefore, $\overline{\pi}$ might not be a c-approximation for $F \cdot f_{\vec{q}} \cdot \pi_u$. Our algorithm adopts the idea of the KLM algorithm by substituting the uniform distribution π_u for $\overline{\pi}$, and substituting $F(\vec{v})$ for $F(\vec{v})f_{\vec{q}}(\vec{v})$ in the description for KLM. Again, let S_j denote the set of all valuations that satisfy C_j. Our algorithm also has three steps in each iteration: (1) Choose a clause C_j

[2] By using the *median of means* method, for any $\delta < 1$, the successful 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 [19]).

[3] Equivalently, $\overline{\pi}$ is a c-approximation to $f_{\vec{q}}$ or $I_{\vec{q}}$.

with probability $\overline{\pi}(S_j)/\sum_{j'}\overline{\pi}(S_{j'})$; (2) then choose a valuation $\vec{v}$ from S_j with probability $\overline{\pi}(\vec{v}|S_j)$; (3) finally, compute the number of clauses that $\vec{v}$ satisfies, denoted by $n(\vec{v})$, then add $F(\vec{v})f_{\vec{q}}(\vec{v})\sum_{j'}\overline{\pi}(S_{j'})/(2^{2^m}\overline{\pi}(\vec{v})n(\vec{v}))$ to a counter N. Given a error rate $\epsilon > 0$, let $Z = 4c^4k^4/\epsilon^2$. After Z iterations, the algorithm returns an estimator $\hat{N} = 2^{2^m}N/Z$.

We note that so far many technical difficulties remain unsolved. For example, we have not specified how to compute $\overline{\pi}(S_j)$ efficiently (in contrast, in KLM computing $\pi_u(S_j)$ is easy—it is exactly $|S_j|/2^t$.) We will address all the technical difficulties in later sections.[4] The framework of the algorithm is illustrated in Algorithm 1, which computes an estimation $\hat{N}$ for $N_{\vec{q}}(F)$ and an estimation $\hat{D}$ for $D_{\vec{q}}(F)$.

Algorithm 1: ApproximatePricing

```
   Input: $\overline{\pi}$, $\vec{q}$, $\epsilon$, a DNF formula $F = C_1 \vee \cdots \vee C_k$.
   Output: Estimations for $N_{\vec{q}}(F)$ and $D_{\vec{q}}(F)$.
 1 For each $j \le k$, let $S_j = \{\vec{v} : C_j(\vec{v}) = 1\}$. Compute
   $G = \sum_{j'}\overline{\pi}(S_{j'})$. (Details in Algorithm 3 in Section 5.)
 2 for $i = 1$ to $Z = 4c^4k^4/\epsilon^2$ do
 3     Choose an index $j$ with probability $p_j = \frac{\overline{\pi}(S_j)}{G}$.
 4     Choose an assignment $\vec{v}$ from $S_j$ with probability
       $\overline{\pi}(\vec{v}|S_j)$. (Details in Algorithm 4 in Section 6.)
 5     Compute $n(\vec{v}) = |\{j' : C_{j'}(\vec{v}) = 1\}|$.
 6     Let $N \leftarrow N + \frac{e^{q(\vec{v})/b}G}{2^{2^m}\overline{\pi}(\vec{v})n(\vec{v})}$.
 7     Choose an assignment $\vec{w}$ with probability $\overline{\pi}(\vec{w})$.
       (Details in Algorithm 2 in Section 4.)
 8     Let $D \leftarrow D + \frac{e^{q(\vec{w})/b}G}{2^{2^m}\overline{\pi}(\vec{w})}$.
 9 end
10 return $\hat{N} = 2^{2^m}N/Z$ and $\hat{D} = 2^{2^m}D/Z$.
```

Theorem 1 *If $\overline{\pi}$ is a c-approximation to $I_{\vec{q}}$ for some constant c, and step 1, 4 and 7 in Algorithm 1 take polynomial time, then Algorithm 1 is an unbiased FPRAS for both $N_{\vec{q}}(F)$ and $D_{\vec{q}}(F)$; and if we let the output of Algorithm 1 be N/D, then, Algorithm 1 is an FPRAS for $I_{\vec{q}}(F)$.*

Due to space constraints, proofs are omitted. A full version with all proofs is available on the first author's website.

For any security F' represented by a CNF formula, we can first use De Mongan's Law to compute its negation F, which is in DNF. Then, we apply Algorithm 1 to compute an approximation $\hat{p}$ for $I_{\vec{q}}(F)$. Because $I_{\vec{q}}(F') + I_{\vec{q}}(F) = 1$ and with a high probability $|\hat{p} - I_{\vec{q}}(F)| \le \epsilon I_{\vec{q}}(F) \le \epsilon$, we have that with a high probability $|1 - \hat{p} - I_{\vec{q}}(F')| < \epsilon$. This shows that Algorithm 1 can be used to compute the prices of securities represented by CNF formulas with additive error bounds.

[4]One of the anonymous reviewers insightfully pointed out that, under the framework studied by Chen et al. [5], there are close connections between our Algorithm 3 and the well-known belief propagation algorithm [16], and between our Algorithm 4 and the back-propagation phase of the belief propagation algorithm.

4 The Distribution Based on Pairwise Win Rates between Teams

We now specify the distribution $\overline{\pi}$ used in Algorithm 1. For any pair of teams T_i and T_j, we let $h(T_i, T_j)$ denote the win rate of T_i against T_j in the history, that is, the number of matches where T_i beats T_j over the total number of matches between them.[5] By definition, $h(T_i, T_j) + h(T_j, T_i) = 1$. $h(\cdot,\cdot)$ is called a *pairwise win rate function*.

Algorithm 2 randomly generates a valuation $\vec{v}$ by using $h(\cdot,\cdot)$. Therefore, it defines a distribution π_h over all valuations, which is used as the input $\overline{\pi}$ in Algorithm 1. Algorithm 2 is also used in step 7 in Algorithm 1 to generate a random valuation according to π_h. The idea behind Algorithm 2 is to keep track of the remaining teams and simulate each match in the tournament using $h(\cdot,\cdot)$. This is done in a up-down flavor (see Figure 1).

Algorithm 2: ValuationSampling

```
  Input: $h(\cdot,\cdot)$.
  Output: A randomly generated valuation $\vec{v}$.
1 For each $i \le 2^m$, associate $\mathbf{x}_i^0$ with $T_i$.
2 for $j = 1$ to $m$ do
3     for $i = 1$ to $2^{m-j}$ do
4         Let $l$ (respectively, $r$) denote the team number
          associated with $\mathbf{x}_{2i-1}^{j-1}$ (respectively, $\mathbf{x}_{2i}^{j-1}$).
5         With probability $h(T_l, T_r)$ let $\mathbf{x}_i^j = 0$ and
          associate $\mathbf{x}_l^j$ with $T_l$; with probability $h(T_r, T_l)$
          let $\mathbf{x}_i^j = 1$ and associate $\mathbf{x}_i^j$ with $T_r$.
6     end
7 end
8 return the values of $(\mathbf{x}_1^1, \mathbf{x}_2^1, \ldots, \mathbf{x}_1^m)$.
```

Since $h(\cdot,\cdot)$ is calculated from historical data, it is common knowledge to every agent. Therefore, it makes sense to assume that $h(T_i, T_j)$ is a reasonable approximation to the agents' belief about the probability that T_i beats T_j if they meet in the current tournament. Of course there are many other factors that might affect the agents' belief. For example, suppose in the previous round T_i beat a strong team and T_j beat a weak team, then the agents' belief about the probability that T_i beats T_j in the current tournament might be smaller than $h(T_i, T_j)$. However, such bias is usually small. Therefore, **we assume that π_h is a not-too-bad approximation (that is, a c-approximation for some constant c) to $I_{\vec{q}}$.** More importantly, as we will show later, π_h is computationally tractable for steps 1 and 4 in Algorithm 1.

Example 2 Suppose there are four teams. Let $h(\cdot,\cdot)$ be defined as follows. $h(T_1, T_2) = h(T_3, T_4) = 0.5$, $h(T_3, T_1) = h(T_4, T_1) = 0.9$, $h(T_3, T_2) = h(T_4, T_2) = 0.3$. In Algorithm 2, we first sample the value of $\mathbf{x}_1^1$. The probability that $\mathbf{x}_1^1 = 0$ is $h(T_1, T_2) = 0.5$. Suppose the outcome is $\mathbf{x}_1^1 = 0$. We then sample the value of $\mathbf{x}_2^1$ using $h(T_3, T_4)$, and suppose that $\mathbf{x}_2^1 = 1$. Finally, we sample the value of $\mathbf{x}_2^1$ using

[5]In case of insufficient historical data, we can use relative strength of the teams to estimate $h(\cdot,\cdot)$.

$h(T_1, T_4) = 0.1$, because T_1 is the winner of $\mathbf{x}_1^1$ and T_4 is the winner of $\mathbf{x}_2^1$. Suppose the outcome is $\mathbf{x}_1^2 = 1$. Then, Algorithm 2 will output $(0_1^1, 1_2^1, 1_1^2)$. □

5 Computing Marginal Probabilities

Let Y denote a subset of variables and let $\vec{v}_Y$ denote a valuation of the variables in Y. In this section we propose a polynomial-time algorithm (Algorithm 3) that computes any marginal probability under π_h. Algorithm 3 is used in step 1 of Algorithm 1 in the following way. For any $j \le k$, let $Y = \{\mathbf{x} : C_j$ contains $\mathbf{x}$ or $\neg\mathbf{x}\}$, and for any $\mathbf{x} \in Y$, if C_j contains $\mathbf{x}$, then let $\vec{v}_Y(\mathbf{x}) = 1$, if C_j contains $\neg\mathbf{x}$, then let $\vec{v}_Y(\mathbf{x}) = 0$ (we assume that C_j does not contain both $\mathbf{x}$ and $\neg\mathbf{x}$). We have that $\pi_h(S_j) = \pi_h(\vec{v}_Y)$.

For any Y and $\vec{v}_Y$, we say that the value of a variable $\mathbf{x}_i^j$ is consistent with $\vec{v}_Y$, if whenever $\mathbf{x}_i^j \in Y$, $\mathbf{x}_i^j$'s value must be the same as in $\vec{v}_Y$ (if $\mathbf{x}_i^j \notin Y$, then its value is always consistent with $\vec{v}_Y$).

Algorithm 3 is based on dynamic programming. For each variable $\mathbf{x}_i^j$, we compute a table that records the probabilities for each team T that can reach $\mathbf{x}_i^j$ to actually win $\mathbf{x}_i^j$, when the values of all ancestors of $\mathbf{x}_i^j$ (including $\mathbf{x}_i^j$) are consistent with $\vec{v}_Y$. Let $\text{MP}_i^j(T)$ denote such a probability for T at $\mathbf{x}_i^j$. Once we have MP_{2i-1}^{j-1} and MP_{2i}^{j-1}, we can compute MP_i^j by using $h(\cdot,\cdot)$ and $\vec{v}_Y$. In other words, MP_i^j's are computed in a up-down flavor (see Figure 1). We note that for each variable we will *not* compute a probability distribution conditioned on $\vec{v}_Y$. That is, $\sum_T \text{MP}_i^j(T)$ might be strictly smaller than 1.

For any j such that $1 \le j \le m$ and any $i \le 2^{m-j}$, let $\text{LG}(\mathbf{x}_i^j)$ (respectively, $\text{RG}(\mathbf{x}_i^j)$) denote the set of teams in the left (respectively, right) branch of the parents of $\mathbf{x}_i^j$. For example, $\text{LG}(\mathbf{x}_1^1) = \{T_1\}$, $\text{RG}(\mathbf{x}_1^2) = \{T_3, T_4\}$. For any $i \le 2^m$, let $M_i^0(T_i) = 1$ and for any $i' \neq i$, let $M_i^0(T_{i'}) = 0$. MP_i^j will be updated according to the following rule.

Rule 1 *For any $1 \le j \le m$, any $i \le 2^{m-j}$, and any team T, define $M_i^j(T)$ as follows.*

(a) If $T \in LG(\mathbf{x}_i^j)$ and either $\mathbf{x}_i^j \notin Y$, or $\vec{v}_Y(\mathbf{x}_i^j) = 0$, then let $MP_i^j(T) = \sum_{T' \in RG(\mathbf{x}_i^j)} h(T, T') MP_{2i-1}^{j-1}(T) MP_{2i}^{j-1}(T')$. That is, if it is not determined in $\vec{v}_Y$ that the winner comes from the right branch $\mathbf{x}_{2i}^{j-1}$, then we count in the situations where a team T from the left branch $\mathbf{x}_{2i-1}^{j-1}$ (which happens with probability $MP_{2i-1}^{j-1}(T)$) wins the match $\mathbf{x}_i^j$. For each T from the left branch, we enumerate all potential competitors that come from the right branch.

(b) If $T \in RG(\mathbf{x}_i^j)$ and either $\mathbf{x}_i^j \notin Y$, or $\vec{v}_Y(\mathbf{x}_i^j) = 1$, then let $MP_i^j(T) = \sum_{T' \in LG(\mathbf{x}_i^j)} h(T, T') MP_{2i-1}^{j-1}(T') MP_{2i}^{j-1}(T)$.

(c) Otherwise let $MP_i^j(T) = 0$.

Proposition 1 *Algorithm 3 computes the marginal probability $\pi_h(\vec{v}_Y)$ in polynomial time.*

Example 3 Suppose there are four teams and $h(\cdot,\cdot)$ is defined the same as in Example 2. Suppose $Y = \{\mathbf{x}_2^1, \mathbf{x}_1^2\}$ and $\vec{v}_Y = (1_2^1, 1_1^2)$. Algorithm 3 computes $\pi_h(\vec{v}_Y)$ as follows. In the first round Rule 1(a) applies to $\text{MP}_1^1(T_1)$, so that $\text{MP}_1^1(T_1) = h(T_1, T_2)\text{MP}_1^0(T_1)\text{MP}_2^0(T_2) = 0.5$; Rule 1(b) applies to $\text{MP}_1^1(T_2)$, so that $\text{MP}_1^1(T_2) = h(T_2, T_1)\text{MP}_2^0(T_2)\text{MP}_1^0(T_1) = 0.5$. Because $\mathbf{x}_2^1 \in Y$ and $\vec{v}_Y(\mathbf{x}_2^1) = 1$, Rule 1(c) applies to $\text{MP}_2^1(T_3)$ and Rule 1(b) applies to $\text{MP}_2^1(T_4)$, so that $\text{MP}_2^1(T_3) = 0$ and $\text{MP}_2^1(T_4) = 0.5$. We note that $\text{MP}_2^1(T_3) + \text{MP}_2^1(T_4) = 0.5 < 1$.

Algorithm 3: CompMarginal

```
Input: $h(\cdot,\cdot)$, a valuation $\vec{v}_Y$ of a set of variables $Y$.
Output: $\pi_h(\vec{v}_Y)$.
1 for $j = 1$ to $m$ do
2     for $i = 1$ to $2^{m-j}$ do
3         Compute $\text{MP}_i^j$ according to Rule 1.
4     end
5 end
6 return $\sum_l \text{MP}_1^m(T_l)$.
```

Now we compute MP_1^2. Because $\mathbf{x}_1^2 \in Y$ and $\vec{v}_Y(\mathbf{x}_1^2) = 1$, Rule 1(c) applies to $\text{MP}_1^2(T_1)$ and $\text{MP}_1^2(T_2)$, and Rule 1(b) applies to $\text{MP}_1^2(T_3)$ and $\text{MP}_1^2(T_4)$. Therefore, $\text{MP}_1^2(T_1) = \text{MP}_1^2(T_2) = \text{MP}_1^2(T_3) = 0$ and $\text{MP}_1^2(T_4) = h(T_4, T_1)\text{MP}_2^1(T_4)\text{MP}_1^1(T_1) + h(T_4, T_2)\text{MP}_2^1(T_4)\text{MP}_1^1(T_2) = 0.9 \times 0.5 \times 0.5 + 0.3 \times 0.5 \times 0.5 = 0.3$. The output of Algorithm 3 is $0 + 0 + 0 + 0.3 = 0.3$.

On the other hand, $\pi_h(\vec{v}_Y)$ can be computed directly. We have $\pi_h(\mathbf{x}_1^1 = 0, \mathbf{x}_2^1 = 1, \mathbf{x}_1^2 = 1) = 0.5 \times 0.5 \times 0.9 = 9/40$ and $\pi_h(\mathbf{x}_1^1 = 1, \mathbf{x}_2^1 = 1, \mathbf{x}_1^2 = 1) = 0.5 \times 0.5 \times 0.3 = 3/40$. Therefore, $\pi_h(\vec{v}_Y) = 12/40 = 0.3$, which is the same as the output of Algorithm 3. □

6 Sampling a Valuation from S_j

In this section we present an algorithm (Algorithm 4) that randomly samples a valuation $\vec{v}$ from S_j according to the conditional probability $\pi_h(\vec{v}|S_j)$. This algorithm is used in step 4 in Algorithm 1. Algorithm 4 samples the outcome in a bottom-up flavor. Instead of sampling the value of each variable one after another, our algorithm samples the *winner* at each variable (i.e., match) sequentially. In the first step, we pin down all variables in Y to be $\vec{v}_Y$, and sample the winner T^* for the whole tournament (equivalently, for the last match $\mathbf{x}_1^m$) proportional to the marginal probability $\text{MP}_1^m(T^*)$ computed in Algorithm 3 (we note that $\text{MP}_1^m(T^*)$'s do not necessarily sum up to 1). Then, we pin down $\mathbf{x}_1^m$ and any other variables that are necessary to make sure that T^* wins the tournament. For example, without loss of generality let $T^* = T_1$. We enforce that $\mathbf{x}_1^m = \mathbf{x}_1^{m-1} = \cdots = \mathbf{x}_1^1 = 0$. Suppose $\mathbf{x}_2^{m-1} \notin Y$. The second step is to sample the winner T' for $\mathbf{x}_2^{m-1}$, with a probability that is proportional to $h(T_1, T')\text{MP}_2^{m-1}(T')$. Generally, in each step we find an undetermined variable $\mathbf{x}$ that is as close to the final round as possible, sample the winner T for $\mathbf{x}$, and then pin down the values of all variables that are necessary for T to win at $\mathbf{x}$. The algorithm terminates after all variables in X_m are determined. Formally, the algorithm is as follows.

Proposition 2 *For any valuation $\vec{v}$ that is consistent with $\vec{v}_Y$, Algorithm 4 runs in polynomial time and returns $\vec{v}$ with probability $\pi_h(\vec{v}|\vec{v}_Y)$.*

Algorithm 4: Sampling

```
   Input: $h(\cdot,\cdot)$, a valuation $\vec{v}_Y$ for a set of variables $Y$.
   Output: A valuation $\vec{v}$ with probability $\pi_h(\vec{v}|\vec{v}_Y)$.
1  For each $j \le m$ and each $i \le 2^{m-j}$, use Algorithm 3 to
   compute $\text{MP}_i^j$.
2  Assign $\vec{v}_Y$ to the variables in $Y$.
3  Sample the winner $T^*$ for $\mathbf{x}_1^m$, with a probability
   proportional to $\text{MP}_1^m(T^*)$.
4  Assign values to $\mathbf{x}_1^m$ and the smallest set of its ancestors
   that are necessary for $T^*$ to win at $\mathbf{x}_1^m$.
5  while $\exists$ an unassigned variable do
6      Find an unassigned variable $\mathbf{x}_i^j$ with the largest $j$.
7      Sample the winner $T$ for $\mathbf{x}_i^j$, with a probability that is
       proportional to $h(T',T)\text{MP}_i^j(T)$, where $T'$ is the
       winner at $\mathbf{x}_{\lceil i/2\rceil}^{j+1}$.
8      Assign values to $\mathbf{x}_i^j$ and the smallest set of its
       ancestors that are necessary for $T$ to win at $\mathbf{x}_i^j$.
9  end
10 return the valuation.
```

7 Common Bidding Languages in DNF

Chen et al. have shown that their method can efficiently price some popular securities, including "Team T wins game $\mathbf{x}$" (Theorem 3.1 [5]), and "Team T wins game $\mathbf{x}$ and Team T' wins game $\mathbf{x}'$, where there is an edge between $\mathbf{x}$ and $\mathbf{x}'$" (Theorem 3.2 [5]). The next proposition provides a far-from-complete list of popular securities that can be modeled by DNF formulas of polynomial size, including the two types of securities considered by Chen et al.

Proposition 3 *The following types of securities can be modeled by DNF formulas of polynomial size.*

- *Team T wins game $\mathbf{x}$.*
- *Team T wins game $\mathbf{x}$ and Team T' wins game $\mathbf{x}'$.*
- *Team T advances further than T'.*
- *The champion is among $\{T, T', T^*\}$.*
- *Team T will meet T' in the tournament.*

Yahoo! operated a combinatorial prediction market game called Predictalot for NCAA basketball (2010, 2011), the FIFA World Cup (2010), and (under the name Predictopus) the Cricket World Cup (2011). Many, but not all, of the predictions people placed can be represented by polynomial-size DNF formulas. Some securities likely cannot be represented compactly by DNF formulas, for example, "at least half of the 1st-round favorites will reach the 2nd round".

8 Future Work

We plan to implement our algorithm in a real combinatorial prediction market like Predictalot. There are many open questions. Is there a better distribution $\overline{\pi}$ (compared to π_h) that can be used in Algorithm 1? Can we extend our algorithm to other types of combinatorial prediction markets? Can we design a Monte-Carlo algorithm for pricing other types of securities? Is there any computationally efficient way to update the prices as the results of some matches come out?

Acknowledgements

Lirong Xia acknowledges a James B. Duke Fellowship and Vincent Conitzer's NSF CAREER 0953756 and IIS-0812113, and an Alfred P. Sloan fellowship for support. We thank all IJCAI-11 reviewers for helpful suggestions and comments, especially the anonymous reviewer who pointed out the two insightful connections discussed in footnote 4.

References

[1] Joyce E. Berg, Robert Forsythe, Forrest D. Nelson, and Thomas A. Rietz. Results from a dozen years of election futures markets research. *The Handbook of Experimental Economics Results*, 1:742–751, 2008.

[2] Joyce E. Berg, Forrest D. Nelson, and Thomas A. Rietz. Prediction market accuracy in the long run. *International Journal of Forecasting*, 24:285–300, 2008.

[3] Mark Chavira and Adnan Darwiche. On probabilistic inference by weighted model counting. *Artif. Intell.*, 172:772–799, 2008.

[4] Yiling Chen, Lance Fortnow, Nicolas Lambert, David M. Pennock, and Jennifer Wortman. Complexity of combinatorial market makers. In *Proc. EC*, pages 190–199, 2008.

[5] Yiling Chen, Sharad Goel, and David M. Pennock. Pricing combinatorial markets for tournaments. In *Proc. STOC*, pages 305–314, 2008.

[6] Yiling Chen and David M. Pennock. A utility framework for bounded-loss market makers. In *Proc. UAI*, pages 349–358, 2007.

[7] Yiling Chen and David M. Pennock. Designing markets for prediction. *AI Magazine*, 31(4):42–52, 2010.

[8] Yiling Chen and Jennifer Wortman Vaughan. A new understanding of prediction markets via no-regret learning. In *Proc. EC*, pages 189–198, 2010.

[9] Rina Dechter, Kalev Kask, Eyal Bin, and Roy Emek. Generating random solutions for constraint satisfaction problems. In *Proc. AAAI*, pages 15–21, 2002.

[10] Lance Fortnow, Joe Kilian, David M. Pennock, and Michael P. Wellman. Betting Boolean-style: A framework for trading in securities based on logical formulas. *Decision Support Systems*, 39(1):87–104, 2004.

[11] Vibhav Gogate and Rina Dechter. Studies in solution sampling. In *Proc. AAAI*, pages 271–276, 2008.

[12] Carla P. Gomes, Ashish Sabharwal, and Bart Selman. Near-uniform sampling of combinatorial spaces using XOR constraints. In *Proc. NIPS*, pages 481–488, 2006.

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

[14] Robin Hanson. Logarithmic market scoring rules for modular combinatorial information aggregation. *Journal of Prediction Markets*, 1:3–15, February 2007.

[15] Richard M. Karp, Michael Luby, and Neal Madras. Monte-Carlo approximation algorithms for enumeration problems. *J. Algorithms*, 10:429–448, 1989.

[16] Judea Pearl. *Probabilistic reasoning in intelligent systems: Networks of plausible inference*. Morgan Kaufmann Publishers Inc., 1988.

[17] Tian Sang, Paul Bearne, and Henry Kautz. Performing Bayesian inference by weighted model counting. In *Proc. AAAI*, pages 475–481, 2005.

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

[19] Vijay Vazirani. *Approximation Algorithms*. Springer Verlag, 2001.

[20] Wei Wei, Jordan Erenrich, and Bart Selman. Towards efficient sampling: exploiting random walk strategies. In *Proc. AAAI*, pages 670–676, 2004.