Combinatorial Prediction Markets for Event Hierarchies

Mingyu Guo*
Duke University
Department of Computer Science
Durham, NC, USA
mingyu@cs.duke.edu

David M. Pennock
Yahoo! Research
111 W. 40th St. 17th Floor
New York, NY 10018
pennockd@yahoo-inc.com

ABSTRACT

We study combinatorial prediction markets where agents bet on the sum of values at any tree node in a hierarchy of events, for example the sum of page views among all the children within a web subdomain. We propose three expressive betting languages that seem natural, and analyze the complexity of pricing using Hanson's logarithmic market scoring rule (LMSR) market maker. *Sum of arbitrary subset (SAS)* allows agents to bet on the weighted sum of an arbitrary subset of values. *Sum with varying weights (SVW)* allows agents to set their own weights in their bets but restricts them to only bet on subsets that correspond to tree nodes in a fixed hierarchy. We show that LMSR pricing is NP-hard for both SAS and SVW. *Sum with predefined weights (SPW)* also restricts bets to nodes in a hierarchy, but using predefined weights. We derive a polynomial time pricing algorithm for SPW. We discuss the algorithm's generalization to other betting contexts, including betting on maximum/minimum and betting on the product of binary values. Finally, we describe a prototype we built to predict web site page views and discuss the implementation issues that arose.

Categories and Subject Descriptors

J.4 [**Computer Applications**]: Social and Behavioral Sciences—*Economics*

General Terms

Economics, Theory

Keywords

Combinatorial prediction markets, logarithmic market scoring rule market maker, computational complexity

1. INTRODUCTION

Prediction markets are powerful mechanisms for eliciting probability estimates of future events. The markets' assessment can be remarkably accurate [12, 13]. The Iowa Electronic Markets (IEM), a real-money based prediction market maintained by the University of Iowa, significantly outperforms the traditional polls over the past five US presidential elections [1, 5, 6, 10]. The Foresight Exchange (http://www.ideosphere.com) and Hollywood Stock Exchange (www.hsx.com), two play-money based prediction markets, have been successful in predicting the probabilities of various events on a large range of topics, such as unresolved scientific questions and Oscar award winners [11].

In a typical prediction market, agents trade securities with each other or with a central market maker. An example security would be "$1 on Duke to win the 2009 NCAA men's basketball championship". Such a security pays off $1 if Duke indeed wins the title, and $0 otherwise. If the market price for this security is $0.2, then it means that the consensus estimated probability of Duke winning the title is 20 percent at the time of the quote. The market price changes along with the trading activities: more demand results in higher price and vise versa.

From Las Vegas to Wall Street, nearly all operating prediction markets are single dimensional. That is, securities of different kinds are being traded in separate markets, even if they may be logically related. For example, the price of one security of Duke winning the 2009 NCAA men's basketball championship should be related to the price of another security of Duke getting into the Final Four. When related securities are handled separately, estimate discrepancies and undesirable arbitrage opportunities arise. To capture the underlying relationship among different securities, we need a combinatorial market in which all allowable securities are being traded, and the market must maintain a consistent set of prices for all the securities.

Let us consider a combinatorial prediction market on the state-by-state results of the US presidential election. The outcome space for such a combinatorial market is extremely large (2^{51}, considering District of Columbia). Low liquidity becomes a problem, as the agents' attention gets divided among exponentially many outcomes. Generalization of standard double auctions may simply fail to find any trades [7, 3]. A better idea is to implement a market maker that is willing to buy or sell any security at any time. When an agent comes in, she asks for a quote of the security of her interest. If the quoted price is lower than the price in the agent's mind, the agent can start buying the security, until the price grows to a point that is close to the agent's estimation. On the other hand, if the price is considered too high, the agent can start selling the security (equivalent to buying the negation of the event). When the market gets stable, the market prices reflect the agents' consensus probability estimations.

In this paper, we will be focusing on a specific type of combinatorial market maker – Hanson's logarithmic market scoring rule market maker (LMSR) [8, 9]. A prediction market based on LMSR requires only bounded subsidy and it has in some sense infinite

*This material is based on work done primarily while Guo was visiting Yahoo! Research. While at Duke, Guo is supported under NSF IIS-0812113.

Cite as: Combinatorial Prediction Markets for Event Hierarchies, Mingyu Guo, David M. Pennock, *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009),* Decker, Sichman, Sierra and Castelfranchi (eds.), May, 10–15, 2009, Budapest, Hungary, pp. 201–208

liquidity. LMSR is becoming the standard market maker for combinatorial setting, and it has been the subject of interest in a number of research papers [4].

In principle, agents should be allowed to bet on (buy/sell securities of) any event (subset of outcomes). However, when the outcome space is large, pricing (computing the exact "price quote") may take exponential time for some securities. Therefore, as a trade off, we can restrict the set of events that are allowed to be bet on, usually through restricting the betting language. Several papers examine the balance between expressiveness and computational complexity [7, 4, 2, 3]. This line of research was initiated by Fortnow et al. [7], followed by Chen et al. [3], in which the authors study betting languages on Boolean combinatorics and permutations for market clearing problems. In Chen et al. [2], the authors analyze the computational complexity of LMSR pricing for permutations and Boolean combinatorics. The authors show that for subset betting, pair betting, and betting on conjunctions and disjunctions, pricing for LMSR market maker is #P-hard. The work closest to our own is that of [4]. The authors study a special case of Boolean combinatorics in which the agents bet on how far a team goes in a single-elimination tournament. They propose a polynomial-time algorithm for the problem of LMSR pricing in the tournament context. The authors also show that the pricing problem is NP-hard for some more general betting languages.

In this paper, we will follow this line of research. We study combinatorial prediction markets in which the agents can bet on the weighted sum of values that are associated with future events. Below we give two detailed example application contexts that involve betting on sum. We will be referring to these two example contexts throughout the paper.

Betting on page views: The total page views or impression of a subdomain of a web site (e.g. www.conferences.hu/AAMAS2009 is a subdomain of www.conferences.hu) is the number of visits to this subdomain (for a given period of time). Page views is a standard metric for Internet advertising as it captures quantity of advertisements that can be supplied to the advertisers. If we can predict the page views of a subdomain for the coming month, that is, if we can predict the quantity of supply of the coming month, then we can set better prices for advertisements. Traditionally, the prediction has been solely based on machine learning algorithms. Prediction market improves upon the traditional approach by adding an extra "tweaking/correcting" stage to the predicting process: We initialize the market according to the best algorithm available, then leave it for the invisible hand to figure out the rights and wrongs. (We have implemented a prototype to predict web site page views. More details are in Section 8.)

A subdomain is called a *leaf* subdomain, if it contains no child subdomains under it. For example, www.conferences.hu is not a leaf subdomain because it contains the child subdomain www.conferences.hu/AAMAS2009. The total page views of a non-leaf subdomain is the *sum* of the page views of its child subdomains. For example, a subdomain about NCAA contains a list of child subdomains: NCAA homepage, NCAA basketball, NCAA football, etc. Betting on the page views of a non-leaf subdomain is essentially betting on the sum of the page views of all its child subdomains.

A natural bet (security) in this context would be "the page views of subdomain x is between v_1 and v_2 (for the coming month)" = "the sum of page views of all the child subdomains of x is between v_1 and v_2 (for the coming month)".

Betting on electoral vote count: [1] In the US presidential election, if a party wins the popular vote of a state, then it wins all the electoral votes of that state (winner-takes-all). We can use a binary variable to denote the election result of a state: it takes value 1 if the Democrats win, and it takes value 0 if the Republicans win. The total number of electoral votes won by the Democrats is then the *weighted sum* of all the binary variables, where the weights are the number of electoral votes of different states (e.g. Ohio has 20 electoral votes – its weight is 20). Betting on the number of electoral votes won by the Democrats from a set of states is essentially betting on the weighted sum of the binary variables representing those states.

A natural bet (security) in this context would be "the total number of electoral votes won by the Democrats is between v_1 and v_2" = "the weighted sum of states won by the Democrats is between v_1 and v_2".

Our paper is organized as follows: In Section 2, we review the preliminaries of Hanson's logarithmic market scoring rule market maker. In Section 3, we propose three expressive betting languages that seem natural. The first betting language (SAS) allows agents to bet on the weighted sum of an arbitrary subset of values. The second betting language (SVW) allows agents to set their own weights in their bets but restricts subsets to form a hierarchy. In Section 4 and Section 5, we show that LMSR pricing is NP-hard for both SAS and SVW. The third betting language (SPW) allows the agents to bet on the weighted sum of selected subsets of values, where the weights are predefined and subsets form a hierarchy. We derive a polynomial time pricing algorithm for SPW in Section 6. In Section 7, we discuss the algorithm's generalization to other betting contexts, including betting on maximum/minimum and betting on the product of binary values. Finally, in Section 8, we describe a prototype we built to predict web site page views and discuss the implementation issues that arose.

2. LOGARITHMIC MARKET SCORING RULE MARKET MAKER (LMSR)

Logarithmic market scoring rules [8, 9] are sequential versions of *logarithmic scoring rules*. Scoring rules map probability distributions and results of future events into amounts of reward. Logarithmic scoring rules are *proper* in the sense that when facing such rules, risk-neutral agents will reveal their true subjective probability distributions of the future events to maximize their expected reward.

Logarithmic market scoring rules can be interpreted as follows: The market starts with some initial distribution over the outcome space. When an agent comes in, she can modify the current market distribution at her will. Her reward is then the reward, under a specific logarithmic scoring rule, for the modified distribution, minus the reward for the distribution before modification. At any time, for any agent, since the reward for the distribution before modification is beyond the agent's control, essentially, the agent can only focus on maximizing the reward for the modified distribution. That is, the agents always face a (proper) logarithmic scoring rule. Therefore, it is a dominant strategy for a myopic agent to reveal her true beliefs under LMSR.

LMSR is usually implemented as a market maker. That is, instead of asking the agents to directly modify the market distribution, there is a market maker that is in charge of maintaining a consistent set of prices (probabilities) for all the allowable securities, and the agents modify the market distribution through buying or selling securities. For example, selling securities of an outcome is equivalent to marking down the probability of that outcome in

[1] For simplicity, we assume all states are winner-takes-all and we ignore all third parties.

the market distribution. Obviously, trading securities is more natural than playing with distribution over an outcome space that is usually exponential in size.

A generic LMSR offers securities corresponding to all outcomes. A security on outcome ω pays off \$1 if ω happens, and \$0 otherwise. At any moment, the market maker keeps track of a vector $\mathbf{q} = (q_\omega)_{\omega\in\Omega}$, which indicates the number of outstanding shares of all outcomes. That is, the number of (active) securities covering outcome ω is denoted by q_ω. Ω is the set of all outcomes.

The *instantaneous* price for security ω under LMSR is

$$p_\omega(\mathbf{q}) = \frac{e^{q_\omega/b}}{\sum_{\tau\in\Omega} e^{q_\tau/b}}$$

b is a positive parameter. When b is small, purchasing or shorting a few securities can significantly change the market distribution. When b is large, the effect of buying or selling a few securities is less noticeable, meaning the effective liquidity of the market is large.

Suppose an agent wants to purchase/short one security of ω. The current outstanding shares are denoted by $\mathbf{q}$, and after purchasing/shorting ω, the set of outstanding shares become $\hat{\mathbf{q}}$. Then the cost of the transaction equals the integral of the instantaneous price following any path from $\mathbf{q}$ to $\hat{\mathbf{q}}$. The cost can be written as $C(\hat{\mathbf{q}}) - C(\mathbf{q})$, where function C is a cost function with the following form:

$$C(\mathbf{q}) = b\log\sum_{\tau\in\Omega} e^{q_\tau/b}$$

Function C has another meaning. At any moment, the worst-case subsidy required to run the market maker is at most $C(\mathbf{q})$. If the market starts with 0 shares on all outcomes (which is an usual assumption), then the worst-case subsidy is $b\log|\Omega|$.

In most cases, it is natural to only bet on compound securities on collections of outcomes. For example, the compound security "It will rain on exactly one day in the next week" is a collection of 7 securities on single outcomes: "It will rain on day x only", for all choices of x. A compound security's instantaneous price is just the sum of the instantaneous prices of all the outcomes covered by the compound security.

3. BETTING LANGUAGES

In this paper, we consider combinatorial prediction markets in which the final outcomes can be represented as tuples of values. Specifically, we consider outcome space Ω whose elements are

$$\omega = (x_1, x_2, \ldots, x_n)$$

where $x_i \in \{0, 1, \ldots, N\}$ for all i. [2]

It is easy to see that the size of the outcome space is $(N+1)^n$. For betting on page views, n is the number of leaf subdomains, and N is the upper bound on the page views of the leaf subdomains. For betting on electoral vote count, n is the number of US states, and $N = 1$ (recall that the result of a state is denoted by a binary variable).

We propose three expressive betting languages that seem natural for betting on sum. They offer different levels of expressiveness, and face different levels of computational difficulty. We first introduce the SAS betting language, which is the most general one among the three.

Sum of arbitrary subset (SAS) – betting on sum of arbitrary subset of the x_i.

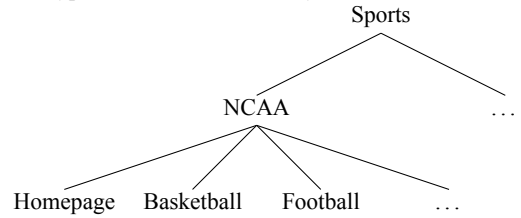
A security under SAS has the following form:

$$v_1 \le \sum_{i\in S} c_i x_i \le v_2$$

where $S \subset \{0, 1, \ldots, n\}$, v_1, v_2 and the c_i are all nonnegative integers. [3] We also allow bets that specify only one end of the triple inequality.

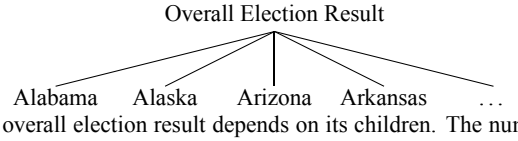
In Section 4, we will show that LMSR pricing is NP-hard even for a restricted version of SAS in which all the c_i are required to be constant 1 (unit weights). Allowing betting on arbitrary subsets makes LMSR pricing computationally infeasible. Therefore, we need to sacrifice some expressiveness. A natural step is to make certain restrictions on the subsets that are allowed to be bet on. Actually, this is not necessarily a bad thing because chances are our interests are focused on selected subsets anyway. We notice that natural events sometimes form *event hierarchies* – events of interests (what we are interested in betting on) correspond to nodes of a tree, and the event corresponding to a non-leaf node is determined by its child nodes. For both example contexts mentioned in the introduction, we see such hierarchies.

Event hierarchy for betting on page views: The following tree describes a typical subdomain hierarchy.



The page views of NCAA is the sum of the page views of its child subdomains. A bet on the page views of NCAA is a bet on the sum of the page views of its child subdomains, that is, the sum of the page views of all the leaf subdomains whose ancestor is NCAA.

Event hierarchy for betting on electoral vote count: In the US presidential election, the state-by-state results, as well as the overall election result, form the following hierarchy:

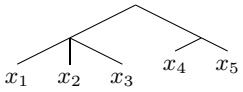


The overall election result depends on its children. The number of electoral votes received by the Democrats in the election is the weighted sum of the results of all states. (Recall that we use binary variables to denote the result of a state: it takes value 1 if the Democrats win, and 0 otherwise.)

The tree structure of an event hierarchy determines which subsets are allowed to be bet on (these subsets correspond to the tree nodes). For example, for the tree below, we are allowed to bet on (the weighted sum of) the following subsets: $\{x_1\}, \{x_2\}, \{x_3\}$, $\{x_4\}, \{x_5\}, \{x_1, x_2, x_3\}, \{x_4, x_5\}, \{x_1, x_2, x_3, x_4, x_5\}$.

[2] It is without loss of generality to restrict the values of the x_i to integers from 0 to N, as long as the x_i take their values from a finite set of rational numbers. For example, if x_1's value is either $1/7$ or $2/3$, then $(21x_1 - 3)$'s value is either 0 or 11, which is in $\{0, 1, \ldots, N\}$ for $N \ge 11$. We can simply bet on the values of the x_i after certain linear transformation.

[3] Again, it is without loss of generality to restrict the values of v_1, v_2 and the c_i to integers, as long as v_1, v_2 and the c_i are rational numbers. For example, a bet on $2/5 \le 1/5x_1 + 1/2x_2 \le 3/7$ can simply be rewritten as $28 \le 14x_1 + 35x_2 \le 30$.



Now we are ready to introduce SVW and SPW.

Sum with varying weights (SVW) – betting on the weighted sum of selected subsets of the x_i. Only subsets corresponding to tree nodes are allowed to be bet on. The agents can set their own weights in their bets.

A security under SVW has the following form:

$$v_1 \leq \sum_{i \in S} c_i x_i \leq v_2$$

where S corresponds to a tree node, v_1, v_2 and the c_i are integers specified by the agents. v_1 and v_2 are nonnegative. The c_i are positive. [4] We also allow bets that specify only one end of the triple inequality.

In Section 5, we will show that LMSR pricing is NP-hard for SVW for any event hierarchy (tree structure).

Sum with predefined weights (SPW) – betting on the weighted sum of selected subsets of the x_i. Only subsets corresponding to tree nodes are allowed to be bet on. The weights are predefined.

A security under SPW has the following form:

$$v_1 \leq \sum_{i \in S} c_i x_i \leq v_2$$

where S corresponds to a tree node, v_1, v_2 are nonnegative integers specified by the agents, and the c_i are predefined constant integers. We also allow bets that specify only one end of the triple inequality.

In Section 6, we derive a polynomial time pricing algorithm for the SPW betting language.

4. COMPLEXITY OF SAS

In this section, we analyze the complexity of pricing using LMSR for the SAS betting language. We show that LMSR pricing is NP-hard even for a restricted version of SAS in which all the weights are required to be 1 (unit weights).

CLAIM 1. *LMSR pricing for the SAS betting language is NP-hard.*

PROOF. Recall that a security under SAS has the following form: $v_1 \leq \sum_{i \in S} c_i x_i \leq v_2$, where S is an arbitrary subset of $\{0, \ldots, n\}$, v_1, v_2 and the c_i are nonnegative integers. In this proof, we will only need to consider securities with the following form: $v_1 \leq \sum_{i \in S} x_i \leq v_2$. That is, we only consider securities in which all the c_i are equal to constant 1. We will show that even if agents only bet on these restricted securities, pricing using LMSR is still NP-hard.

Let us consider an arbitrary 3-SAT expression with n_v binary variables $z_1, z_2, \ldots, z_{n_v}$ and n_c clauses. E.g.

$$\underbrace{(z_1 \vee \neg z_2 \vee z_{n_v}) \wedge (z_5 \vee \neg z_6 \vee z_{n_v}) \wedge \ldots \wedge (z_1 \vee \neg z_1 \vee z_7)}_{n_c}$$

We will show that LMSR pricing for SAS involves solving the satisfiability problem of the above 3-SAT expression.

Recall that the outcome space Ω is the set of all n tuples $\omega = (x_1, x_2, \ldots, x_n)$, where $x_i \in \{0, 1, \ldots, N\}$ for all i. Let us consider a LMSR market marker for which $n = (2n_v + 2)n_c$. For presentation purpose, we rename the x_i so that the outcomes are now n tuples as follows:

$$\begin{gathered}
(\underbrace{z_{11}, \bar{z}_{11}, z_{21}, \bar{z}_{21}, \ldots, z_{n_v 1}, \bar{z}_{n_v 1}, u_1, v_1}_{2n_v+2}, \\
\underbrace{z_{12}, \bar{z}_{12}, z_{22}, \bar{z}_{22}, \ldots, z_{n_v 2}, \bar{z}_{n_v 2}, u_2, v_2}_{2n_v+2}, \\
\vdots \\
\underbrace{z_{1n_c}, \bar{z}_{1n_c}, z_{2n_c}, \bar{z}_{2n_c}, \ldots, z_{n_v n_c}, \bar{z}_{n_v n_c}, u_{n_c}, v_{n_c}}_{2n_v+2}) \\
\text{(altogether } n_c \text{ rows)}
\end{gathered}$$

z_{ij}, $\bar{z}_{ij}$, u_j and v_j are in $\{0, 1, \ldots, N\}$ for i from 1 to n_v and j from 1 to n_c.

Basically, we want to link the value of z_{ij} to the value of z_i in the j-th clause of the 3-SAT expression under consideration. (For each i, we need to make sure that the values of the z_{ij} are the same over different j, since z_i's value should be the same in all clauses.) We want to link the value of $\bar{z}_{ij}$ to the logical negative of z_i. The u_j and v_j are auxiliary variables.

Suppose the following securities have been purchased. (We assume that there were no outstanding securities when the market started. That is, the following securities are the only outstanding securities.)

1. P securities on $0 \leq z_{ij} \leq 1$ for all i from 1 to n_v and all j from 1 to n_c
2. P securities on $0 \leq \bar{z}_{ij} \leq 1$ for all i from 1 to n_v and all j from 1 to n_c
3. P securities on $0 \leq u_j \leq 1$ for all j from 1 to n_c
4. P securities on $0 \leq v_j \leq 1$ for all j from 1 to n_c
5. P securities on $\sum_{j=1}^{n_c} z_{ij} = 0$ for all i from 1 to n_v; P securities on $\sum_{j=1}^{n_c} z_{ij} = n_c$ for all i from 1 to n_v
6. P securities on $z_{ij} + \bar{z}_{ij} = 1$ for all i from 1 to n_v and all j from 1 to n_c
7. P securities on $0 \leq w_j + u_j + v_j \leq 3$ for all j, where w_j is the sum of three selected variables among z_{ij} and $\bar{z}_{ij}$ that correspond to the three literals in the j-th clause of the 3-SAT expression under consideration. For example, if the j-th clause of the 3-SAT expression is $(z_1 \vee \neg z_2 \vee z_3)$, then $w_j = z_{1j} + \bar{z}_{2j} + z_{3j}$
8. Q securities on $w_j + u_j + v_j = 3$ for all j, where w_j is defined the same as above

$$P = (n_c + 1)(2n_v + 2)n_c \log(N)b$$
$$Q = (2n_v + 2)n_c \log(N)b$$

If there exists a satisfactory assignment of the 3-SAT expression, then there exists one outcome that satisfies all the above groups of securities [5]. (Let the z_i be any satisfactory assignment. The following outcome satisfies all the groups: $z_{ij} = z_i$ for all i and j; $\bar{z}_{ij} = \bar{z}_i$ for all i and j; If $w_j = 1$, then $u_j = v_j = 1$; If $w_j = 2$, then $u_j = 1$ and $v_j = 0$; If $w_j = 3$, then $u_j = v_j = 0$.)

If there exists one outcome that satisfies all the above groups of securities, then it corresponds to a satisfactory assignment of the 3-SAT expression. One satisfactory assignment is simply $z_i = z_{ij}$ for arbitrary j. (All the variables are binary according to the first four groups of bets. For specific i, the values of the z_{ij} are the same

[4] If the c_i are allowed to be zeros, then it reduces to the case of betting on arbitrary subsets.

[5] We say an outcome satisfies the fifth group of securities if it satisfies half of them (n_v out of $2n_v$).

over all j (either all 0 or all 1) according to the fifth group of bets. The value of z_{ij} and $\bar{z}_{ij}$ are different according to the sixth group of bets. That is, $\neg z_i$ corresponds to $\bar{z}_{ij}$ for all j. All the clauses of the 3-SAT are satisfied by the z_i according to the eighth group of securities, since $w_j + u_j + v_j = 3$ implies $w_j \geq 1$.)

That is, there exists one outcome that satisfies all the above groups of securities if and only if the 3-SAT expression has a satisfactory assignment.

Consider the pricing problem of the following security:

$$\sum_{j=1}^{n_c}(w_j + u_j + v_j) = 3n_c$$

This security is allowed by the SAS betting language, since it is the sum of a subset of variables with unit weights. From now on, we refer to this security as the objective security.

We first assume that the 3-SAT expression is satisfiable. There exists at least one outcome that satisfies all the above groups of securities, and it must be covered by the objective security according to the eighth group of existing securities. Recall that the instantaneous price for outcome ω is $p_\omega(\mathbf{q}) = \frac{e^{q_\omega/b}}{\sum_{\tau\in\Omega} e^{q_\tau/b}}$, where q_ω is the number of outstanding shares for outcome ω. The number of outstanding shares for an outcome that satisfies all the existing groups of securities is $P(2n_vn_c + 2n_c + n_v + n_vn_c + n_c) + Qn_c$. The instantaneous price for such an outcome is

$$\frac{e^{(P(3n_vn_c+3n_c+n_v)+Qn_c)/b}}{\sum_{\tau\in\Omega} e^{q_\tau/b}}$$

Since a compound security's instantaneous price is the sum of the instantaneous prices of all the outcomes covered by the compound security, we have that the price of the objective security is greater than or equal to the above expression.

If an outcome is not covered by the objective security, then it makes $w_j + u_j + v_j \neq 3$ for at least one j. The number of outstanding shares for such an outcome is at most $P(2n_vn_c + 2n_c + n_v + n_vn_c + n_c) + Q(n_c - 1)$. Therefore, the instantaneous price for such an outcome is at most

$$\frac{e^{(P(3n_vn_c+3n_c+n_v)+Q(n_c-1))/b}}{\sum_{\tau\in\Omega} e^{q_\tau/b}}$$

There are at most $N^n - 1 = N^{(2n_v+2)n_c} - 1$ such outcomes (N^n is the size of the outcome space, and there is at least one outcome that is covered by the objective security according to our assumption). So the sum of the instantaneous prices of all outcomes that are not covered by the objective security is at most

$$\frac{(N^{(2n_v+2)n_c} - 1)e^{(P(3n_vn_c+3n_c+n_v)+Q(n_c-1))/b}}{\sum_{\tau\in\Omega} e^{q_\tau/b}}$$

$$< \frac{e^{(2n_v+2)n_c\log(N)+(P(3n_vn_c+3n_c+n_v)+Q(n_c-1))/b}}{\sum_{\tau\in\Omega} e^{q_\tau/b}}$$

$$= \frac{e^{(P(3n_vn_c+3n_c+n_v)+Qn_c)/b}}{\sum_{\tau\in\Omega} e^{q_\tau/b}}$$

That is, the price of the objective security is greater than the total price of all the outcomes that are not covered by it. Hence, if the 3-SAT expression is satisfiable, then the price of the objective security is greater than $\frac{1}{2}$.

Now we assume that the 3-SAT expression is not satisfiable. There does not exist an outcome that satisfies all the existing groups of securities. If an outcome is covered by the objective security and satisfies the first seven groups of existing securities, then it also satisfies the eighth group of securities, which is contrary to the fact that the 3-SAT expression is not satisfiable. That is, all outcomes covered by the objective security must violate some of the first seven groups of securities.

The number of outstanding shares for any outcome that is covered by the objective security is at most $P(2n_vn_c + 2n_c + n_v + n_vn_c + n_c - 1) + Qn_c$. Therefore, the instantaneous price for such an outcome is at most

$$\frac{e^{(P(3n_vn_c+3n_c+n_v-1)+Qn_c)/b}}{\sum_{\tau\in\Omega} e^{q_\tau/b}}$$

There are at most $N^n = N^{(2n_v+2)n_c}$ such outcomes. So the sum of the instantaneous prices of all the outcomes that are covered by the objective security is at most

$$\frac{N^{(2n_v+2)n_c}e^{(P(3n_vn_c+3n_c+n_v-1)+Qn_c)/b}}{\sum_{\tau\in\Omega} e^{q_\tau/b}}$$

$$= \frac{e^{(2n_v+2)n_c\log(N)+(P(3n_vn_c+3n_c+n_v-1)+Qn_c)/b}}{\sum_{\tau\in\Omega} e^{q_\tau/b}}$$

$$= \frac{e^{(P(3n_vn_c+3n_c+n_v-1)+Q(n_c+1))/b}}{\sum_{\tau\in\Omega} e^{q_\tau/b}}$$

Now consider an outcome that corresponds to an arbitrary assignment of the 3-SAT expression. (E.g. $z_{ij} = z_i$ and $\bar{z}_{ij} = \neg z_i$ for all i and j; $u_j = 0$ and $v_j = 0$ for all j.) The outcome satisfies the first seven groups of existing securities, and does not satisfy the objective security. Its instantaneous price is at least

$$\frac{e^{P(3n_vn_c+3n_c+n_v)/b}}{\sum_{\tau\in\Omega} e^{q_\tau/b}}$$

$$\geq \frac{e^{(P(3n_vn_c+3n_c+n_v-1)+Q(n_c+1))/b}}{\sum_{\tau\in\Omega} e^{q_\tau/b}}$$

That is, if the 3-SAT expression is not satisfiable, then the price of the objective security is less than or equal to $\frac{1}{2}$.

If there exists a LMSR pricing algorithm for SAS that takes only $P(n)$ time, then there exists an algorithm that solves any 3-SAT satisfiability problem with n_v variables in $P(n)$ time. Since $n = (2n_v + 2)n_c \leq (2n_v + 2)\binom{2n_v}{3}$, the algorithm solves any 3-SAT satisfiability problem with n_v variables in $P(n_v)$ time. This is impossible unless P=NP. Therefore, LMSR pricing for the SAS betting language is NP-hard. □

5. COMPLEXITY OF SVW

In this section, we analyze the complexity of pricing using LMSR for the SVW betting language. The proof of Claim 1 tells us that if agents are allowed to bet on arbitrary subsets of the x_i, then the pricing problem is NP-hard even if we require unit weights. SVW restricts the set of subsets that are allowed to be bet on (only subsets corresponding to tree nodes are eligible), but on the other hand, it still allows agents to set their own weights in the their bets. It turns out that the result is still negative: LMSR pricing is NP-hard for SVW for any event hierarchy.

CLAIM 2. *LMSR pricing for the SVW betting language is NP-hard for any event hierarchy.*

PROOF. Recall that a security under SVW has the following form: $v_1 \leq \sum_{i\in S} c_i x_i \leq v_2$, where S corresponds to a tree node,

v_1, v_2 and the c_i are integers specified by the agents. v_1 and v_2 are nonnegative. The c_i are positive.

For any event hierarchy, the following subsets are always allowed to be bet on: $\{x_i\}$ for $i = 1, 2, \ldots, n$, and $\{x_1, x_2, \ldots, x_n\}$ (they correspond to the leafs and the root). We will construct our proof based on securities only on these subsets. Therefore, our result applies to any event hierarchy.

Suppose the following securities have been purchased. (We assume that there were no outstanding securities when the market started. That is, the following securities are the only outstanding securities.)

$n \log(N)b$ securities on $x_i = 1$ ($c_i x_i = c_i$) for all i from 1 to n.
$n \log(N)b$ securities on $x_i = 0$ ($c_i x_i = 0$) for all i from 1 to n.

Consider the pricing of the following two securities:
$\sum_{i=1}^{n} c_i x_i = I$ (security A) and $\sum_{i=1}^{n} c_i x_i = 0$ (security B).
I is a specific positive integer.

If there exists a subset of the c_i that sum to exactly I, then there exists one outcome $(t_1, t_2, \ldots, t_n)$ whose number of outstanding shares is $n^2 \log(N)b$, by setting $t_i = 1$ if c_i is in the subset of numbers that sum to I, and $t_i = 0$ otherwise. This outcome is covered by security A. For security B, the only outcome it covers is $(0, 0, \ldots, 0)$, whose number of outstanding shares is also $n^2 \log(N)b$. Therefore, the price of security A is at least as great as the price of security B, if there exists a subset of the c_i that sum exactly to I.

If there does not exist a subset of the c_i that sum to I, then all outcomes covered by security A have at most $n(n-1)\log(N)b$ outstanding shares. There are at most $N^n - 1$ such outcomes. The instantaneous price of security A is then at most

$$\frac{(N^n - 1)e^{n(n-1)\log(N)}}{\sum_{\tau \in \Omega} e^{q_\tau / b}}$$

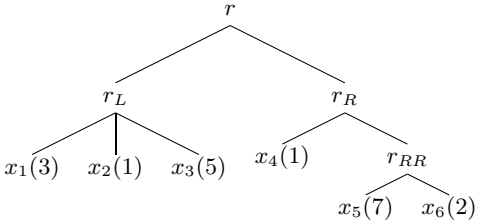
$$< \frac{N^n e^{n(n-1)\log(N)}}{\sum_{\tau \in \Omega} e^{q_\tau / b}} = \frac{e^{n^2 \log(N)}}{\sum_{\tau \in \Omega} e^{q_\tau / b}}$$

We notice that the right-hand side of the inequality is exactly the instantaneous price of security B. That is, the price of security A is less than the price of security B, if there does not exist a subset of the c_i that sum exactly to I.

Therefore, LMSR pricing for the SVW betting language is at least as difficult as the Subset-sum problem with n positive integers, which is NP-complete. □

6. A POLYNOMIAL-TIME PRICING ALGORITHM FOR SPW

The SPW betting language allows agents to bet on the weighted sum of selected subsets of the x_i. These subsets correspond to events that form a tree. The weights are predefined. For example, for the following event hierarchy (values in the parenthesis are the predefined weights of the nodes):



The agents are allowed to bet on the values of $x_1, x_2, \ldots, x_6$, r_L ($3x_1 + 1x_2 + 5x_3$), r_{RR} ($7x_5 + 2x_6$), r_R ($1x_4 + 7x_5 + 2x_6$), and r ($3x_1 + 1x_2 + 5x_3 + 1x_4 + 7x_5 + 2x_6$).

Before introducing our algorithm, we first propose the following lemmas.

LEMMA 1. *Let y be a random variable associated with any event. The distribution of y is characterized by the outstanding securities in the market. Let v be an arbitrary constant. If $P(y = v) = p$, then after introducing one extra security on $y = v$, we have $P(y = v) = \frac{pe^{1/b}}{pe^{1/b} + (1-p)}$.*

One way to interpret the above claim is that, after introducing one extra copy of security $y = v$, the probability of $y = v$ is first magnified by a factor of $e^{1/b}$, then the distribution vector of y is normalized (multiplied by some value so that the sum of all the elements is back to 1).

PROOF. We use $\mathbf{q} = (q_\tau)_{\tau \in \Omega}$ to indicate the number of outstanding shares of all outcomes before introducing the extra security. We have

$$p = \frac{\sum_{\tau \in \{\tau | y = v\}} e^{q_\tau / b}}{\sum_{\tau \in \Omega} e^{q_\tau / b}}$$

$$= \frac{\sum_{\tau \in \{\tau | y = v\}} e^{q_\tau / b}}{\sum_{\tau \in \{\tau | y = v\}} e^{q_\tau / b} + \sum_{\tau \in \{\tau | y \neq v\}} e^{q_\tau / b}}$$

After introducing the extra security, we have

$$P(y = v) = \frac{\sum_{\tau \in \{\tau | y = v\}} e^{(q_\tau + 1)/b}}{\sum_{\tau \in \{\tau | y = v\}} e^{(q_\tau + 1)/b} + \sum_{\tau \in \{\tau | y \neq v\}} e^{q_\tau / b}}$$

$$= \frac{pe^{1/b}}{pe^{1/b} + (1 - p)}$$

□

LEMMA 2. *Let y, z be two random variables that represent the values of two arbitrary events. The distribution of y is characterized by the outstanding securities in the market. Let v_y, v_z be two arbitrary constants. Let $p = P(y = v_y)$, $p' = P(y = v_y | z = v_z)$ and $p'' = P(y = v_y | z \neq v_z)$.*

1. *If we introduce M copies of security $z = v_z$ into the market, then we have $\lim_{M \to \infty} P(y = v_y) = p'$.*
2. *If we introduce negative M copies of security $z = v_z$ (both M copies of $z > v_z$ and M copies of $z < v_z$) into the market, then we have $\lim_{M \to \infty} P(y = v_y) = p''$.*

PROOF. Due to space constraint, we will only present the proof of statement 2.

We use $\mathbf{q} = (q_\tau)_{\tau \in \Omega}$ to indicate the number of outstanding shares of all outcomes. We have

$$p'' = P(y = v_y | z \neq v_z) = P(y = v_y \wedge z \neq v_z) / P(z \neq v_z)$$

$$= \frac{\sum_{\tau \in \{\tau | y = v_y \wedge z \neq v_z\}} e^{q_\tau / b}}{\sum_{\tau \in \Omega} e^{q_\tau / b}} / \frac{\sum_{\tau \in \{\tau | z \neq v_z\}} e^{q_\tau / b}}{\sum_{\tau \in \Omega} e^{q_\tau / b}}$$

$$= \frac{\sum_{\tau \in \{\tau | y = v_y \wedge z \neq v_z\}} e^{q_\tau / b}}{\sum_{\tau \in \{\tau | z \neq v_z\}} e^{q_\tau / b}}$$

After shorting M copies of security $z = v_z$, we have $P(y = v_y)$ equals

$$\frac{\sum_{\tau \in \{\tau | y = v_y \wedge z = v_z\}} e^{(q_\tau - M)/b} + \sum_{\tau \in \{\tau | y = v_y \wedge z \neq v_z\}} e^{q_\tau / b}}{\sum_{\tau \in \{\tau | z = v_z\}} e^{(q_\tau - M)/b} + \sum_{\tau \in \{\tau | z \neq v_z\}} e^{q_\tau / b}}$$

As M goes to infinity, $e^{(q_\tau - M)/b}$ goes to 0 for any τ. So

$$P(y = v_y) = \frac{\sum_{\tau \in \{\tau | y = v_y \wedge z \neq v_z\}} e^{q_\tau / b}}{\sum_{\tau \in \{\tau | z \neq v_z\}} e^{q_\tau / b}} = p''$$

□

LEMMA 3. *The outcome space consists of tuples of n coordinates. If the n coordinates can be separated into k groups, and no outstanding security mentions coordinates of different groups, then coordinates of different groups are independent – the market can be interpreted as k separate markets.*

One simple example suffices to illustrate the idea behind the above claim. Let the outcome space be $\{(x_1, x_2, x_3) | x_1$ is the number of states won by the Democrats in the election; x_2 is the number of states won by the Republicans in the election; x_3 is tomorrow's temperature$\}$. When the market starts (with no outstanding securities), the x_i are pairwise independent (property of LMSR). Now suppose the securities are divided into two groups. One group of securities are on election. They have some effect on the distribution of x_1 or x_2 or both. The other group of securities are on temperature. They have some effect on the distribution of x_3. With these two groups of securities, (x_1, x_2) and x_3 are still independent. A detailed proof is omitted due to space constraint.

Now we are ready to introduce the LMSR pricing algorithm for SPW. The algorithm takes as input the set of outstanding securities and an objective security of the following form: $r = v$ where r is an event (a tree node) and v is a constant integer. The algorithm outputs the instantaneous price (probability) of the objective security. (The price of a security on a range, e.g. $v_1 \leq r \leq v_2$, can be computed as $\sum_{v=v_1}^{v_2} P(r = v)$.)

The tree nodes are random variables that take integer values from 0 to CnN, where C is the maximal weight (constant). We will use array of size $CnN + 1$ as the data structure for storing distribution of a random variable.

The algorithm is based on the following routine $dist(r)$: it computes the market distribution of the random variable corresponding to tree node r, considering only outstanding securities on r and r's offspring (ignoring all other outstanding securities).

Outline of the algorithm

Let r_0 be the root of the tree. To compute the price of security $r_0 = v$, we simply run $dist(r_0)$ (no securities ignored). To compute the price of security $r = v$ where $r \neq r_0$, we first run $dist(r_0)$ to get the distribution of r_0. Then we recompute $dist(r_0)$, considering an extra infinite copies of security $r = v$. By Lemma 2, $dist(r_0)$ returns the distribution of r_0 conditional on $r = v$. Then we recompute $dist(r_0)$, considering an extra negative infinite copies of security $r = v$. By Lemma 2, we get the distribution of r_0 conditional on $r \neq v$. Since $P(r_0 = v_0) = P(r_0 = v_0 | r = v)P(r = v) + P(r_0 = v_0 | r \neq v)(1 - P(r = v))$ for any v_0, we can solve for the value of $P(r = v)$ based on the computed distributions.

Outline of the routine $dist(r)$

r is not a leaf node: Recall that when computing $dist(r)$, we are considering only securities on r and r's offspring. We further ignore all securities on r. The remaining securities are separated into a few groups, with each group corresponding to a branch of r's offspring. Let $r_1, r_2, \ldots, r_k$ be r's children. According to Lemma 3, the values of r's children are independent, and the distribution of r_i is just $dist(r_i)$. We compute $dist(r_i)$ for all i. Then we compute the distribution of r by aggregating all $dist(r_i)$. We first compute the distribution of $s_2 = r_1 + r_2$ by aggregating the distribution of r_1 and r_2. We then compute the distribution of $s_3 = r_1 + r_2 + r_3$ by aggregating the distribution of s_2 and r_3. We are done in $k - 1$ steps. The time complexity of each step is at most the square of the size of the distribution vector, which is polynomial in n and N. Therefore, the whole aggregation process is polynomial time. Now we have the distribution of r. However, this is the distribution that considers only securities on r's offspring. To get $dist(r)$, we need to add back in all securities on r. According to Lemma 1, we only need to magnify the probability of $r = v$ by a factor of $e^{x_v/b}$ (x_v is the number of securities on $r = v$: securities having the form of $v_1 \leq r \leq v_2$, with $v_1 \leq v \leq v_2$), and then normalize the distribution vector. [6]

r is a leaf node: For leaf node r, computing $dist(r)$ is much easier. For any possible value v, the probability of $r = v$ is proportional to $e^{x_v/b}$, where x_v is the number of securities on $r = v$.

Complexity of the algorithm

We only need to show that the routine $dist(r)$ is polynomial time. $dist(r)$ is a recursive routine, but it visits any node at most once. The number of nodes is polynomial in n (each non-leaf node has at least two children). The non-recursive part of $dist(r)$ takes polynomial time (in n and N). Therefore, $dist(r)$ is polynomial time in n and N, so is our algorithm.

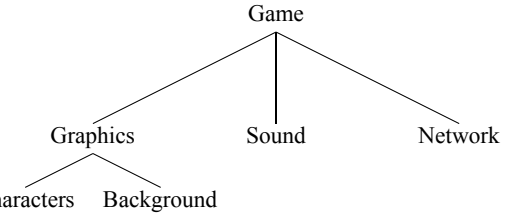
7. OTHER BETTING CONTEXTS

So far we have been only focusing on event hierarchies based on weighted sum. In principle, the algorithm we proposed in the previous section can be applied to any betting context, as long as the events of interest form a tree structure. However, for some betting contexts, the algorithm may not be polynomial time. A sufficient condition for the algorithm to be polynomial time is that

- The size of the set of all possible values over all tree nodes is polynomial of n and N.
- Let r be an arbitrary non-leaf tree node. Let $r_1, r_2, \ldots, r_k$ be r's children. r can be written as $r_1 \oplus r_2 \oplus \ldots \oplus r_k$, where $\oplus$ is an associative binary operator (e.g. addition, multiplication). That is, $r = ((((r_1 \oplus r_2) \oplus r_3) \oplus r_4) \ldots \oplus r_k)$. The operator $\oplus$ may be different for different tree nodes. [7]

In this section, we give two example contexts based on operators other than sum – betting on maximum/minimum and betting on the product of binary values. Both example contexts satisfy the above sufficient condition. Hence for both contexts, our algorithm can be applied (polynomial time).

Betting on maximum or minimum: Let us consider the following scenario. An electronic game company wants to predict the earliest possible release date of its next generation game. The game's components are organized as follows:



We may run a combinatorial prediction market that allows people to bet on the maximum number of days it takes from today to finish a component. A bet (security) would be like "Background can be

[6] If $x_v = \infty$, then $P(r = v) = 1$. If $x_v = -\infty$, $P(r = v) = 0$.

[7] A more general version of this condition is that r can be written as $f((((r_1 \oplus_1 r_2) \oplus_2 r_3) \ldots \oplus_{k-1} r_k)$, where the $\oplus_i$ are arbitrary binary operators and f is an arbitrary function. (We assume that the operators and the function can be evaluated in polynomial time.)

finished in 60 days", or "The whole game can be finished in 100 days". This is an event hierarchy based on the maximum operator $x \oplus y = \max(x, y)$: the number of days it takes to finish a (non-leaf) component is the maximum of the number of days it takes to finish any child component.

Betting on the product of binary values: Let us consider a slightly modified scenario. The company wants to predict whether the game can be released before some deadline. We may run a combinatorial prediction market in which people bet on whether a component can be finished on time. We use a binary value to denote whether a component can be finished before the deadline. A bet (security) would be like "Background can be finished before the deadline", or "The whole game can not be finished before the deadline". This is an event hierarchy based on the product of binary values: a (non-leaf) component can be finished on time if and only if all its child components can be finished before the deadline.

8. BETTING ON PAGE VIEWS: IMPLEMENTATION ISSUES

Based on the algorithm proposed in Section 6, we implemented a prototype to predict web site page views. The prediction market we implemented is truly combinatorial with exponential-size state space, yet the prices of the allowable securities can be computed in real time. The prototype takes the form of a multi-user web application. Below we briefly discuss a few issues we encountered during the implementation.

The page views of a subdomain can be huge (in the magnitude of billions). Our algorithm can not handle such a large N from a practical point of view. Let L and U be the lower bound and upper bound on the page views of the subdomains. Recall that a bet takes the form of $v_1 \leq r \leq v_2$. Instead of allowing v_1 and v_2 to be arbitrary integers in $\{0, 1, \ldots, U\}$, we require them to be taken from $\{L + i\Delta | i = 0, 1, \ldots, (U - L)/\Delta\}$. Large (small) Δ leads to faster (slower) computation and lower (higher) precision.

Even though our aim is to build a combinatorial prediction market where we can bet on the page views of all subdomains, we choose to compromise on this idea of having a single market when the page views of a subdomain is significantly less than that of its siblings (judging from historical data). We simply ignore such subdomains or run separate markets for them, because activities on these subdomains do not affect the market distribution (of their siblings and ancestors) in a noticeable way, and by removing them we speed up the computation.

Even when speed is not a concern, sometimes it is beneficial to separate the market. Let r be a node deep down the tree. In our algorithm, to compute the price of $r = v$, we compute three distributions: the distribution of r_0 (root), the distribution of r_0 conditional on $r = v$, and the distribution of r_0 conditional on $r \neq v$. Then we solve for the value of $P(r = v)$ based on the fact that $P(r_0 = v_0) = P(r_0 = v_0 | r = v)P(r = v) + P(r_0 = v_0 | r \neq v)(1 - P(r = v))$ for any v_0. However, if the three distributions are close (which is likely to be the case if r is deep down the tree), then the solution based on the above equation may contain a significant numerical error (division by a value that is close to 0). A better idea is to pretend that the root is only a few levels above r. That is, we only consider a branch of the market when dealing with nodes deep down the tree.

9. CONCLUSION

We studied combinatorial prediction markets where agents bet on the sum of values at any node in a hierarchy of events, for example the sum of page views among all the children within a web subdomain. We proposed three expressive betting languages that seem natural, and analyzed the complexity of pricing using Hanson's logarithmic market scoring rule (LMSR) market maker. *Sum of arbitrary subset (SAS)* allows agents to bet on the weighted sum of an arbitrary subset of values. *Sum with varying weights (SVW)* allows agents to set their own weights in their bets but only allows bets on nodes in a hierarchy. We showed that LMSR pricing is NP-hard for both SAS and SVW. *Sum with predefined weights (SPW)* allows agents to bet on the weighted sum of subsets corresponding to nodes in a hierarchy, where the weights are predefined. We derived a polynomial time pricing algorithm for SPW. We discussed the algorithm's generalization to other betting contexts, including betting on max/min and betting on the product of binary values. Finally, we described a prototype we built to predict web site page views and discussed the implementation issues that arose.

10. REFERENCES

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

[2] Y. Chen, L. Fortnow, N. Lambert, D. M. Pennock, and J. Wortman. Complexity of combinatorial market makers. In *Proceedings of the ACM Conference on Electronic Commerce (EC)*, Chicago, IL, USA, 2008.

[3] Y. Chen, L. Fortnow, E. Nikolova, and D. M. Pennock. Betting on permutations. In *Proceedings of the ACM Conference on Electronic Commerce (EC)*, pages 326–335, San Diego, CA, USA, 2007.

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

[5] R. Forsythe, F. Nelson, G. R. Neumann, and J. Wright. Anatomy of an experimental political stock market. *The American Economic Review*, 82(5):1142–1161, Dec. 1992.

[6] R. Forsythe, T. A. Rietz, and T. W. Ross. Wishes, expectations and actions: a survey on price formation in election stock markets. *Journal of Economic Behavior and Organization*, 39:83–110, 1999.

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

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

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

[10] K. Oliven and T. A. Rietz. Suckers are born but markets are made: Individual rationality, arbitrage, and market efficiency on an electronic futures market. *Management Science*, 50(3):336–351, 2004.

[11] D. M. Pennock, S. Lawrence, C. L. Giles, and F. A. Nielsen. The real power of artificial markets. *Science*, 291:987–988, Feb. 2002.

[12] C. R. Plott and S. Sunder. Efficiency of experimental security markets with insider information: An application of rational expectations models. *Journal of Political Economy*, 90:663–698, 1982.

[13] C. R. Plott and S. Sunder. Rational expectations and the aggregation of diverse information in laboratory experiments. *Econometrica*, 56:1085–1118, 1988.