Log-time Prediction Markets for Interval Securities

Miroslav Dudík[1,†], Xintong Wang[2,†], David M. Pennock[3], and David M. Rothschild[1]

[1] Microsoft Research, New York City. {mdudik,davidmr}@microsoft.com
[2] University of Michigan, Ann Arbor. xintongw@umich.edu
[3] Rutgers University. dpennock17@gmail.com
† Authors contribute equally

Abstract. We design a prediction market to recover a complete and fully general probability distribution over a continuous random variable. Traders buy and sell *interval securities*, or binary contracts that pay \$1 if the outcome falls into an interval and \$0 otherwise. We allow traders to express any interval endpoint of arbitrary precision, in service of recovering distributions of any complex shape and number of modes. Our market takes the form of a central *automated market maker* offering prices for every interval security that traders propose. All market operations can be achieved in time logarithmic in the number of outcomes (that traders distinguish), providing the first computationally efficient market for a continuous variable. We present two designs. The first replicates the popular *logarithmic market scoring rule* (LMSR) exactly, keeping its bounded-loss guarantee, while utilizing a balanced binary tree to execute all operations exponentially faster than standard LMSR markets. We exploit modularity properties of LMSR to decompose computations along nodes of the tree. The second features two or more parallel LMSR market makers that mediate submarkets with increasingly fine-grained outcome partitions. This design remains computationally efficient for all operations, including arbitrage removal across submarkets, and adds two additional benefits: (1) the ability to flexibly express utility for information at various resolutions by assigning different liquidity values, and (2) the ability to guarantee true *constant* bounded loss by geometrically decreasing the liquidity in each submarket. We conduct simulation experiments to illustrate the flexibility of our second design, showing that it can converge fast at both coarse and fine resolutions, whereas standard LMSR markets need to pick the preferred outcome resolution ex ante.

1 Introduction

Consider a one-dimensional continuous random variable, such as the opening value of the S&P 500 index on December 17, 2021. We design a market for trading interval securities corresponding to predictions that a continuous outcome will fall into some specified interval, say between 2957.60 and 3804.59, implemented as contracts that pay out \$1 if the outcome falls in the interval and \$0 if it does not. We develop two automated market makers that always offer to buy or sell any interval security at some price. Both market makers are fully *expressive*, meaning that traders can select any interval endpoints that they want, with arbitrary precision, corresponding to a continuous outcome space. Our market makers are the first to simultaneously achieve expressiveness and computational efficiency for a continuous variable. In addition, our proposed market designs feature arbitrage-free prices and bounded loss for the market maker.

A form of interval trade called a *condor spread* is common in financial options markets, with significant volume of trade and money. However, the available financial options are not expressive: they only support a limited subset of approximate interval trades. An S&P 500 *call option* with *strike price* 3300 and expiration date December 17, 2021, pays $\max\{S - 3300, 0\}$, where S is the opening value of the index on the expiration date. As of this writing, S&P 500 options expiring on December 17, 2021, distinguish 56 different strike prices, allowing the purchase of around 1500 distinct intervals of minimum width 25. Moreover, each condor spread, or approximate interval, requires trading four different options.[1] Finally, as each strike price trades independently despite the obvious logical constraints on their relative values, the market will require processing time linear in the number of offered strike prices to remove arbitrage.

Outside traditional financial markets, the popular *logarithmic market scoring rule* (LMSR) market maker [15,16] has been used to elicit information through trade of interval securities. For instance, the

[1] For example, 25 shares of "\$1 iff [2650,2775]" $\approx$ $\max\{S - 2650, 0\} - \max\{S - 2675, 0\} - \max\{S - 2750, 0\} + \max\{S - 2775, 0\}$.

Gates Hillman Prediction Market at Carnegie Mellon University operated LMSR on 365 outcomes, representing 365 days of one year, to forecast the opening time of the new computer science building [21]. Traders could bet on different intervals by choosing a start and an end date. A similar market[2] was later launched at the University of Texas at Austin, using a liquidity-sensitive variation of LMSR [20]. Moreover, LMSR has been deployed to predict product-sales levels [23], instructor ratings [4], and political events [14].

LMSR has two limitations that prevent its scaling to markets with a continuous outcome space. First, LMSR's worst-case loss can grow unbounded if traders select intervals with prior probability approaching zero [12]. Second, and crucially, standard implementations of LMSR operations run in time linear in the number of outcomes, or the number of distinct future values traders define—in our case, arbitrarily many. The constant-log-utility market maker [8] and other barrier-function-based market makers [22] feature constant bounded loss and so avoid the former problem, but still suffer the latter regarding computational intractability. Thus, basic operations in previously studied and deployed markets feature a relatively small set of predetermined interval securities and run in time linear in the number of supported outcomes, limiting the ability to aggregate high-precision trades and elicit the full distribution of a continuous random variable.

In this paper, we develop two new market makers that both operate exponentially faster than LMSR and previous designs. Market operations can be run in time *logarithmic* in the number of distinct intervals traded, or linear in the number of bits describing the outcome space (e.g., 64 bits if the outcome is represented as IEEE 754 double). In addition to computational efficiency, our market makers update online in such a way that their worst-case loss, even against adversarial trades and outcomes, is bounded.

Our first market maker calculates LMSR prices exactly, but does so exponentially faster by using a balanced binary tree data structure to efficiently implement interval queries and trades. The special form of the LMSR cost function permits its normalization constant—a key LMSR quantity—to be computed via only local computations on the balanced tree. Our work here contributes to the rich literature that shows how to overcome the worst-case #P-hardness of LMSR pricing [5] by exploiting the outcome space structure and limiting expressivity [6,13,7,24,18].

To maximize predictive accuracy for a given level of subsidy, if traders' information is coarse, the LMSR outcome space should be coarse, and if traders' information is fine, the outcome space should be fine. Our second market maker splits liquidity across two or more layers of increasing fineness, allowing the market to automatically match the coarseness of traders' information. It can also maintain arbitrage-free prices, while achieving a *constant* bounded loss that is independent of how infinitesimally small the traders' intervals become. In experiments, we show that the second market maker can get close to the "best of both worlds" displayed by coarse and fine LMSR markets, regardless of the traders' information structure.

Because both proposed market designs allow trading intervals at machine precision, they can elicit any probability distribution over the continuous random variable that can be practically encoded by a machine. Throughout this paper, we use the S&P 500 index value as a running example, since stock options are a prevalent and relevant application, but our framework is generic and can handle any one-dimensional continuous variable, for example: the number of coronavirus infections by the end of the year; the date when a vaccine will be released to market; the landfall point of a hurricane along a coastline; or the number of tickets sold in the first weekend of a new movie release.

2 Formal Setting

We first review cost-function-based market making [1,8], and then introduce interval markets.

2.1 Cost-function-based Market Making

Let Ω denote a finite set of *outcomes*, corresponding to mutually exclusive and exhaustive states of the world. We are interested in eliciting expectations of binary random variables $\phi_i \colon \Omega \to \{0,1\}$, indexed by $i \in \mathcal{I}$, which model the occurrence of various events, such as "*S&P 500 will open between 2957.60 and 3804.59 on December 17, 2021*". Each variable ϕ_i is associated with a *security* that pays out $\phi_i(\omega)$ when the outcome $\omega \in \Omega$ occurs. Therefore, the random variable ϕ_i is also called the *payoff function*. Binary securities pay out \$1 if the specified event occurs, and \$0 otherwise. The vector $(\phi_i)_{i\in\mathcal{I}}$ is denoted $\boldsymbol{\phi}$. Traders trade *bundles* $\boldsymbol{\delta} \in \mathbb{R}^{|\mathcal{I}|}$ of security with a central market maker, where positive entries in $\boldsymbol{\delta}$ correspond to purchases, and negative values the short sales. A trader holding a bundle $\boldsymbol{\delta}$ receives a payoff of $\boldsymbol{\delta} \cdot \boldsymbol{\phi}(\omega)$, when ω occurs.

[2] `https://www.cs.utexas.edu/news/2012/research-corner-gates-building-prediction-market`

Following [1] and [8], we assume that the market maker determines security prices using a convex and differentiable potential function $C\colon \mathbb{R}^{|\mathcal{I}|} \to \mathbb{R}$, called a *cost function*. The state of the market is specified by a vector $\boldsymbol{\theta} \in \mathbb{R}^{|\mathcal{I}|}$, listing the number of shares of each security sold by the market maker so far. A trader wishing to buy a bundle $\boldsymbol{\delta}$ in the market state $\boldsymbol{\theta}$ must pay $C(\boldsymbol{\theta} + \boldsymbol{\delta}) - C(\boldsymbol{\theta})$ to the market maker, after which the new state becomes $\boldsymbol{\theta} + \boldsymbol{\delta}$. Thus, the vector of instantaneous prices in the corresponding state $\boldsymbol{\theta}$ is $\boldsymbol{p}(\boldsymbol{\theta}) \coloneqq \nabla C(\boldsymbol{\theta})$. Its entries can be interpreted as the market's collective estimates of $\mathbb{E}[\phi_i]$: a trader can make an expected profit by buying (at least a small amount of) the security i if she believes that $\mathbb{E}[\phi_i]$ is larger than the instantaneous price $p_i(\boldsymbol{\theta}) = \partial C(\boldsymbol{\theta})/\partial\theta_i$, and by selling if she believes the opposite. Therefore, risk neutral traders with sufficient budgets maximize their expected profits by moving the price vector to match their expectation of $\boldsymbol{\phi}$. Any expected payoff must lie in the convex hull of the set $\{\boldsymbol{\phi}(\omega)\}_{\omega\in\Omega}$, which we denote $\mathcal{M}$ and call a *coherent price space* with its elements referred to as *coherent price vectors*.

We assume that the cost function satisfies two standard properties: *no arbitrage* and *bounded loss*. The *no arbitrage* property requires that as long as all outcomes ω are possible, there be no market transactions with a guaranteed profit for a trader. In this paper, we use the fact that C is arbitrage-free if and only if it yields price vectors $\boldsymbol{p}(\boldsymbol{\theta})$ that are always coherent [1]. The *bounded loss* property is defined in terms of the worst-case loss of a market maker, $\sup_{\boldsymbol{\theta}\in\mathbb{R}^{|\mathcal{I}|}} \big(\sup_{\omega\in\Omega}\big(\boldsymbol{\theta}\cdot\boldsymbol{\phi}(\omega)\big) - C(\boldsymbol{\theta}) + C(\mathbf{0})\big)$, meaning the largest possible difference, across all possible trading sequences and outcomes, between the amount that the market maker has to pay the traders (once the outcome is realized) and the amount that the market maker has collected (when securities were traded). The *bounded loss* property requires that this worst-case loss be a priori bounded by a constant.

2.2 Complete Markets and Logarithmic Market Scoring Rule (LMSR)

In a complete market, we have $\mathcal{I} = \Omega$ and securities are indicators of individual outcomes, $\phi_i(\omega) = 1\{\omega = i\}$, where $1\{\cdot\}$ denotes the binary indicator, equal to 1 if true and 0 if false. For instance, to set up a complete market for the S&P 500 opening price on December 17, 2021, we define ω as the price rounded to the nearest cent and capped at \$5000, meaning that larger prices are treated as \$5000. The resulting complete market is defined by $\mathcal{I} = \Omega = \{0, 0.01, \ldots, 4999.99, 5000\}$. We denote each market security as ϕ_ω. A risk-neutral trader is incentivized to move the price of each security ϕ_ω to her estimate of $\mathbb{E}[\phi_\omega] = \mathbb{P}[\omega]$, that is, to her subjective probability of ω occurring. Thus, traders can express arbitrary probability distributions over Ω.

Throughout the paper, we use variants of logarithmic market scoring rule (LMSR) [15], which is a cost function for a complete market. LMSR has the cost function and prices of the form

$$C(\boldsymbol{\theta}) = b\log\left(\sum_{\omega\in\Omega} e^{\theta_\omega/b}\right), \qquad p_\omega(\boldsymbol{\theta}) = \partial C(\boldsymbol{\theta})/\partial\theta_\omega = \frac{e^{\theta_\omega/b}}{\sum_{\nu\in\Omega} e^{\theta_\nu/b}}, \tag{1}$$

where b is the liquidity parameter, controlling how fast the price moves in response to trading. It is shown that b also controls the worst-case loss of the market maker, which equals $b\log|\Omega|$ [15].

The securities in a complete market can be used to express bets on any event E. Specifically, one share of a security for the event E can be represented by the indicator bundle $\mathbf{1}_E \in \mathbb{R}^\Omega$ with entries $1_{E,\omega} = 1\{\omega \in E\}$. We refer to this bundle as the *bundle security for event E* to highlight the fact that it pays out \$0 or \$1 depending on whether E occurs, but it is *not* one of the market securities ϕ_ω. The immediate price of the bundle $\mathbf{1}_E$ in the state $\boldsymbol{\theta}$ is then

$$p_E(\boldsymbol{\theta}) \coloneqq \mathbf{1}_E \cdot \boldsymbol{p}(\boldsymbol{\theta}) = \sum_{\omega\in E} p_\omega(\boldsymbol{\theta}) = \frac{\sum_{\omega\in E} e^{\theta_\omega/b}}{\sum_{\nu\in\Omega} e^{\theta_\nu/b}}. \tag{2}$$

The cost of buying the bundle $s\mathbf{1}_E$, sometimes referred to as "s shares of $\mathbf{1}_E$", is a function of $p_E(\boldsymbol{\theta})$ and s:

$$\begin{aligned} C(\boldsymbol{\theta} + s\mathbf{1}_E) - C(\boldsymbol{\theta}) &= b\log\left(\sum_{\omega\notin E} e^{\theta_\omega/b} + \sum_{\omega\in E} e^{(\theta_\omega+s)/b}\right) - b\log\left(\sum_{\omega\in\Omega} e^{\theta_\omega/b}\right) \\ &= b\log\Big(p_{E^c}(\boldsymbol{\theta}) + e^{s/b}p_E(\boldsymbol{\theta})\Big) = b\log\Big(1 - p_E(\boldsymbol{\theta}) + e^{s/b}p_E(\boldsymbol{\theta})\Big). \end{aligned} \tag{3}$$

Above, we write E^c for the complementary event $E^c = \Omega\backslash E$, and use the fact $p_E(\boldsymbol{\theta}) + p_{E^c}(\boldsymbol{\theta}) = 1$, which follows immediately from Eq. (2).

2.3 Interval Securities over $[0, 1)$

We consider betting on outcomes within an interval $[0, 1)$. Our approach generalizes to settings in which the outcomes are in $[\alpha, \beta)$ for any $[\alpha, \beta) \subseteq [-\infty, \infty)$, by applying any increasing transformation $F : [\alpha, \beta) \to [0, 1)$ and then running the market on $[0, 1)$. We assume that the outcome ω is specified with K bits, meaning that there are $N = 2^K$ outcomes with $\Omega = \{j/N : j \in \{0, 1, \ldots, N-1\}\}$. For instance, to represent the S&P 500 example, we set $N = 2^{19} = 524{,}288$ and define the transformed outcome $\omega = \omega'/N$, where ω' is the *S&P* 500 price in cents. We can now cap the S&P 500 price at \$5242.87 instead of \$5000. At the end of Sections 3 and 4, we discuss how the assumption of pre-specified bit precision can be removed.

In the outcome space Ω, we would like to enable the price and cost queries as well as buying and selling of bundle securities for the interval events $I = [\alpha, \beta)$ for any $\alpha, \beta \in \Omega \cup \{1\}$.[3] For cost-based markets, sell transactions are equivalent to buying a negative amount of shares, so we only seek to design algorithms for three operations: $\mathbf{price}(I)$, $\mathbf{cost}(I, s)$, and $\mathbf{buy}(I, s)$, where I is the interval event and s the number of shares. A naive implementation of **price** and **cost** queries following Eqs. (2) and (3) would be linear in N. Here we seek to implement these operations in time that is logarithmic in N, i.e., linear in $K = \log N$.

3 A Log-time LMSR Market Maker

We design a data structure, referred to as an *LMSR tree*, which resembles an *interval tree* [9, Section 15.3] but includes additional annotations to support LMSR calculations. We first define the LMSR tree, and show that it can facilitate market operations in time logarithmic in the number of intervals that traders define.

3.1 An LMSR Tree for $[0, 1)$

We represent an LMSR tree T with a *full binary tree*, where each node z has either no children (when z is a leaf) or exactly two children, denoted *left*(z) and *right*(z) (when z is an inner node). We denote the root node as *root* and the parent of any non-root node as *par*(z). Each node z is associated with an interval $I_z = [\alpha_z, \beta_z)$ that follows the tree hierarchy: the root is associated with the full interval $I_{root} = [0, 1)$, and children of a node z hold intervals that partition $I_z = [\alpha_z, \beta_z)$ into adjacent intervals, i.e., $I_{left(z)} = [\alpha_z, \gamma_z)$ and $I_{right(z)} = [\gamma_z, \beta_z)$ for some $\gamma_z \in (\alpha_z, \beta_z)$. We refer to this structural property as the *binary-search property.* It helps to find the unique leaf containing any $\omega \in \Omega$ by descending from *root* and choosing left or right in each node based on whether $\omega < \gamma_z$ or $\omega \geq \gamma_z$.

To achieve log-time market operations, we keep track of the *height* of each node z, defined as the length of the longest path from z to any leaf. In this paper, we adopt an *AVL tree* [3] at the basis of our LMSR tree, but other balanced binary-search trees could also be used, such as red-black trees or splay trees. The node heights, denoted h_z, are used to maintain the *height-balance property* of AVL trees: in each inner node z, we have $|h_{left(z)} - h_{right(z)}| \leq 1$. This property ensures that the path length from root to any leaf is at most $\mathcal{O}(\log n)$, where n is the number of leaves of the tree [17].

To facilitate LMSR computations, we maintain a scalar quantity $s_z \in \mathbb{R}$ for each node z, which records the number of *bundle securities* associated with I_z sold by the market maker. Therefore, the market state represented by an LMSR tree T is

$$\boldsymbol{\theta}(T) = \sum_{z \in T} s_z \mathbf{1}_{I_z}, \tag{4}$$

and we can further obtain the components of $\boldsymbol{\theta}(T)$ for each individual outcome ω as follows[4]:

$$\theta_\omega(T) = \sum_{z \in T} s_z 1_{I_z, \omega} = \sum_{z \ni \omega} s_z. \tag{5}$$

The normalization constant in the LMSR price expression (Eq. 2) is then

$$\sum_{\omega \in \Omega} e^{\theta_\omega / b} = \sum_{\omega \in \Omega} e^{\sum_{\omega \in z} s_z / b} = \sum_{\omega \in \Omega} \prod_{z \ni \omega} e^{s_z / b}. \tag{6}$$

[3] Throughout the paper, we operate in the outcome space discretized to events $\omega \in \Omega$ specified with K bits, but would like to indeed discuss interval events $[a, b)$ that include reals of arbitrary precisions. Since, in measure theory, events are subsets of the outcome space, what we mean here are events of form $[a, b) \cap \Omega$.

[4] To facilitate presentation, we write $\omega \in z$ to mean $\omega \in I_z$, and $z' \subseteq z$ to mean $I_{z'} \subseteq I_z$. Thus, $z' \subseteq z$ corresponds to z' being a descendant of z in T, and $z' \subset z$ corresponds to z' being a strict descendant of z.

We decompose the above normalization constant computation along the nodes of an LMSR tree, by defining a *partial normalization constant* S_z in each node[5]:

$$S_z := \frac{1}{N} \sum_{\omega \in z} \prod_{z' \colon z \supseteq z' \ni \omega} e^{s_{z'}/b}. \tag{7}$$

It enables the following key recursive relationship, which is at the core of implementing **price** and **buy**:

$$S_z = \begin{cases} e^{s_z/b} \cdot (\beta_z - \alpha_z) & \text{if } z \text{ is a leaf,} \\ e^{s_z/b} \cdot \left(S_{\mathit{left}(z)} + S_{\mathit{right}(z)}\right) & \text{otherwise.} \end{cases} \tag{8}$$

Combining the above annotations and properties, we formally define an LMSR tree as follows.

Definition 1. *An* LMSR tree *is a full binary tree, where each node z is annotated with $I_z = [\alpha_z, \beta_z)$, h_z, s_z, and S_z, where $\alpha_z, \beta_z \in \Omega \cup \{1\}$, $h_z \geq 0$, $s_z \in \mathbb{R}$, and $S_z \geq 0$, which satisfy:*

- Binary-search property: $I_{\mathit{root}} = [0, 1)$*, and for every inner node z,*

$$\alpha_z = \alpha_{\mathit{left}(z)} \; < \; \beta_{\mathit{left}(z)} = \alpha_{\mathit{right}(z)} \; < \; \beta_{\mathit{right}(z)} = \beta_z.$$

- Height balance: $h_z = 0$ *for leaves, and for every inner node z,*

$$h_z = 1 + \max\{h_{\mathit{left}(z)}, h_{\mathit{right}(z)}\}, \quad |h_{\mathit{left}(z)} - h_{\mathit{right}(z)}| \leq 1.$$

- Partial-normalization correctness: $S_z = e^{s_z/b} \cdot (\beta_z - \alpha_z)$ *for leaves, and for every inner node z,*

$$S_z = e^{s_z/b} \cdot \left(S_{\mathit{left}(z)} + S_{\mathit{right}(z)}\right).$$

Based on the LMSR tree construction, we implement the following operations for any interval $I = [\alpha, \beta)$:

- **price**(I, T): return the price of bundle security for I;
- **cost**(I, s, T): return the cost of s shares of bundle security for I;
- **buy**(I, s, T): update T to reflect the purchase of s shares of bundle security for I.

In order to implement **cost**, it suffices to implement **price** by Eq. (3). Since the price of $[\alpha, \beta)$ can be obtained from prices for $[\alpha, 1)$ and $[\beta, 1)$, i.e., $p_{[\alpha,\beta)}(\boldsymbol{\theta}) = p_{[\alpha,1)}(\boldsymbol{\theta}) - p_{[\beta,1)}(\boldsymbol{\theta})$, we focus on implementing **price** for intervals of the form $[\alpha, 1)$. Similarly, buying s shares of $[\alpha, \beta)$ is equivalent to first buying s shares of $[\alpha, 1)$ and then buying $(-s)$ shares of $[\beta, 1)$, as in both cases we end up in the same market state $\boldsymbol{\theta} + s\mathbf{1}_{[\alpha,\beta)}$. Thus, we implement **price** and **buy** for one-sided intervals $I = [\alpha, 1)$, and all the remaining operations will follow.

3.2 Price Queries

We consider price queries for $I = [\alpha, 1)$. Let $\mathit{vals}(T) = \{\alpha_z : z \in T\}$ denote the set of distinct left endpoints in the tree nodes. We start by assuming that $\alpha \in \mathit{vals}(T)$, and will later relax this assumption. We proceed in two steps. *First*, we construct the set of nodes $\mathcal{Z}$ whose associated intervals I_z are disjoint and cover I. We construct $\mathcal{Z}$ by doing a binary search for α: as we go down the tree, we put in $\mathcal{Z}$ all of the unvisited right children of the visited nodes, which have $\alpha_z > \alpha$, as well as the final node with $\alpha_z = \alpha$. Recall that n is the number of leaves of the LMSR tree T, and the height balance implies that $\mathcal{Z}$ has a cardinality of $\mathcal{O}(\log n)$. The resulting set $\mathcal{Z}$ satisfies $p_I(\boldsymbol{\theta}) = \sum_{z \in \mathcal{Z}} p_{I_z}(\boldsymbol{\theta})$.

Second, we determine $p_{I_z}(\boldsymbol{\theta})$ for each of the nodes $z \in \mathcal{Z}$. Starting from the LMSR price in Eq. (2), we take advantage of the defined partial normalization constants S_z and calculate $p_{I_z}(\boldsymbol{\theta})$ as follows:

$$p_{I_z}(\boldsymbol{\theta}) = \frac{1}{N S_{\mathit{root}}} \sum_{\omega \in z} e^{\theta_\omega / b} = \frac{1}{S_{\mathit{root}}} \cdot \frac{1}{N} \sum_{\omega \in z} \prod_{z' \ni \omega} e^{s_{z'}/b} \tag{9}$$

$$= \frac{1}{S_{\mathit{root}}} \cdot \frac{1}{N} \sum_{\omega \in z} \left[\left(\prod_{z' \colon z \supseteq z' \ni \omega} e^{s_{z'}/b} \right) \left(\prod_{z' \supset z} e^{s_{z'}/b} \right) \right] \tag{10}$$

$$= \frac{S_z}{S_{\mathit{root}}} \underbrace{\left(\prod_{z' \supset z} e^{s_{z'}/b} \right)}_{P_z}. \tag{11}$$

[5] The $1/N$ scale in Eq. (7) leads to a more natural interpretation of S_z, when z is a leaf (in the recursive relationship).

In Eq. (9), we expand θ_ω using Eq. (5). In Eq. (10), we use the fact that any node z' with a non-empty intersection with z, i.e., $I_z \cap I_{z'} \neq \emptyset$, must be either a descendant or an ancestor of z (as a direct consequence of the binary-search property). The product P_z in Eq. (11) iterates over z' on the path from root to z. Since the node z is found by following a binary-search path from the root, we can calculate P_z along the way.

We now handle the case when $\alpha \notin \mathit{vals}(T)$. After we reach the final leaf z on the search path, we have $\alpha_z < \alpha < \beta_z$. We conceptually create two children of z: z' and z'' with $I_{z'} = [\alpha_z, \alpha)$ and $I_{z''} = [\alpha, \beta_z)$, and include z'' in $\mathcal{Z}$. Since θ_ω is constant across $\omega \in I_z$, we obtain $p_{I_{z''}}(\boldsymbol{\theta}) = \frac{\beta_z - \alpha}{\beta_z - \alpha_z} \cdot p_{I_z}(\boldsymbol{\theta})$ by applying Eq. (2).

Summarizing the foregoing procedures yields Algorithm 1, which simultaneously constructs the set $\mathcal{Z}$ and calculates the prices $p_{I_z}(\boldsymbol{\theta})$. Since it suffices to go down a single path and only perform constant-time computation in each node, the resulting algorithm runs in time $\mathcal{O}(\log n_{vals})$, where n_{vals} denotes the number of distinct values appeared as endpoints of intervals in all the executed transactions. We defer the proof of Theorem 1, alongside all other proofs from this paper, to the appendix.

Theorem 1. *Algorithm 1 implements* **price**(I, T) *in time* $\mathcal{O}(\log n_{vals})$.

Algorithm 1 Query price of an interval $I = [\alpha, 1)$.

```
Input: Interval $I = [\alpha, 1)$ with $\alpha \in \Omega$.
       LMSR tree $T$, with nodes $z$ annotated with $I_z = [\alpha_z, \beta_z)$, $h_z$, $s_z$ and $S_z$.
Output: Price of bundle security for $I$.

1: Initialize $z \leftarrow root$, $P \leftarrow 1$, $price \leftarrow 0$
2: while $\alpha_z \neq \alpha$ and $z$ is not a leaf do
3:     $P \leftarrow Pe^{s_z/b}$
4:     if $\alpha < \alpha_{right(z)}$ then
5:         $price \leftarrow price + PS_{right(z)}/S_{root}$
6:         $z \leftarrow left(z)$
7:     else
8:         $z \leftarrow right(z)$
9: return $price + \frac{\beta_z - \alpha}{\beta_z - \alpha_z} \cdot PS_z/S_{root}$
```

3.3 Buy Transactions

We next show how to implement **buy**$([\alpha, 1), s, T)$ in time $\mathcal{O}(\log n_{vals})$, while maintaining the LMSR tree properties. The main challenge is to simultaneously maintain *partial-normalization correctness* and *height balance*. We address this by adapting standard AVL-tree rebalancing.

We begin by considering the case $\alpha \in \mathit{vals}(T)$. Similarly to price queries, we conduct binary search for α to obtain the set of nodes $\mathcal{Z}$ that cover $I = [\alpha, 1)$, i.e., $I = \biguplus_{z \in \mathcal{Z}} I_z$. We update the values of s_z across $z \in \mathcal{Z}$ by adding s, and obtain T' that has the same structure as T, but with the updated share quantities

$$s'_z = \begin{cases} s_z + s & \text{if } z \in \mathcal{Z} \\ s_z & \text{otherwise.} \end{cases}$$

Thus, the resulting market state is

$$\boldsymbol{\theta}(T') = \sum_{z \in T'} s'_z \mathbf{1}_{I_z} = \sum_{z \in T} s_z \mathbf{1}_{I_z} + \sum_{z \in \mathcal{Z}} s \mathbf{1}_{I_z} = \boldsymbol{\theta}(T) + s \mathbf{1}_I.$$

It remains to update the partial normalization constants S_z in the tree. Here we rely on the recursive relationship defined in Eq. (8) to update the ancestors of the nodes $z \in \mathcal{Z}$, all of which lie along the search path to α, and each update requires constant time.

When $\alpha \notin \mathit{vals}(T)$, we need to split the leaf z that contains $\alpha \in [\alpha_z, \beta_z)$ before adding shares to *right*(z). This may lead to the violation of the *height-balance property* of T. Similar to the AVL insertion algorithm [17, Section 6.2.3], we fix any imbalance by means of *rotations*, as we go back along the search path. Rotations are operations that modify small portions of the tree, and at most two rotations are needed to rebalance the tree [3]. We further show in Appendix A.2 Lemma 1 that in each rotation, only a constant number of nodes will require updates to preserve the *partial-normalization correctness*. Algorithm 2 describes the full **buy** operation, which runs in time $\mathcal{O}(\log n_{vals})$ thanks to the height balance.

Algorithm 2 Buy s shares of bundle security for an interval $I = [\alpha, 1)$.

```
    Input: Quantity $s \in \mathbb{R}$ and an interval $I = [\alpha, 1)$ with $\alpha \in \Omega$.
           LMSR tree $T$, with nodes $z$ annotated with $I_z = [\alpha_z, \beta_z)$, $h_z$, $s_z$ and $S_z$.
    Output: Tree $T$ updated to reflect the purchase of $s$ shares of bundle security for $I$.
 1: Define subroutines:
        NewLeaf($\alpha_0, \beta_0$): return a new leaf node $z$ with
            $I_z = [\alpha_0, \beta_0)$, $h_z = 0$, $s_z = 0$, $S_z = (\beta_0 - \alpha_0)$
        ResetInnerNode($z$): reset $h_z$ and $S_z$ based on the children of $z$ and the value $s_z$:
            $h_z \leftarrow 1 + \max\{h_{left(z)}, h_{right(z)}\}$, $S_z \leftarrow e^{s_z/b}(S_{left(z)} + S_{right(z)})$
        AddShares($z, s$): increase the number of shares held in $z$ by $s$:
            $s_z \leftarrow s_z + s$, $S_z \leftarrow e^{s/b} S_z$
 2: Initialize $z \leftarrow root$
 3: while $\alpha_z \neq \alpha$ and $z$ is not a leaf do                    ▷ add $s$ shares to $z \in \mathcal{Z}$
 4:     if $\alpha < \alpha_{right(z)}$ then
 5:         AddShares($right(z), s$)
 6:         $z \leftarrow left(z)$
 7:     else
 8:         $z \leftarrow right(z)$
 9: if $\alpha_z < \alpha$ then                                                ▷ split the leaf $z$
10:     $left(z) \leftarrow$ NewLeaf($\alpha_z, \alpha$), $right(z) \leftarrow$ NewLeaf($\alpha, \beta_z$)
11:     $z \leftarrow right(z)$
12: AddShares($z, s$)
13: while $z$ is not a $root$ do          ▷ trace the path back to update $S_z$ and restore height balance
14:     $z \leftarrow par(z)$
15:     if $|h_{left(z)} - h_{right(z)}| \geq 2$ then
16:         Rotate $z$ and possibly one of its children to restore height balance (details in Appendix A.2 Algorithms 5)
17:     ResetInnerNode($z$)
```

Theorem 2. *Algorithm 2 implements* **buy**(I, s, T) *in time* $\mathcal{O}(\log n_{vals})$.

Thus, we have shown that **price**, **cost** and **buy** operations can all be implemented in time $\mathcal{O}(\log n_{vals})$, which is bounded above by the log of the number of **buy** transactions $\mathcal{O}(\log n_{buy})$ as well as the bit precision of the outcome $\mathcal{O}(\log N) = \mathcal{O}(K)$.[6] We note that none of the operations require the knowledge of K, so the market in fact supports queries with arbitrary precision. However, the market precision does affect the worst-case loss bound for the market maker, which is $\mathcal{O}(\log N) = \mathcal{O}(K)$. In the next section, we present a different construction, which achieves a constant worst-case loss independent of the market precision, while the interval operations require time $\mathcal{O}(\log N) = \mathcal{O}(K)$, rather than $\mathcal{O}(\log n_{vals})$.

4 A Multi-resolution Linearly Constrained Market Maker

This section introduces our second contribution, a novel cost-function-based market maker, referred to as the *multi-resolution linearly constrained market maker* (multi-resolution LCMM). The multi-resolution LCMM is based on LMSR, but it enables more flexibility by assigning two or more parallel LMSRs with different liquidity parameters to orchestrate submarkets that offer interval securities at different resolutions. Thus, the multi-resolution LCMM can flexibly interpolate between LMSRs at different resolutions and guarantee a *constant* worst-case loss in sum. However, running submarkets independently can create arbitrage opportunities, as any interval expressible in a coarser market can also be expressed in a finer one. To prevent arbitrage, we design a matrix that imposes linear constraints to tie market prices among different levels and supports the computationally efficient removal of any arbitrage opportunity. We define the multi-resolution LCMM and its properties, and show that **price**, **cost** and **buy** can be implemented in time $\mathcal{O}(\log N)$.

[6] Clearly, $n_{vals} \leq 2n_{buy}$ with each **buy** transaction introducing at most two new endpoint values. The value of n_{vals} is also bounded above by $N + 1$ since the interval endpoints are always in $\Omega \cup \{1\}$.

4.1 A Multi-resolution LCMM for $[0, 1)$

A Multi-resolution Market A binary search tree remains at the core construction of our multi-resolution market. Unlike a log-time LMSR that uses a dynamic tree, it builds upon a *static* one, where each level of the tree represents a submarket of intervals, forming a finer and finer partition of $[0, 1)$. Before introducing the formal setting, we give an example of a market that offers interval securities at two resolutions.

Example 1. Two-level market for $[0, 1)$. We consider two submarkets, indexed by $\mathcal{Z}_1 = \{11, 12\}$ and $\mathcal{Z}_2 = \{21, 22, 23, 24\}$, which partition $[0, 1)$ into interval events at two levels of coarseness:

$$\mathcal{I}_1 : I_{11} = \left[0, \tfrac{1}{2}\right), I_{12} = \left[\tfrac{1}{2}, 1\right); \qquad \mathcal{I}_2 : I_{21} = \left[0, \tfrac{1}{4}\right), I_{22} = \left[\tfrac{1}{4}, \tfrac{1}{2}\right), I_{23} = \left[\tfrac{1}{2}, \tfrac{3}{4}\right), I_{24} = \left[\tfrac{3}{4}, 1\right).$$

Thus, the market provides six interval securities in total that are associated with their corresponding interval events (i.e., $\mathcal{I} = \mathcal{I}_1 \biguplus \mathcal{I}_2$ and $|\mathcal{I}| = 6$). We index the securities by $z \in \mathcal{Z}$, where $\mathcal{Z} = \mathcal{Z}_1 \biguplus \mathcal{Z}_2 = \{11, 12, 21, 22, 23, 24\}$. Each security pays out $\phi_z(\omega) = 1\{\omega \in I_z\}$. □

Extending the example to multiple resolutions, we represent the initial independent submarkets with a *complete binary tree* T^* of depth K, which corresponds to the bit precision of the outcome ω. Let $\mathcal{Z}^*$ denote the set of nodes of T^*, and $\mathcal{Z}_k$ for $k \in \{0, 1, \ldots, K\}$ the set of nodes at each level. $\mathcal{Z}_0$ contains the root node associated with $I_{root} = [0, 1)$, and each *consecutive* level contains the children of nodes from the previous level, which split their corresponding parent intervals exactly in half. Thus, level k forms a partition of $[0, 1)$ into 2^k intervals of size 2^{-k}, and the final level $\mathcal{Z}_K$ contains $N = 2^K$ leaves.

The multi-resolution market offers interval securities indexed by nodes, and their payoffs are defined by the corresponding intervals I_z, i.e., $\phi_z(\omega) = 1\{\omega \in I_z\}$. We partition the securities into submarkets corresponding to levels, i.e., $\mathcal{I}_k = \mathcal{Z}_k$ for $k \le K$, where $|\mathcal{I}_k| = 2^k$ and $\mathcal{I} = \biguplus_{k \le K} \mathcal{I}_k$. For each level, we define the LMSR cost function C_k with a *separate* liquidity parameter $b_k > 0$:

$$C_k(\boldsymbol{\theta}_k) = b_k \log\left(\sum_{z \in \mathcal{Z}_k} e^{\theta_z / b_k}\right).$$

A Linearly Constrained Market Maker Recall that $\boldsymbol{\phi} : \Omega \to \mathbb{R}^{|\mathcal{I}|}$ is the payoff function, and $\boldsymbol{\theta} \in \mathbb{R}^{|\mathcal{I}|}$ is the market state vector. Following the multi-resolution market construction, $\mathcal{I}_k$ describes a block of securities that is treated as a submarket at level k for $0 < k \le K$. We write $\boldsymbol{\phi}_k(\omega) := (\phi_z(\omega))_{z \in \mathcal{Z}_k}$ and $\boldsymbol{\theta}_k$ for each of the submarket. We adopt bounded-loss and arbitrage-free convex cost functions $C_k : \mathbb{R}^{|\mathcal{I}_k|} \to \mathbb{R}$ for each submarket, which combines into a *direct-sum cost* $\tilde{C}(\boldsymbol{\theta}) = \sum_{k \le K} C_k(\boldsymbol{\theta}_k)$ for the whole market. This corresponds to pricing securities in each block $\mathcal{I}_k$ independently using C_k. However, if there are logical dependencies between securities in different levels, independent pricing may lead to the lack of price consistency among submarkets and create arbitrage opportunities.

Example 2. Arbitrage in two-level market. Continuing Example 1, we define separate LMSR costs, where $b_1 = 1$ and $b_2 = 1$:

$$C_1(\boldsymbol{\theta}_1) = \log\left(e^{\theta_{11}} + e^{\theta_{12}}\right); \qquad C_2(\boldsymbol{\theta}_2) = \log\left(e^{\theta_{21}} + e^{\theta_{22}} + e^{\theta_{23}} + e^{\theta_{24}}\right).$$

The direct-sum market $\tilde{C}(\boldsymbol{\theta}) = C_1(\boldsymbol{\theta}_1) + C_2(\boldsymbol{\theta}_2)$ may give rise to incoherent prices. For example, after buying some shares of security ϕ_{21} associated with $I_{21} = \left[0, \frac{1}{4}\right)$ from submarket $\mathcal{I}_2$, the market can end up with

$$\tilde{p}_{I_{11} = [0, \frac{1}{2})}(\boldsymbol{\theta}) = 0.5; \qquad \tilde{p}_{I_{21} = [0, \frac{1}{4})}(\boldsymbol{\theta}) + \tilde{p}_{I_{22} = [\frac{1}{4}, \frac{1}{2})}(\boldsymbol{\theta}) = 0.6.$$

This violates the *no arbitrage* property that requires, in our two-level market case, $\mathbb{P}[I_{11}] = \mathbb{P}[I_{21}] + \mathbb{P}[I_{22}]$ and $\mathbb{P}[I_{12}] = \mathbb{P}[I_{23}] + \mathbb{P}[I_{24}]$ under any probability distribution over Ω. We specify the linear price constraints $\mu_{11} - \mu_{21} - \mu_{22} = 0$ and $\mu_{12} - \mu_{23} - \mu_{24} = 0$ by vectors

$$\mathbf{a}_1 = (1, 0, -1, -1, 0, 0)^\top; \qquad \mathbf{a}_2 = (0, 1, 0, 0, -1, -1)^\top.$$

The vectors together form the constraint matrix $\mathbf{A} = (\mathbf{a}_1\, \mathbf{a}_2) \in \mathbb{R}^{|\mathcal{I}| \times |\mathcal{J}|}$, where $\mathcal{J} = \mathcal{I} \backslash \mathcal{I}_K$. □

We extend Example 2 to specify price constraints in a multi-resolution market to achieve the *no arbitrage* property. Recall that $\mathcal{M}$ denotes a *coherent price space*, where any expected payoff lies in the convex hull of $\{\boldsymbol{\phi}(\omega)\}_{\omega\in\Omega}$. It is always polyhedral and can be described by a set of linear inequalities, which can be used for arbitrage removal [10]. Arbitrage opportunities arise whenever prices fall outside the set of coherent prices $\mathcal{M}$ [1]. For the multi-resolution market, we specify a set of *homogeneous linear equalities* describing a superset of $\mathcal{M}$. Equalities are indexed by $j \in \mathcal{J}$ and described by a matrix $\mathbf{A} \in \mathbb{R}^{|\mathcal{I}|\times|\mathcal{J}|}$, such that

$$\mathcal{M} \subseteq \{\boldsymbol{\mu} \in \mathbb{R}^{|\mathcal{I}|} :\ \mathbf{A}^\top \boldsymbol{\mu} = \mathbf{0}\}. \tag{12}$$

Specifically, we design the constraint matrix $\mathbf{A}$ to ensure that any pair of submarkets is price coherent in a multi-resolution market, meaning that any interval event $I \subseteq \Omega$ gets the same price on all levels that can express it. Therefore, for each *inner node* $y \in \mathcal{Z}_l$ where $l < K$, we have

$$\mu_y = \sum_{z\in\mathcal{Z}_k:\, z\subset y} \mu_z \qquad \text{for any } l < k \le K.$$

To facilitate the implementation in a binary tree, we further tie the price of y to the prices of *all* of y's descendants and weight each level by its liquidity parameter b_k:

$$\underbrace{\Big(\sum_{k>\ell} b_k\Big)}_{B_\ell} \mu_y = \sum_{k>\ell}\Big(b_k \sum_{z\in\mathcal{Z}_k:\, z\subset y} \mu_z\Big). \tag{13}$$

This design turns out to be more algorithmically convenient to remove arbitrage and restore price consistency (as we will see in Section 4.3). Now we formally define the constraint matrix $\mathbf{A}$. Let $\mathcal{Y}^* = \mathcal{Z}^*\backslash\mathcal{Z}_K$ be the set of inner nodes of T^* and let *level*(z) denote the level of a node z. The matrix $\mathbf{A} \in \mathbb{R}^{|\mathcal{Z}^*|\times|\mathcal{Y}^*|}$ contains the constraints from Eq. (13) across all $y \in \mathcal{Y}^*$:

$$A_{zy} = \begin{cases} B_{level(z)} & \text{if } z = y, \\ -b_{level(z)} & \text{if } z \subset y, \\ 0 & \text{otherwise.} \end{cases} \tag{14}$$

We refer to the *linearly constrained market maker* with the above matrix $\mathbf{A}$ as the *multi-resolution LCMM*. The derivation above shows that the constraints in the matrix $\mathbf{A}$ are *necessary* to assure no arbitrage. Next theorem shows that they are also sufficient. The proof shows that consecutive levels are coherent, which by transitivity implies that the overall price vector is coherent (see Appendix A.3 for details).

Theorem 3. *A multi-resolution LCMM is arbitrage-free.*

The multi-resolution LCMM also enjoys the *bounded loss* property. For a suitable choice of liquidities, such as $b_k = \mathcal{O}(1/k^{2.01})$, it can achieve a constant worst-case loss bound. The proof uses the fact that the loss of LCMM is bounded by the sum of losses of level markets, which are at most $b_k \log|\mathcal{Z}_k| = kb_k \log 2$.

Theorem 4. *Let $\{b_k\}_{k=1}^\infty$ be a sequence of positive numbers such that $\sum_{k=1}^\infty kb_k = B^*$ for some finite B^*. Then the multi-resolution LCMM with liquidity parameters b_k for $k \le K$ guarantees the worst-case loss of the market maker of at most $B^* \log 2$, regardless of the outcome precision K.*

By the design of matrix $\mathbf{A}$, arbitrage opportunities appear if the price of bundle $\mathbf{a}_j$ differs from zero, where $\mathbf{a}_j$ denotes the jth column of $\mathbf{A}$. Traders profit by buying a positive quantity of $\mathbf{a}_j$ if its price is negative, and selling if it is positive. Thus, the constraint matrix $\mathbf{A}$ also provides a recipe on arbitrage removals.

Example 3. Arbitrage Removal by LCMM in two-level market. Continuing Example 2, we have prices $\tilde{\boldsymbol{p}}(\boldsymbol{\theta})$ violate the constraint matrix $\mathbf{A}$, i.e., $\mathbf{a}_1^\top \tilde{\boldsymbol{p}}(\boldsymbol{\theta}) = \tilde{p}_{11}(\boldsymbol{\theta}) - \tilde{p}_{21}(\boldsymbol{\theta}) - \tilde{p}_{22}(\boldsymbol{\theta}) = 0.5 - 0.6 \neq 0$. Thus, $\mathbf{a}_1$ reveals a profitable arbitrage opportunity: buy the security ϕ_{11} (at the initial price 0.5) and simultaneously sell the securities ϕ_{21} and ϕ_{22} (at the initial price 0.6). This increases the price of ϕ_{11} and decreases the prices of ϕ_{21} and ϕ_{22}. After a sufficiently large quantity s of shares of ϕ_{11} is bought (and the same quantities of ϕ_{21} and ϕ_{22} are sold), $\mathbf{a}_1^\top \tilde{\boldsymbol{p}}(\tilde{\boldsymbol{\theta}}) = 0$ is achieved in a new state $\tilde{\boldsymbol{\theta}} = \boldsymbol{\theta} + s\mathbf{a}_1 = \boldsymbol{\theta} + \mathbf{A}\boldsymbol{\eta}$, where $\boldsymbol{\eta} := (s, 0)^\top$. Therefore, after each update of $\boldsymbol{\theta}$, we use an LCMM who follows the constraint matrix $\mathbf{A}$ to automatically find a new state $\tilde{\boldsymbol{\theta}} = \boldsymbol{\theta} + s\mathbf{a}_1 + t\mathbf{a}_2 = \boldsymbol{\theta} + \mathbf{A}\boldsymbol{\eta}$ where $\boldsymbol{\eta} := (s, t)^\top$, such that $\mathbf{a}_1^\top \tilde{\boldsymbol{p}}(\tilde{\boldsymbol{\theta}}) = 0$ and $\mathbf{a}_2^\top \tilde{\boldsymbol{p}}(\tilde{\boldsymbol{\theta}}) = 0$ hold. □

We generalize Example 3 to the multi-resolution market. Formally, an LCMM is described by the cost function

$$C(\boldsymbol{\theta}) = \inf_{\boldsymbol{\eta}\in\mathbb{R}^{|\mathcal{J}|}} \tilde{C}(\boldsymbol{\theta} + \mathbf{A}\boldsymbol{\eta}). \tag{15}$$

It relies on the direct-sum cost $\tilde{C}$, but with each trader purchase $\boldsymbol{\delta}$ that causes inconsistent prices, an LCMM automatically seeks the most advantageous cost for the trader by buying bundles $\mathbf{A}\boldsymbol{\delta}_{\text{arb}}$ on the trader's behalf to remove arbitrage. Trader purchases are accumulated as the state $\boldsymbol{\theta}$, and automatic purchases made by the LCMM are accumulated as $\mathbf{A}\boldsymbol{\eta}$. The purchase of bundle $\mathbf{A}\boldsymbol{\delta}_{\text{arb}}$ has no effect on the trader's payoff, because $(\mathbf{A}\boldsymbol{\delta}_{\text{arb}})^\top\boldsymbol{\phi}(\omega) = 0$ for all $\omega \in \Omega$ thanks to Eq. (12) and the fact that $\boldsymbol{\phi}(\omega) \in \mathcal{M}$. However, the purchase of $\mathbf{A}\boldsymbol{\delta}_{\text{arb}}$ can lower the cost, so optimizing over $\boldsymbol{\delta}_{\text{arb}}$ benefits the traders, while maintaining the same worst-case loss guarantee for the market maker as $\tilde{C}$ [10].

Consider a fixed $\boldsymbol{\theta}$ and the corresponding $\boldsymbol{\eta}^\star$ minimizing Eq. (15). We calculate prices by LCMM as

$$\boldsymbol{p}(\boldsymbol{\theta}) = \nabla C(\boldsymbol{\theta}) = \nabla\tilde{C}(\boldsymbol{\theta} + \mathbf{A}\boldsymbol{\eta}^\star).$$

By the first order optimality, $\boldsymbol{\eta}^\star$ minimizes Eq. (15) if and only if $\mathbf{A}^\top\big(\nabla\tilde{C}(\boldsymbol{\theta} + \mathbf{A}\boldsymbol{\eta}^\star)\big) = \mathbf{0}$. This means that $\mathbf{A}^\top\boldsymbol{p}(\boldsymbol{\theta}) = \mathbf{0}$, and thus arbitrage opportunities expressed by $\mathbf{A}$ are completely removed by the LCMM cost function C. To implement an LCMM, we maintain the state $\tilde{\boldsymbol{\theta}} = \boldsymbol{\theta} + \mathbf{A}\boldsymbol{\eta}$ in the direct-sum market $\tilde{C}$. After updating $\boldsymbol{\theta}$ to a new value $\boldsymbol{\theta}' = \boldsymbol{\theta} + \boldsymbol{\delta}$, we seek to find $\boldsymbol{\eta}' = \boldsymbol{\eta} + \boldsymbol{\delta}_{\text{arb}}$ that removes all the arbitrage opportunities expressed by $\mathbf{A}$. The resulting cost for the trader is

$$\tilde{C}(\boldsymbol{\theta}' + \mathbf{A}\boldsymbol{\eta}') - \tilde{C}(\boldsymbol{\theta} + \mathbf{A}\boldsymbol{\eta}) = \tilde{C}(\tilde{\boldsymbol{\theta}} + \boldsymbol{\delta} + \mathbf{A}\boldsymbol{\delta}_{\text{arb}}) - \tilde{C}(\tilde{\boldsymbol{\theta}}).$$

A Multi-resolution LCMM tree Combining the above constructions and annotations, we can now formally define the multi-resolution LCMM tree. The market state of a multi-resolution LCMM is represented by vectors $\boldsymbol{\theta} \in \mathbb{R}^{|\mathcal{Z}^*|}$ and $\boldsymbol{\eta} \in \mathbb{R}^{|\mathcal{Y}^*|}$, whose dimensions can be intractably large (e.g., on the order of $2^K = N$). However, since each LCMM operation involves only a small set of coordinates of $\boldsymbol{\theta}$ and $\boldsymbol{\eta}$, we keep track of these coordinates accessed so far by organizing an annotated subtree T of T^*, referred to as an *LCMM tree*:

Definition 2. *An* LCMM tree T *is a full binary tree, where each node z is annotated with $I_z = [\alpha_z, \beta_z)$, $\theta_z \in \mathbb{R}$, $\eta_z \in \mathbb{R}$, such that $I_{root} = [0, 1)$, and for every inner node z:*

$$\alpha_z = \alpha_{left(z)}, \quad \beta_{left(z)} = \alpha_{right(z)} = \frac{\alpha_z + \beta_z}{2}, \quad \beta_{right(z)} = \beta_z.$$

The tree T contains the coordinates of $\boldsymbol{\theta}$ and $\boldsymbol{\eta}$ accessed so far. Since $\boldsymbol{\theta}$ and $\boldsymbol{\eta}$ are initialized to zero, their remaining entries are zero. We write $\boldsymbol{\theta}(T) \in \mathbb{R}^{|\mathcal{Z}^*|}$ and $\boldsymbol{\eta}(T) \in \mathbb{R}^{|\mathcal{Y}^*|}$ for the vectors represented by T. In order to accurately generate LCMM prices, we will maintain $\boldsymbol{\eta}(T)$ that minimizes Eq. (15), or equivalently, $\boldsymbol{\eta}(T)$ that satisfies

$$\mathbf{A}^\top\tilde{\boldsymbol{p}}\big(\boldsymbol{\theta}(T) + \mathbf{A}\boldsymbol{\eta}(T)\big) = \mathbf{0}.$$

If this property holds, we say that an LCMM tree T is *coherent*.

Similar to the LMSR market of Section 3, we seek efficient implementations of $\textbf{price}(I, T)$, $\textbf{cost}(I, s, T)$ and $\textbf{buy}(I, s, T)$ for any interval $I = [\alpha, \beta)$, with $\alpha, \beta \in \Omega \cup \{1\}$. However, unlike Section 3, additional challenges arise because of the need to maintain price coherence.

4.2 Price Queries

In a multi-resolution market, there are many ways to decompose an interval I, but they all yield the same price thanks to coherence. The *no arbitrage* property also guarantees that the price of $[\alpha, \beta)$ can be obtained by subtracting the price of $[\beta, 1)$ from $[\alpha, 1)$. Therefore, we focus on pricing intervals of the form $I = [\alpha, 1)$.

Let T be a coherent LCMM tree, and $\boldsymbol{\theta} := \boldsymbol{\theta}(T)$ and $\boldsymbol{\eta} := \boldsymbol{\eta}(T)$ the vectors represented by T. Let $\tilde{\boldsymbol{\theta}} = \boldsymbol{\theta} + \mathbf{A}\boldsymbol{\eta}$ be the corresponding state in $\tilde{C}$, so the current security prices are $\boldsymbol{\mu} := \tilde{\boldsymbol{p}}(\tilde{\boldsymbol{\theta}})$. We proceed similarly as for the log-time LMSR by identifying a set of nodes $\mathcal{Z}$ such that $I = \biguplus_{z\in\mathcal{Z}} I_z$ and $\tilde{p}_I(\tilde{\boldsymbol{\theta}}) = \sum_{z\in\mathcal{Z}} \mu_z$. We then rely on price coherence to calculate each μ_z along the search path.

Specifically, assume that z is not a root node and we know the price of its parent. Let $sib(z)$ denote the sibling of z. We relate the price of z to the price of $par(z)$ as follows:

$$\mu_z = \frac{\mu_z}{\mu_{par(z)}} \cdot \mu_{par(z)} = \frac{\mu_z}{\mu_z + \mu_{sib(z)}} \cdot \mu_{par(z)} \tag{16}$$

$$= \frac{e^{\tilde{\theta}_z/b_k}}{e^{\tilde{\theta}_z/b_k} + e^{\tilde{\theta}_{sib(z)}/b_k}} \cdot \mu_{par(z)}, \tag{17}$$

where Eq. (16) follows by price coherence and Eq. (17) follows by price calculation in Eq. (1). Thus, we can descend the search path to calculate the price μ_z, beginning with $\mu_{root} = 1$ and calculating μ_y for all nodes y along the path. It remains to obtain $\tilde{\theta}_z$, which we follow the construction of $\mathbf{A}$ in Eq. (14) to calculate

$$\tilde{\theta}_z = \theta_z + \sum_{y \in \mathcal{Y}^*} A_{zy}\eta_y = \theta_z + B_{level(z)}\eta_z - b_{level(z)} \sum_{y \supset z} \eta_y. \tag{18}$$

Plugging the above equation back in Eq. (17) and using the fact that $k = level(z)$, we obtain[7]

$$\mu_z = \frac{\exp\{(\theta_z + B_k\eta_z)/b_k\}}{\exp\{(\theta_z + B_k\eta_z)/b_k\} + \exp\{(\theta_{sib(z)} + B_k\eta_{sib(z)})/b_k\}}. \tag{19}$$

Combining all of this yields Algorithm 3. In the final line of the algorithm, we address the case when the search path ends in the leaf z with $\alpha_z < \alpha < \beta_z$. In this case, rather than expanding the tree to the lowest level, we use price coherence again: since any strict descendant $z' \subset z$ on the path from z to a leaf node $u \in \mathcal{Z}_K$ has $\theta_{z'} = \eta_{z'} = 0$ by market initialization, all leaf nodes u have the same price by Eq. (18). Therefore, the price of $[\alpha, \beta_z)$ equals $\frac{\beta_z - \alpha}{\beta_z - \alpha_z} \cdot \mu_z$.

As the length of any search path in T is at most K, the running time of Algorithm 3 is $\mathcal{O}(K)$. More precisely, the time to price $I = [\alpha, 1)$ is $\mathcal{O}(prec(\alpha))$, where we use $prec(\alpha)$ to denote the bit precision of α, defined as the smallest integer k such that α is an integer multiple of 2^{-k}.

Theorem 5. *Let $I = [\alpha, 1)$, $\alpha \in \Omega$. Algorithm 3 implements* **price**(I, T) *in time $\mathcal{O}(prec(\alpha))$.*

Algorithm 3 Query price of an interval $I = [\alpha, 1)$.

```
Input: Interval $I = [\alpha, 1)$ with $\alpha \in \Omega$.
       Coherent LCMM tree $T$, with nodes $z$ annotated with $I_z = [\alpha_z, \beta_z)$, $\theta_z$, $\eta_z$.
Output: Price of bundle security for $I$.

1: Initialize $z \leftarrow root$, $\mu_z \leftarrow 1$, $price \leftarrow 0$
2: while $\alpha_z \neq \alpha$ and $z$ is not a leaf do
3:     $z_l \leftarrow left(z)$, $z_r \leftarrow right(z)$, $k \leftarrow level(z_l)$
4:     $e_l \leftarrow \exp\{(\theta_{z_l} + B_k\eta_{z_l})/b_k\}$, $e_r \leftarrow \exp\{(\theta_{z_r} + B_k\eta_{z_r})/b_k\}$, $\mu_{z_l} \leftarrow \frac{e_l}{e_l+e_r}\mu_z$, $\mu_{z_r} \leftarrow \frac{e_r}{e_l+e_r}\mu_z$
5:     if $\alpha < \alpha_{right(z)}$ then
6:         $price \leftarrow price + \mu_{z_r}$
7:         $z \leftarrow z_l$
8:     else
9:         $z \leftarrow z_r$
10: return $price + \frac{\beta_z - \alpha}{\beta_z - \alpha_z} \cdot \mu_z$
```

4.3 Buy and Cost Operations

Different from cost queries for LMSR, the cost query for multi-resolution LCMM cannot be directly derived from prices. We instead augment the **buy** operation to return the cost of bought shares. Thus, we can implement **cost** by executing **buy** and then reverting all the changes in the tree. As we will show in Algorithm 4,

[7] The factor $\exp\{-\sum_{y \supset z} \eta_y\} = \exp\{-\sum_{y \supset sib(z)} \eta_y\}$ appears in both the numerator and the denominator after plugging Eq. (18) to Eq. (17), so it cancels out.

buy operations run in time $\mathcal{O}(K)$, so changes can be reverted in time $\mathcal{O}(K)$ (e.g., by logging the **buy** updates and executing them backwards). As before, we focus on $\mathbf{buy}(I, s, T)$ for intervals of the form $I = [\alpha, 1)$. By buying s shares of $[\alpha, 1)$ and then $(-s)$ shares of $[\beta, 1)$, we obtain buying $[\alpha, \beta)$. Similar to the price query, we start with a set of nodes $\mathcal{Z}$ that partition I, and buy s shares of each security ϕ_y for $y \in \mathcal{Z}$.

Consider one of such nodes $y \in \mathcal{Z}$ at level $\ell := \mathit{level}(y)$. Increasing θ_y by s creates price incoherence between the submarket at level ℓ and submarkets at all the other levels. We first remove any arbitrage opportunity appeared between level ℓ and lower levels with $k > \ell$. We show in Appendix A.6 Lemma 3 that in order to restore coherence, it suffices to update η_y by a suitable closed-form amount:

$$t = \frac{b_\ell}{B_{\ell-1}} \log\left(\frac{1-\mu_y}{\mu_y} \cdot \frac{\mu_{\mathit{left}(y)} + \mu_{\mathit{right}(y)}}{1 - \mu_{\mathit{left}(y)} - \mu_{\mathit{right}(y)}}\right). \tag{20}$$

This key algorithmic step is enabled by the specific structure of the arbitrage bundle $\mathbf{a}_y$, which corresponds to buying ϕ_y on the level ℓ while selling securities associated with all the descendants of y, appropriately weighted by their respective liquidity values as specified in the constraint matrix $\mathbf{A}$.

The market then remains incoherent between ℓ and upper levels $k < \ell$. Since the updates have been localized to the subtree rooted at y, we use Lemma 3 again to update $\eta_{\mathit{par}(y)}$ and restore coherence among all levels $k \geq \ell - 1$. We continue in this manner back along the path to root to restore a coherent market.

Algorithm 4 Buy s shares of bundle security for an interval $I = [\alpha, 1)$.

```
Input: Quantity $s \in \mathbb{R}$ and an interval $I = [\alpha, 1)$ with $\alpha \in \Omega$.
       Coherent LCMM tree $T$, with nodes $z$ annotated with $I_z = [\alpha_z, \beta_z)$, $\theta_z$, $\eta_z$.
Output: Cost of $s$ shares of bundle security for $I$, and the tree $T$ updated to reflect the trade.

1: Initialize global variable $cost \leftarrow 0$        ▷ track the cost incurred by the transaction
2: Initialize $z \leftarrow root$, $\mu_z \leftarrow 1$
3: while $\alpha_z \neq \alpha$ do        ▷ search for $\alpha$ and calculate prices $\mu_z$ along the path
4:     if $z$ is a leaf then
5:         $left(z) \leftarrow$ NewLeaf($\alpha_z$, $\frac{1}{2}(\alpha_z + \beta_z)$), $right(z) \leftarrow$ NewLeaf($\frac{1}{2}(\alpha_z + \beta_z)$, $\beta_z$)
6:     $z_l \leftarrow left(z)$, $z_r \leftarrow right(z)$, $k \leftarrow level(z_l)$
7:     $e_l \leftarrow \exp\{(\theta_{z_l} + B_k\eta_{z_l})/b_k\}$, $e_r \leftarrow \exp\{(\theta_{z_r} + B_k\eta_{z_r})/b_k\}$, $\mu_{z_l} \leftarrow \frac{e_l}{e_l+e_r}\mu_z$, $\mu_{z_r} \leftarrow \frac{e_r}{e_l+e_r}\mu_z$
8:     if $\alpha < \alpha_{right(z)}$ then
9:         $z \leftarrow z_l$
10:    else
11:        $z \leftarrow z_r$
12: AddShares($z, s$)
13: while $z$ is not a $root$ do        ▷ add shares to $z \in \mathcal{Z}$ and remove arbitrage level by level up the search path
14:     $z' \leftarrow sib(z)$, $y \leftarrow par(z)$
15:     if $z' = right(y)$ then
16:         AddShares($z', s$)
17:     RemoveArbitrage($y$, $\mu_z + \mu_{z'}$)
18:     $z \leftarrow y$
19: return $cost$

20: function NewLeaf($\alpha_0, \beta_0$):
21:     return a new leaf node $z$ with $I_z = [\alpha_0, \beta_0)$, $\theta_z = 0$, $\eta_z = 0$
22: procedure RemoveArbitrage($y, \mu_{other}$):
23:     Let $\ell = level(y)$, $y' = sib(y)$
24:     Let $t = \frac{b_\ell}{B_{\ell-1}} \log\left(\frac{1-\mu_y}{\mu_y} \cdot \frac{\mu_{other}}{1-\mu_{other}}\right)$, $S = \mu_y e^{tB_\ell/b_\ell} + 1 - \mu_y$, $S_{other} = \mu_{other} e^{-t} + 1 - \mu_{other}$
25:     $\eta_y \leftarrow \eta_y + t$, $\mu_y \leftarrow \mu_y e^{tB_\ell/b_\ell}/S$, $\mu_{y'} \leftarrow \mu_{y'}/S$
26:     $cost \leftarrow cost + (b_\ell \log S) + (B_\ell \log S_{other})$
27: procedure AddShares($z, s$):
28:     Let $\ell = level(z)$, $z' = sib(z)$, $\mu_{other} = \mu_z$, $S = \mu_z e^{s/b_\ell} + 1 - \mu_z$
29:     $\theta_z \leftarrow \theta_z + s$, $\mu_z \leftarrow \mu_z e^{s/b_\ell}/S$, $\mu_{z'} \leftarrow \mu_{z'}/S$
30:     $cost \leftarrow cost + (b_\ell \log S)$
31:     RemoveArbitrage($z, \mu_{other}$)
```

We summarize the detailed procedure in Algorithm 4, which performs $\mathbf{buy}(I, s, T)$ and simultaneously keeps track of $\mathbf{cost}(I, s, T)$. The algorithm first executes the binary search for α, and calculates prices μ_z along the way (lines 2–11). It then proceeds back up the search path, adding s shares to nodes within the cover $\mathcal{Z}$ (lines 12 and 16). With each addition of shares, the arbitrage with respect to finer (or lower) levels needs to be removed (line 31). While returning up the search path, the arbitrage with respect to the next level up is also removed (line 17). The RemoveArbitrage procedure (lines 22–26) updates η_y by adding t, as calculated in Eq. (20) and proved in Lemma 3. Throughout the execution, the algorithm also calculates the total cost of the **buy** transaction (lines 26 and 30) by evaluating Eq. (3) in the component submarkets with C_k. Note that costs in all submarkets with $k > \ell$ can be evaluated simultaneously thanks to the restored coherence and the structure of $\mathbf{a}_y$ (line 26). Similar to price queries, Algorithm 4 runs in time $\mathcal{O}(prec(\alpha))$.

Theorem 6. *Let $I = [\alpha, 1)$, $\alpha \in \Omega$. Algorithm 4 implements a simultaneous* $\mathbf{buy}(I, s, T)$ *and* $\mathbf{cost}(I, s, T)$ *in time $\mathcal{O}(prec(\alpha))$.*

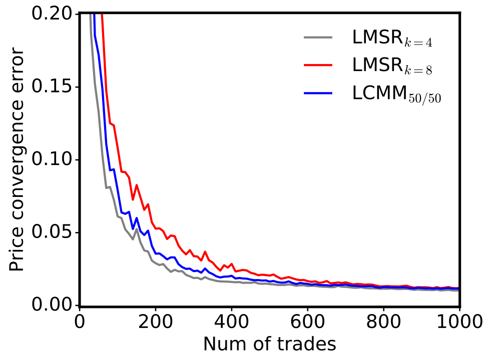
In Algorithms 3 and 4, we assume that each node z can store a scalar μ_z, which can be modified and persist during the run of the algorithm to support price calculations, but are disposed afterwards. The only part of our algorithm that explicitly depends on K are the cumulative liquidities $B_\ell = \sum_{k=\ell+1}^{K} b_k$. To remove such dependence, we can instead use $B'_\ell = \sum_{k=\ell+1}^{\infty} b_k = B^* - \sum_{k=1}^{\ell} b_k$, where $B^* = \sum_{k=1}^{\infty} b_k$. This has no impact on the correctness of our algorithms: if at a given time the largest level in the tree T is L, we can simply view T as implementing a multi-resolution LCMM with $K = L + 1$ and liquidities $b_1, b_2, \dots, b_L, B'_L$. The last level $K = L + 1$ then corresponds to infinitely many mutually coherent markets $\{C_k\}_{k=L+1}^{\infty}$. Thus, a multi-resolution LCMM can achieve a constant loss bound regardless of K and support market operations for $I = [\alpha, \beta)$ in time $\mathcal{O}(prec(\alpha) + prec(\beta))$, which equals the number of bits required to describe I.

5 Numerical Experiments

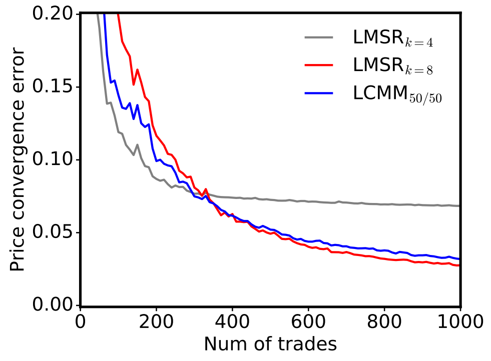
We have shown that both the log-time LMSR and multi-resolution LCMM can support fast betting on interval securities. In this section, we empirically highlight the greater flexibility of LCMM by showing how LCMM can interpolate between LMSRs at different resolutions. To motivate the experiment, we note that formally LCMM subsumes LMSR, but affords additional flexibility for the market designer. For example, an LMSR that operates at precision $k = 4$ with liquidity b can be represented by an LCMM with the level liquidity values $\mathbf{b} = (0, 0, 0, b, 0, 0, \dots)$. However, LCMM allows more flexible liquidity attenuation according to the information-gathering objective. For example, if the market expects most of the information at precision 4, but also wants to support bets up to precision 8, one could run an LCMM with the liquidity placed at two levels as $\mathbf{b} = (0, 0, 0, b_4, 0, 0, 0, b_8)$. By choosing different values b_4 and b_8, the market designer can express utility for information at different precision levels.

We conduct agent-based simulation using the trader model with exponential utility and exponential-family beliefs [2]. Agents trade with a market maker (either a LMSR or a multi-resolution LCMM) to bet on intervals within $[0, 1)$, following the dynamics described below. We note that while our market makers support agents with any beliefs and utility functions, the exponential trader model is convenient, because it allows a closed-form derivation of *market-clearing price*, meaning the clearing price reached when agents only trade among themselves, without a market maker [2,11]. This can be viewed as a "ground truth" for the information elicitation. We evaluate market makers in terms of their *price convergence error*, calculated as the relative entropy between the market-clearing price and the price maintained by the market maker.

Trading Dynamics We simulate a market consisting of ten traders. The outcome space is $[0, 1)$, discretized at the precision $K = 10$. Traders, indexed as $i \in \{1, \dots, 10\}$, have noisy access to the underlying true signal $p = 0.4$. Trader i's belief takes form of a beta distribution $\mathrm{Beta}(a_i, b_i)$ with $a_i \sim \mathrm{Binomial}(p, n_i)$, $b_i = n_i - a_i$, and $n_i = 16i$ representing the quality of the agent's observation of the signal p. Each trader i has an exponential utility $u_i(W) = -e^{-W}$, where W is the trader's wealth. We consider budget-limited cost-based market makers, whose worst-case loss may not exceed a budget constraint B. For LMSR at precision k, this means setting the liquidity parameter to $b = B/\log(2^k)$. In our experiments, we consider two LMSR markets at precision levels 4 and 8, denoted as $\texttt{LMSR}_{k=4}$ and $\texttt{LMSR}_{k=8}$. On the other hand, a multi-resolution LCMM



(a) Price convergence error at precision $k = 4$.



(b) Price convergence error at precision $k = 8$.

Fig. 1: The price convergence error, calculated as the relative entropy between the market-clearing price and the price maintained by the respective market maker, measured at two resolution levels.

has an infinite number of choices for its liquidity at each precision level. To showcase its interpolation ability, we consider LCMM that evenly splits its budget to precision levels 4 and 8, and denote it as $\texttt{LCMM}_{50/50}$.

Each market starts with the uniform prior, i.e., the initial market prices for all outcomes are equal. In each time step, a uniformly random agent is picked to trade. The selected agent considers a set of 50 interval securities, with endpoints randomly sampled according to the agent's belief. The candidate intervals are rounded to the precision of the corresponding market. The agent considers trading the expected-utility-optimizing number of shares for each interval, and ultimately picks the best interval and executes the trade.

Results Following the described protocol, we run markets mediated by the three respective market makers, $\texttt{LMSR}_{k=4}$, $\texttt{LMSR}_{k=8}$, and $\texttt{LCMM}_{50/50}$, over a range of budget constraints. To decrease variance, we generate 40 simulation traces (described by a sequence of agent arrivals and their draws of the candidate intervals) and run the market makers on those same traces. Fig. 1 shows the results for the budget $B = 1$ (see Appendix B for results at different budget levels). We plot the price convergence error at different resolution levels as a function of the number of trades, averaged over 40 simulation traces. As one may expect, $\texttt{LMSR}_{k=4}$ achieves a faster price convergence at the coarser precision level $k = 4$ compared to $\texttt{LMSR}_{k=8}$ (Fig. 1a), but fails to elicit information at any finer granularity by design.[8] The proposed $\texttt{LCMM}_{50/50}$, by equally splitting the budget between $k = 4$ and $k = 8$, is able to interpolate between the performance of $\texttt{LMSR}_{k=4}$ and $\texttt{LMSR}_{k=8}$, and achieves the "best of both worlds": it can elicit forecasts at the finer level $k = 8$ similarly to $\texttt{LMSR}_{k=8}$, but also obtain a fast convergence at the coarser level $k = 4$, almost matching the convergence speed of $\texttt{LMSR}_{k=4}$.

6 Conclusion

We have proposed efficient implementations of two market makers that support trading interval securities of arbitrary precision. Both of our implementations are exponentially faster than previous approaches. Our *log-time LMSR market maker* uses a balanced tree to implement market operations in a standard LMSR market in time that is only logarithmic in the number of distinct interval endpoints appearing in the trading history. Our *multi-resolution LCMM* is a new market maker that runs multiple LMSR markets at different resolutions with different liquidities, and as a result has more flexibility to capture market administrator's preferences in eliciting information at different levels of granularity. Moreover, this design allows achieving a constant worst-case loss bound.

Two natural questions arise from our work. First, do our constructions generalize to two- or higher-dimensional outcomes? One promising avenue might be to combine the ideas from our log-time LMSR market maker with the work on multi-dimensional segment trees [19] to obtain efficient multi-dimensional LMSR based on a static tree. However, it is not clear how to generalize our balanced LMSR tree construction or the multi-resolution LCMM. The second question is, does our approach extends to non-interval securities, such as call options in financial markets? We leave these questions open for future research.

[8] In Fig. 1b, to facilitate comparisons, we assume that $\texttt{LMSR}_{k=4}$ equally splits the price of a coarse interval into finer intervals.

References

1. Abernethy, J., Chen, Y., Vaughan, J.W.: An optimization-based framework for automated market-making. In: Proceedings of the 12th ACM Conference on Electronic Commerce (2011)
2. Abernethy, J., Kutty, S., Lahaie, S., Sami, R.: Information aggregation in exponential family markets. In: Proceedings of the 15th ACM Conference on Economics and Computation. pp. 395–412 (2014)
3. Adel′son-Vel′skiĭ, G.M., Landis, E.M.: An algorithm for the organization of information. Soviet Mathematics—Doklady **3**, 1259–1263 (1962)
4. Chakraborty, M., Das, S., Lavoie, A., Magdon-Ismail, M., Naamad, Y.: Instructor rating markets. In: Proceedings of the 27th AAAI Conference on Artificial Intelligence. pp. 159–165 (2013)
5. Chen, Y., Fortnow, L., Lambert, N., Pennock, D.M., Vaughan, J.W.: Complexity of combinatorial market makers. In: Proceedings of the 9th ACM Conference on Electronic Commerce (2008)
6. Chen, Y., Fortnow, L., Nikolova, E., Pennock, D.M.: Betting on permutations. In: Proceedings of the 8th ACM Conference on Electronic Commerce. pp. 326–335 (2007)
7. Chen, Y., Goel, S., Pennock, D.M.: Pricing combinatorial markets for tournaments. In: Proceedings of the 40th Annual ACM Symposium on Theory of Computing. pp. 305–314 (2008)
8. Chen, Y., Pennock, D.M.: A utility framework for bounded-loss market makers. In: Proceedings of the 23rd Conference on Uncertainty in Artificial Intelligence (2007)
9. Cormen, T.H., Leiserson, C.E., Rivest, R.L.: Introduction to Algorithms. The MIT Press (1999)
10. Dudík, M., Lahaie, S., Pennock, D.M.: A tractable combinatorial market maker using constraint generation. In: Proceedings of the 13th ACM Conference on Electronic Commerce (2012)
11. Dudík, M., Lahaie, S., Rogers, R.M., Wortman Vaughan, J.: A decomposition of forecast error in prediction markets. In: Advances in Neural Information Processing Systems, pp. 4371–4380 (2017)
12. Gao, X., Chen, Y., Pennock, D.M.: Betting on the real line. In: Proceedings of the 5th Workshop on Internet and Network Economics (2009)
13. Guo, M., Pennock, D.M.: Combinatorial prediction markets for event hierarchies. In: Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems. pp. 201–208 (2009)
14. Hanson, R.D.: Decision markets. IEEE Intelligent Systems **14**(3), 16–19 (1999)
15. Hanson, R.D.: Combinatorial information market design. Information Systems Frontiers **5**(1), 107–119 (2003)
16. Hanson, R.D.: Logarithmic market scoring rules for modular combinatorial information aggregation. Journal of Prediction Markets **1**(1), 1–15 (2007)
17. Knuth, D.E.: The Art of Computer Programming, Volume 3: Sorting and Searching. Addison Wesley (1998)
18. Laskey, K.B., Sun, W., Hanson, R.D., Twardy, C., Matsumoto, S., Goldfedder, B.: Graphical model market maker for combinatorial prediction markets. Journal of Artificial Intelligence Research **63**, 421–460 (2018)
19. Mishra, P.: On updating and querying sub-arrays of multidimensional arrays. CoRR **abs/1311.6093** (2016)
20. Othman, A., Pennock, D.M., Reeves, D.M., Sandholm, T.: A practical liquidity-sensitive automated market maker. ACM Transactions on Economics and Computation **1**(3), 14:1–14:25 (2013)
21. Othman, A., Sandholm, T.: Automated market-making in the large: The gates hillman prediction market. In: Proceedings of the 11th ACM Conference on Electronic Commerce. pp. 367–376 (2010)
22. Othman, A., Sandholm, T.: Automated market makers that enable new settings: Extending constant-utility cost functions. In: Auctions, Market Mechanisms, and Their Applications. pp. 19–30 (2012)
23. Plott, C.R., Chen, K.Y.: Information aggregation mechanisms: Concept, design and implementation for a sales forecasting problem (2002), working paper No. 1131, California Institute of Technology
24. Xia, L., Pennock, D.M.: An efficient monte-carlo algorithm for pricing combinatorial prediction markets for tournaments. In: Proceedings of the 22nd International Joint Conference on Artificial Intelligence. pp. 452–457 (2011)

A Deferred Proofs

A.1 Proof of Theorem 1

The binary-search property implies that the nodes z included in the price calculation (lines 5 and 9) form the cover of I, so the algorithm correctly returns the price of I. The running time follows thanks to height balance, which implies the depth of the tree is $\mathcal{O}(\log n_{vals})$.

A.2 Proof of Theorem 2

We start by showing that a rotation at node z preserves its *partial normalization correctness*. There are two kinds of rotations, depicted in Fig. 2. The *left rotation* takes as input a node z, with children denoted z_1 and z_{23}, and children of z_{23} denoted z_2 and z_3, and rearranges these relationships by removing the node z_{23} and creating a node z_{12}, such that z now has children z_{12} and z_3, and z_{12} has children z_1 and z_2. The *right rotation* is the symmetric operation.

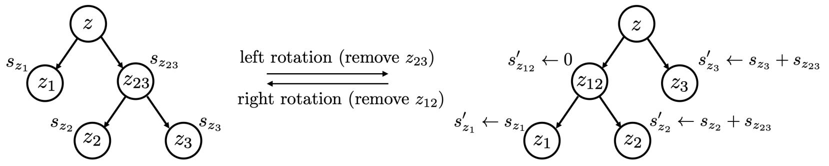

z
s_{z_1}
$s_{z_{23}}$
z_1
z_{23}
s_{z_2}
s_{z_3}
z_2
z_3
left rotation (remove z_{23})
right rotation (remove z_{12})
$s'_{z_{12}} \leftarrow 0$
$s'_{z_3} \leftarrow s_{z_3} + s_{z_{23}}$
z_{12}
$s'_{z_1} \leftarrow s_{z_1}$
$s'_{z_2} \leftarrow s_{z_2} + s_{z_{23}}$

Fig. 2: Left and right rotations with node z as an input. Depicted update corresponds to the left rotation.

The full procedure of `RotateLeft` is described in Algorithm 5. When performing rotations, we need to ensure that the node removal (i.e., removal of z_{23} in left rotation and of z_{12} in right rotation) does not impact the market state. We achieve this by moving the shares from the removed node into its children, so at the time of removal it holds zero shares (see line 4 of Algorithm 5).

Algorithm 5 Buy s shares of bundle security for an interval $I = [\alpha, 1)$.

```
1: Define subroutines:
       ResetInnerNode($z$): reset $h_z$ and $S_z$ based on the children of $z$ and the value $s_z$:
           $h_z \leftarrow 1 + \max\{h_{left(z)}, h_{right(z)}\}$, $S_z \leftarrow e^{s_z/b}(S_{left(z)} + S_{right(z)})$
       AddShares($z, s$): increase the number of shares held in $z$ by $s$:
           $s_z \leftarrow s_z + s$, $S_z \leftarrow e^{s/b} S_z$
2: procedure RotateLeft($z$):
3:     Let $z_1 = left(z)$, $z_{23} = right(z)$, $z_2 = left(z_{23})$, $z_3 = right(z_{23})$
4:     AddShares($z_2, s_{z_{23}}$), AddShares($z_3, s_{z_{23}}$), delete node $z_{23}$
5:     Let $z_{12}$ be a new node with:
           $left(z_{12}) = z_1$, $right(z_{12}) = z_2$, $I_{z_{12}} = I_{z_1} \cup I_{z_2}$, $s_{z_{12}} = 0$
6:     ResetInnerNode($z_{12}$)
7:     Update node $z$:
           $left(z) \leftarrow z_{12}$, $right(z) \leftarrow z_3$, ResetInnerNode($z$)
```

Lemma 1. *A rotation operation preserves partial-normalization correctness.*

Proof. We prove that the original partial normalization value of node z, S_z, is the same as the updated value, S'_z, after a left rotation. A right rotation follows symetrically.

$$\begin{aligned}
S_z &= e^{s_z/b} \cdot (S_{z_1} + S_{z_{23}}) \\
&= e^{s_z/b} \cdot \left(S_{z_1} + e^{s_{z}{23}/b} \cdot (S_{z_2} + S_{z_3})\right) \\
&= e^{s_z/b} \cdot \left(S'_{z_1} + S'_{z_2} + S'_{z_3}\right) \\
&= e^{s_z/b} \cdot \left(e^{s'_{z}{12}/b} \cdot \left(S'_{z_1} + S'_{z_2}\right) + S'_{z_3}\right) && \text{(since } s'_{z_{12}} = 0\text{)} \\
&= e^{s_z/b} \cdot \left(S'_{z_{12}} + S'_{z_3}\right) = S'_z
\end{aligned}$$

Proof (Proof of Theorem 2). The correctness of the **buy** operation follows because the shares are added to the nodes that form the cover of I (lines 5 and 12 in Algorithms 2), and the updates up the search path restore the properties of the LMSR tree (lines 13–17 in Algorithms 2). The running time follows from height balance, which implies that the length of the search path is $\mathcal{O}(\log n) = \mathcal{O}(\log n_{vals})$.

A.3 Proof of Theorem 3

We first show that the constraints $\mathbf{A}^\top \boldsymbol{\mu} = \mathbf{0}$ imply that all levels $\ell = 0, 1, \ldots, K$ in $\boldsymbol{\mu}$ are mutually coherent. To do this, it suffices to show that all pairs of consecutive levels ℓ and $\ell + 1$ are coherent, i.e., $\mu_y = \mu_{y_l} + \mu_{y_r}$ for all $y \in \mathcal{Z}_\ell$ where we let $y_l = \mathit{left}(y)$ and $y_r = \mathit{right}(y)$.

We proceed by induction, beginning with $\ell = K - 1$. In this base case, the constraint $\mathbf{a}_y^\top \boldsymbol{\mu} = 0$, expressed in Eq. (13), states that $b_K \mu_y = b_K \mu_{y_l} + b_K \mu_{y_r}$, implying levels $K - 1$ and K are coherent.

Now assume that all the levels $k > \ell$ are mutually coherent. We aim to show that levels l and $l + 1$ are coherent. Pick any $y \in \mathcal{Z}_\ell$. Then the constraint $\mathbf{a}_y^\top \boldsymbol{\mu} = 0$, expressed in Eq. (13), implies that

$$\begin{aligned}\left(\sum_{k>\ell} b_k\right) \mu_y &= \sum_{k>\ell} b_k \sum_{z \in \mathcal{Z}_k :\, z \subset y} \mu_z \\ &= \sum_{k>\ell} b_k \left(\sum_{z \in \mathcal{Z}_k :\, z \subseteq y_l} \mu_z + \sum_{z \in \mathcal{Z}_k :\, z \subseteq y_r} \mu_z \right) \\ &= \sum_{k>\ell} b_k \left(\mu_{y_l} + \mu_{y_r} \right). \qquad (21)\end{aligned}$$

Eq. (21) follows because y_l and y_r are in level $\ell + 1$, which is coherent with all levels $k \geq \ell + 1$ by the inductive assumption. Thus, we obtain that $\mu_y = \mu_{y_l} + \mu_{y_r}$ for all $y \in \mathcal{Z}_\ell$, establishing the coherence between levels ℓ and $\ell + 1$ and completing the induction.

To finish the proof, we note that the LCMM prices at level K are determined by C_K, so they describe a probability distribution over Ω. Since $\mathbf{A}^\top \boldsymbol{p}(\boldsymbol{\theta}) = \mathbf{0}$, all the levels in $\boldsymbol{p}(\boldsymbol{\theta})$ are coherent with level K, which means that they correspond to the expectation of $\boldsymbol{\phi}$ under the probability distribution described by the prices at level K. Thus, $\boldsymbol{p}(\boldsymbol{\theta})$ is a coherent price vector and the multi-resolution LCMM is therefore arbitrage-free.

A.4 Proof of Theorem 4

The worst-case loss of an LCMM is bounded by the sum of the worst-case losses of the component markets C_k [10]. In our case, these are LMSR submarkets with losses bounded by $b_k \log |\mathcal{Z}_k|$, so the worst-case loss of the resulting LCMM is at most

$$\sum_{k=1}^{K} b_k \log(2^k) = \sum_{k=1}^{K} b_k (k \log 2) \leq B^* \log 2,$$

proving the theorem.

A.5 Proof of Theorem 5

Algorithm 3 returns the correct price of I, because prices are coherent among submarkets and the nodes included in price calculations form a cover of I.

A.6 Proof of Theorem 6 and Additional Deferred Material from Section 4.3

We begin by deriving an identity that will be useful in the following analysis. For this derivation, let C be an LMSR with the liquidity parameter b, defined over an outcome space Ω. We will derive a relationship between the price vector in a state $\boldsymbol{\theta}$ and the price vector in a new state $\boldsymbol{\theta}' = \boldsymbol{\theta} + \boldsymbol{\delta}$, where $\boldsymbol{\delta}$ is any bundle restricted to securities in E, i.e., $\delta_\omega = 0$ for $\omega \notin E$. Denoting $\boldsymbol{\mu} = \boldsymbol{p}(\boldsymbol{\theta})$, $\mu_E = p_E(\boldsymbol{\theta})$, and $\boldsymbol{\mu}' = \boldsymbol{p}(\boldsymbol{\theta}')$, we have

$$\begin{aligned}\mu'_\omega &= \frac{e^{\theta_\omega / b} e^{\delta_\omega / b}}{\sum_{\nu \notin E} e^{\theta_\nu / b} + \sum_{\nu \in E} e^{\theta_\nu / b} e^{\delta_\nu / b}} \\ &= \frac{\mu_\omega e^{\delta_\omega / b}}{1 - \mu_E + \sum_{\nu \in E} \mu_\nu e^{\delta_\nu / b}}, \qquad (22)\end{aligned}$$

where Eq. (22) follows by dividing the numerator as well as denominator by $\sum_{\nu \in \Omega} e^{\theta_\nu / b}$.

We next establish correctness of the arbitrage removal procedure from Algorithm 4. The following lemma provides a critical step:

Lemma 2. *Fix a level $\ell < K$. Let $\tilde{\boldsymbol{\theta}}$ be a market state in $\tilde{C}$ such that the associated prices, $\boldsymbol{\mu} = \tilde{\boldsymbol{p}}(\tilde{\boldsymbol{\theta}})$, are coherent among all levels $k > \ell$. Then, for any $t \in \mathbb{R}$ and any node y with level$(y) \leq \ell$, the prices after buying t shares of $\mathbf{a}_y$, i.e., $\boldsymbol{\mu}' = \tilde{\boldsymbol{p}}(\tilde{\boldsymbol{\theta}} + t\mathbf{a}_y)$, remain coherent among all levels $k > \ell$.*

To use Lemma 2 for arbitrage removal, we start with a market state $\tilde{\boldsymbol{\theta}}$ where all levels are coherent. When a trader buys some shares of a security ϕ_y, the level $\ell = \mathit{level}(y)$ loses coherence with other levels. By buying a certain number of shares of $\mathbf{a}_y$, it is possible to restore coherence between ℓ and $\ell + 1$, and Lemma 2 then implies that coherence with all further levels $k > \ell + 1$ is also restored. The process of restoring coherence now continues with the parent of y and the bundle $\mathbf{a}_{par(y)}$ as implemented in Algorithm 4.

Proof. Consider two arbitrary levels k and m with $\ell < k < m$. Since prices are coherent between levels k and m before buying t shares of $\mathbf{a}_y$, we have, for any $z \in \mathcal{Z}_k$,

$$\mu_z = \sum_{u \in \mathcal{Z}_m \colon u \subset z} \mu_u . \tag{23}$$

Let π_y denote the price of ϕ_y according to the securities in $\mathcal{Z}_k$ and $\mathcal{Z}_m$, that is, $\pi_y = \sum_{z \in \mathcal{Z}_k \colon z \subset y} \mu_z = \sum_{u \in \mathcal{Z}_m \colon u \subset y} \mu_u$. Note that π_y might differ from μ_y, because level ℓ is not necessarily coherent with levels k and m. Let $\tilde{\boldsymbol{\theta}}' = \tilde{\boldsymbol{\theta}} + t\mathbf{a}_y$. From the definition of matrix $\mathbf{A}$, the updated $\tilde{\theta}'_z$ and $\tilde{\theta}'_u$ for any $z \in \mathcal{Z}_k$ and $u \in \mathcal{Z}_m$ are

$$\tilde{\theta}'_z = \begin{cases} \tilde{\theta}_z - tb_k & \text{if } z \subset y, \\ \tilde{\theta}_z & \text{otherwise,} \end{cases} \qquad \tilde{\theta}'_u = \begin{cases} \tilde{\theta}_u - tb_m & \text{if } u \subset y, \\ \tilde{\theta}_u & \text{otherwise.} \end{cases}$$

We calculate the new price μ'_z of any node $z \in \mathcal{Z}_k$ and show it equals to the price derived from its descendants $u \in \mathcal{Z}_m$. First, if $z \subset y$, then by Eq. (22) and Eq. (23),

$$\mu'_z = \frac{\mu_z e^{-t}}{\pi_y e^{-t} + 1 - \pi_y} = \frac{\sum_{u \in \mathcal{Z}_m \colon u \subset z} \mu_u e^{-t}}{\pi_y e^{-t} + 1 - \pi_y} = \sum_{u \in \mathcal{Z}_m \colon u \subset z} \mu'_u .$$

If $z \not\subset y$, then we similarly have

$$\mu'_z = \frac{\mu_z}{\pi_y e^{-t} + 1 - \pi_y} = \frac{\sum_{u \in \mathcal{Z}_m \colon u \subset z} \mu_u}{\pi_y e^{-t} + 1 - \pi_y} = \sum_{u \in \mathcal{Z}_m \colon u \subset z} \mu'_u .$$

Thus, prices remain coherent among all levels $m > k > \ell$.

Building upon Lemma 2, the following lemma provides the precise trade required to restore coherence after an update.

Lemma 3. *Fix a level $\ell < K$ and a node $y \in \mathcal{Z}_\ell$ and let $y_l = \mathit{left}(y)$ and $y_r = \mathit{right}(y)$. Let $\tilde{\boldsymbol{\theta}}^0$ and $\tilde{\boldsymbol{\theta}} = \tilde{\boldsymbol{\theta}}^0 + \boldsymbol{\delta}$ be market states in $\tilde{C}$, with associated prices $\boldsymbol{\mu}^0 = \tilde{\boldsymbol{p}}(\tilde{\boldsymbol{\theta}}^0)$ and $\boldsymbol{\mu} = \tilde{\boldsymbol{p}}(\tilde{\boldsymbol{\theta}})$ such that:*

- *prices $\boldsymbol{\mu}^0$ are coherent among all levels $k \geq \ell$;*
- *$\boldsymbol{\delta}$ is a vector, which is zero outside descendants of y, i.e., $\delta_z = 0$ whenever $z \not\subseteq y$;*
- *prices $\boldsymbol{\mu}$ are coherent among all levels $k > \ell$.*

Let $\tilde{\boldsymbol{\theta}}' = \tilde{\boldsymbol{\theta}} + t\mathbf{a}_y$ where

$$t = \frac{b_\ell}{B_{\ell-1}} \log \left(\frac{1 - \mu_y}{\mu_y} \cdot \frac{\mu_{y_l} + \mu_{y_r}}{1 - \mu_{y_l} - \mu_{y_r}} \right).$$

Then the associated prices $\boldsymbol{\mu}' = \tilde{\boldsymbol{p}}(\tilde{\boldsymbol{\theta}}')$ are coherent among all levels $k \geq \ell$.

Proof. By Lemma 2, adding $t\mathbf{a}_y$ to $\tilde{\boldsymbol{\theta}}$ maintains coherence among levels $k > \ell$, so it suffices to show that levels ℓ and $\ell+1$ are mutually coherent in $\boldsymbol{\mu}'$. Thus, we have to show that $\mu'_z = \mu'_{\mathit{left}(z)} + \mu'_{\mathit{right}(z)}$ for all $z \in \mathcal{Z}_\ell$.

First note that by the assumption on $\boldsymbol{\delta}$ and the definition of $\mathbf{a}_y$, we have

$$\begin{aligned} \tilde{\theta}^0_z &= \tilde{\theta}_z = \tilde{\theta}'_z \quad \text{for all } z \in \mathcal{Z}_\ell \backslash \{y\} \\ \tilde{\theta}^0_u &= \tilde{\theta}_u = \tilde{\theta}'_u \quad \text{for all } u \in \mathcal{Z}_{\ell+1} \backslash \{y_l, y_r\}. \end{aligned}$$

Therefore, by Eq. (22), we have for all $z \in \mathcal{Z}_\ell \backslash \{y\}$

$$\frac{\mu'_z}{1-\mu'_y} = \frac{\mu^0_z}{1-\mu^0_y}, \qquad \text{and} \qquad \frac{\mu'_{\mathit{left}(z)} + \mu'_{\mathit{right}(z)}}{1-\mu'_{y_l}-\mu'_{y_r}} = \frac{\mu^0_{\mathit{left}(z)} + \mu^0_{\mathit{right}(z)}}{1-\mu^0_{y_l}-\mu^0_{y_r}}. \tag{24}$$

Since the vector $\boldsymbol{\mu}^0$ satisfies $\mu^0_z = \mu^0_{\mathit{left}(z)} + \mu^0_{\mathit{right}(z)}$ for all $z \in \mathcal{Z}_\ell \backslash \{y\}$, Eq. (24) implies that we also have $\mu'_z = \mu'_{\mathit{left}(z)} + \mu'_{\mathit{right}(z)}$ for all $z \in \mathcal{Z}_\ell \backslash \{y\}$ as long as $\mu'_y = \mu'_{y_l} + \mu'_{y_r}$. Thus, in order to show that levels ℓ and $\ell+1$ are coherent in $\boldsymbol{\mu}'$, it suffices to show that $\mu'_y = \mu'_{y_l} + \mu'_{y_r}$.

We begin by explicitly calculating $\tilde{\theta}'_z$ and $\tilde{\theta}'_u$ for any $z \in \mathcal{Z}_\ell$ and any $u \in \mathcal{Z}_{\ell+1}$:

$$\tilde{\theta}'_z = \begin{cases} \tilde{\theta}_z + tB_\ell & \text{if } z = y, \\ \tilde{\theta}_z & \text{otherwise,} \end{cases} \qquad \tilde{\theta}'_u = \begin{cases} \tilde{\theta}_u - tb_{\ell+1} & \text{if } u \in \{y_l, y_r\}, \\ \tilde{\theta}_u & \text{otherwise.} \end{cases}$$

Therefore,

$$\mu'_y = \frac{\mu_y e^{tB_\ell/b_\ell}}{\mu_y e^{tB_\ell/b_\ell} + 1 - \mu_y} = \frac{1}{1 + \frac{1-\mu_y}{\mu_y} e^{-tB_\ell/b_\ell}}$$

and similarly,

$$\mu'_{y_l} + \mu'_{y_r} = \frac{(\mu_{y_l} + \mu_{y_r})e^{-t}}{(\mu_{y_l} + \mu_{y_r})e^{-t} + 1 - \mu_{y_l} - \mu_{y_r}} = \frac{1}{1 + \frac{1-\mu_{y_l}-\mu_{y_r}}{\mu_{y_l}+\mu_{y_r}} e^{t}}.$$

Thus, it remains to show that

$$\tfrac{1-\mu_y}{\mu_y} e^{-tB_\ell/b_\ell} = \tfrac{1-\mu_{y_l}-\mu_{y_r}}{\mu_{y_l}+\mu_{y_r}} e^{t},$$

or equivalently:

$$\tfrac{1-\mu_y}{\mu_y} \cdot \tfrac{\mu_{y_l}+\mu_{y_r}}{1-\mu_{y_l}-\mu_{y_r}} = e^{t(1+B_\ell/b_\ell)}.$$

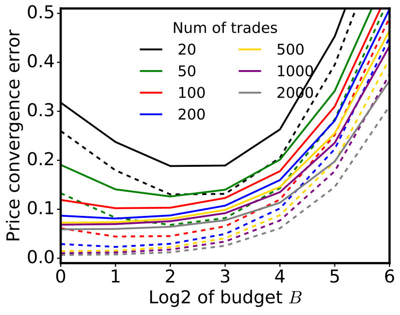
But this follows from our choice of t and the fact that $B_{\ell-1} = B_\ell + b_\ell$, completing the proof.

We finish the section with the proof of Theorem 6.

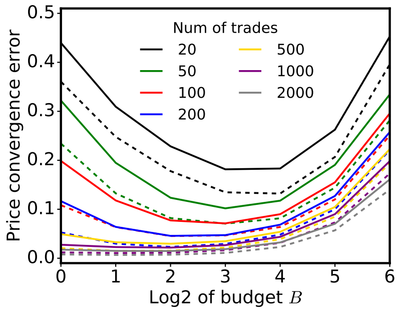
Proof (Proof of Theorem 6). Algorithm 4 correctly updates the tree (and returns the cost), because the shares are added to the nodes that form a cover of I, and coherence is then restored by applying Lemma 3 up the search path. Running times of both algorithms are proportional to the length of the search path to the first node z with $\alpha_z = \alpha$, whose level coincides with the precision of α.

B Additional Numerical Experiments

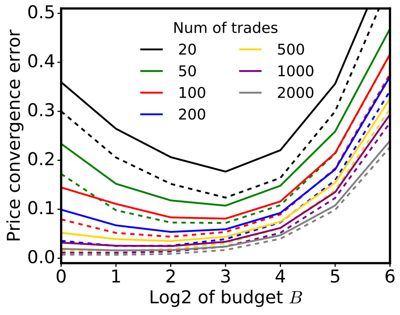
In Section 5, we demonstrated that by splitting the budget between submarkets that offer interval securities at different precisions, the multi-resolution LCMM is able to interpolate the performance of LMSR market makers. It can aggregate information at the coarser level efficiently, while achieving accurate belief elicitation at the finer resolution, given enough trading period. Here we provide numerical results over a wider range of budget constraints, validating how the multi-resolution LCMM may balance the price convergence behavior of LMSR markets.



(a) $\texttt{LMSR}_{k=4}$.



(b) $\texttt{LMSR}_{k=8}$.



(c) $\texttt{LCMM}_{50/50}$.

Fig. 3: The price convergence error as a function of liquidity and the number of trades (indicated by the color of the line) for the three respective market makers. Solid lines record price convergence error at the finer precision level $k = 8$, and dashed ones at the coarser level $k = 4$.

Fig. 3 shows the price convergence error as a function of budget constraint (thus, the liquidity parameter) and the number of trades for the three respective market makers. Results are averaged over forty random but controlled trading sequences. The solid lines depict the price convergence error at precision level $k = 8$, and the dashed ones for precision level $k = 4$. The minimum point on each curve indicates the optimal budget, or the optimal value of the liquidity parameter to adopt, for the particular cost function and a specific number of trades.

Intuitively, when the budget for running a market is sufficient, a market operator can support interval securities at any fine-grained precision level, or use only a portion of the budget to achieve optimal performance. However, when the budget for running a market is limited, say B less than 8, the market designer can preferably aggregate information faster at a coarser resolution by limiting the precision of interval endpoints (e.g., adopting $\texttt{LMSR}_{k=4}$). However, by design, it can not accurately elicit beliefs at finer resolutions, even when the market is run for a sufficiently long period of time. The $\texttt{LMSR}_{k=8}$, on the other hand, benefits from a larger number of trades to aggregate more fine-grained information. Running the two LMSR markets independently may balance this convergence trade-off, but inevitably results in inconsistent prices between the markets. Given the different convergence properties of separate LMSRs, a multi-resolution LCMM can allocate its budget accordingly to achieve a desired convergence performance, while maintaining coherent prices. For example, a market designer, who considers information at precision levels $k = 4$ and $k = 8$ equally important, may divide the budget between the two levels to enjoy faster price convergence at the coarser resolution, while accurately aggregating a full probability distribution of the continuous variable as trading proceeds.