# Log-time Prediction Markets for Interval Securities

Miroslav Dudík*
Microsoft Research, New York, NY
mdudik@microsoft.com

Xintong Wang*
University of Michigan, Ann Arbor, MI
xintongw@umich.edu

David M. Pennock
Rutgers University, New Brunswick, NJ
dpennock@dimacs.rutgers.edu

David M. Rothschild
Microsoft Research, New York, NY
davidmr@microsoft.com

## ABSTRACT

We design a prediction market to recover a complete and fully general probability distribution over a random variable. Traders buy and sell *interval securities* that pay $1 if the outcome falls into an interval and $0 otherwise. Our market takes the form of a central *automated market maker* and allows traders to express interval endpoints of arbitrary precision. We present two designs in both of which market operations take time logarithmic in the number of intervals (that traders distinguish), providing the first computationally efficient market for a continuous variable. Our first design replicates the popular *logarithmic market scoring rule* (LMSR), but operates exponentially faster than a standard LMSR by exploiting its modularity properties to construct a balanced binary tree and decompose computations along the tree nodes. The second design consists of two or more parallel LMSR market makers that mediate submarkets of increasingly fine-grained outcome partitions. This design remains computationally efficient for all operations, including arbitrage removal across submarkets. It adds two additional benefits for the market designer: (1) the ability to express utility for information at various resolutions by assigning different liquidity values, and (2) the ability to guarantee a true constant bounded loss by appropriately decreasing the liquidity in each submarket.

## KEYWORDS

prediction market; automated market maker; expressive betting

**ACM Reference Format:**
Miroslav Dudík*, Xintong Wang*, David M. Pennock, and David M. Rothschild. 2021. Log-time Prediction Markets for Interval Securities. In *Proc. of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2021), Online, May 3–7, 2021*, IFAAMAS, 9 pages.

## 1 INTRODUCTION

Consider a one-dimensional 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 the outcome will fall into some specified interval, say between 2957.60 and 3804.59, implemented as binary contracts that pay out $1 if the outcome falls in the interval and $0 otherwise. We are interested in designing *automated market makers* to facilitate a fully *expressive* market computationally efficiently. Traders can select custom interval endpoints of arbitrary precision corresponding to a continuous outcome space, whereas the market maker will always offer to buy or sell any interval security at some price.

A form of interval security called the *condor spread* is common in *financial options* markets, with significant volume of trade. Each condor spread involves trading four different options,[1] and financial options offered by the market may only support a limited subset of approximate intervals. As of this writing, S&P 500 options expiring on December 17, 2021, distinguish 56 strike prices, allowing the purchase of around 1500 distinct intervals of minimum width 25. Moreover, as each strike price trades independently despite the logical constraints on their relative values, it will require time linear in the number of offered strike prices to remove arbitrage.

Outside traditional financial markets, the *logarithmic market scoring rule* (LMSR) market maker [15, 16] has been used to elicit information through the trade of interval securities. The 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, standard implementations of LMSR operations run in time linear in the number of outcomes or distinct future values traders define—in our case, arbitrarily many. The constant-log-utility and other barrier-function-based market makers [8, 22] achieve constant bounded loss, but still suffer the second limitation regarding computational intractability. Thus, previous markets allow only a relatively small set of predetermined intervals 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 propose two automated market makers that perform exponentially faster than the standard LMSR and previous designs. Market operations (i.e., **price**, **cost**, and **buy**) can be executed in time *logarithmic* in the number of distinct intervals traded,

*Authors contribute equally.

*Proc. of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2021), U. Endriss, A. Nowé, F. Dignum, A. Lomuscio (eds.), May 3–7, 2021, Online.*


[1] A *call option* written on an underlying stock with *strike price* $K$ and expiration date $T$ pays $\max\{S-K, 0\}$, where $S$ is the opening price of the stock on date $T$. For example, 25 shares of "$1 iff [2650,2775]" $\approx \max\{S-2650, 0\} - \max\{S-2675, 0\} - \max\{S-2750, 0\} + \max\{S-2775, 0\}$.

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

or linear in the number of bits describing the outcome space. Our first market maker calculates LMSR exactly, but employs a balanced binary tree to implement interval queries and trades. We show that the normalization constant of LMSR—a key quantity in its price and cost function—can be calculated recursively via local computations on the balanced tree. Our work here contributes to the rich literature that aims to overcome the worst-case #P-hardness of LMSR pricing [5] by exploiting the outcome space structure and limiting expressivity [6, 7, 13, 18, 24].

Our second market maker works by maintaining parallel LMSR submarkets that adopt different liquidity parameters and offer interval securities at various resolutions. We show that liquidity parameters can be chosen to guarantee a *constant* bounded loss independent of market precision and prices can be kept coherent efficiently by removing arbitrages across submarkets. We demonstrate through agent-based simulation that our second design enjoys more flexible liquidity choices to facilitate the information-gathering objective: it can get close to the "best of both worlds" displayed by coarse and fine LMSR markets, with prices converging fast at both resolutions regardless of the traders' information structure.

The two proposed designs, to our knowledge, are the first to simultaneously achieve expressiveness and computational efficiency. As both market makers facilitate trading intervals at arbitrary precision, they can elicit any probability distribution over a continuous random variable that can be practically encoded by a machine. We use the S&P 500 index value as a running example, but our framework is generic and can handle any one-dimensional continuous variable, for example, the landfall point of a hurricane along a coastline or the number of tickets sold in the first week of a 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 $\Omega$ 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 $\phi_i : \Omega \to \{0, 1\}$, 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 $\phi_i$ is associated with a *security* that pays out $\phi_i(\omega)$ when the outcome $\omega \in \Omega$ occurs, and thus $\phi_i$ 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 entries to short sales. A trader holding a bundle $\boldsymbol{\delta}$ receives a payoff of $\boldsymbol{\delta} \cdot \boldsymbol{\phi}(\omega)$, when $\omega$ occurs.

Following [1] and [8], we assume that the market maker determines security prices using a convex and differentiable potential function $C : \mathbb{R}^{|\mathcal{I}|} \to \mathbb{R}$, 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 who wants 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 $\boldsymbol{\theta} + \boldsymbol{\delta}$.

The vector of instantaneous prices in the corresponding state $\boldsymbol{\theta}$ is $\boldsymbol{p}(\boldsymbol{\theta}) := \nabla C(\boldsymbol{\theta})$. 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 $\omega$ are possible, there be no market transaction 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, $\sup_{\boldsymbol{\theta}\in\mathbb{R}^{|\mathcal{I}|}} \sup_{\omega\in\Omega} \left[\boldsymbol{\theta} \cdot \boldsymbol{\phi}(\omega) - C(\boldsymbol{\theta}) + C(\mathbf{0})\right]$, meaning the largest 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 property requires that this worst-case loss be a priori bounded by a constant.

### 2.2 Complete Markets and LMSR

In a complete market, we have $\mathcal{I} = \Omega$. Securities are indicators of individual outcomes, $\phi_i(\omega) = 1\{\omega = i\}$, where $1\{\cdot\}$ denotes the binary indicator. We denote each market security as $\phi_\omega$. A risk-neutral trader is incentivized to move the price of each security $\phi_\omega$ to her estimate of $\mathbb{E}[\phi_\omega] = \mathbb{P}[\omega]$, which is her subjective probability of $\omega$ occurring. Thus, traders can express arbitrary probability distributions over $\Omega$. We consider variants of LMSR market maker [15] for a complete market, described by cost function and prices

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

where $b$ is the liquidity parameter, controlling how fast the price moves in response to trading and limiting the worst-case loss of the market maker to $b \log|\Omega|$ [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*. The immediate price of the bundle $\mathbf{1}_E$ in the state $\boldsymbol{\theta}$ is

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

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

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

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 from Eq. (2).

### 2.3 Interval Securities over $[0, 1)$

We consider betting on outcomes within an interval $[0, 1)$. Our approach generalizes to outcomes that are in any $[\alpha, \beta) \subseteq [-\infty, \infty)$ by applying any increasing transformation $F : [\alpha, \beta) \to [0, 1)$. We assume that the outcome $\omega$ is specified with $K$ bits, meaning that there are $N = 2^K$ outcomes with $\Omega = \{j/N : j \in \{0, 1, \ldots, N-1\}\}$. At the end of Sections 3 and 4, we discuss how the assumption of pre-specified bit precision can be removed.

Example 1 (Complete market for S&P 500). *We construct a complete market for the S&P 500 opening price on December 17, 2021, by setting $N = 2^{19} = 524{,}288$. The resulting complete market is $\mathcal{I} = \{0,\ 0.01,\ \ldots,\ 5242.86,\ 5242.87\}$, where we cap prices at \$5242.87 (i.e., larger prices are treated as \$5242.87). The transformed outcome is then $\omega = \omega'/N$, where $\omega'$ is the S&P 500 price in cents.*

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

## 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 distinct 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). The root is denoted $root$ and the parent of any non-root node $par(z)$.

Definition 1 (LMSR Tree). *An* LMSR tree *is a full binary tree, where each node $z$ is annotated with an interval $I_z = [\alpha_z, \beta_z)$ with $\alpha_z, \beta_z \in \Omega \cup \{1\}$, a height $h_z \geq 0$, a quantity $s_z \in \mathbb{R}$ that records the number of sold bundle securities associated with $I_z$, and a partial normalization constant $S_z \geq 0$ (defined below in Eq. 6).*

*An LMSR tree is required to satisfy:*

- Binary-search property*: $I_{root} = [0, 1)$, and for inner node $z$,*
$$\alpha_z = \alpha_{left(z)} \ < \ \beta_{left(z)} = \alpha_{right(z)} \ < \ \beta_{right(z)} = \beta_z.$$
- Height balance*: $h_z = 0$ for leaves, and for inner node $z$,*
$$h_z = 1 + \max\{h_{left(z)}, h_{right(z)}\}, \quad |h_{left(z)} - h_{right(z)}| \leq 1.$$
- Partial-normalization correctness*: $S_z = e^{s_z/b} \cdot (\beta_z - \alpha_z)$ for leaves, and for inner node $z$,*
$$S_z = e^{s_z/b} \cdot \left(S_{left(z)} + S_{right(z)}\right).$$

The *binary-search property* helps to find the unique leaf that contains any $\omega \in \Omega$ by descending from *root* and choosing left or right in each node based on whether $\omega < \beta_{left(z)}$ or $\omega \geq \beta_{left(z)}$. The *height-balance property* ensures that the path length from root to any leaf is at most $O(\log n)$, where $n$ is the number of leaves of the tree [17]. We adopt an *AVL tree* [3] at the basis of our LMSR tree, but other balanced binary-search trees (e.g., red-black trees or splay trees) could also be used.

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 and its components for each individual outcome $\omega$ represented by the LMSR tree $T$ are[3]

$$\boldsymbol{\theta}(T) = \sum_{z \in T} s_z \mathbf{1}_{I_z}; \quad \theta_\omega(T) = \sum_{z \in T} s_z \mathbf{1}_{I_z, \omega} = \sum_{z \ni \omega} s_z. \tag{4}$$

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

$$\sum_{\omega \in \Omega} e^{\theta_\omega / b} = \sum_{\omega \in \Omega} e^{\sum_{z \ni \omega} s_z / b} = \sum_{\omega \in \Omega} \prod_{z \ni \omega} e^{s_z/b}. \tag{5}$$

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

$$S_z \coloneqq \frac{1}{N} \sum_{\omega \in z} \prod_{z' :\, z \supseteq z' \ni \omega} e^{s_{z'}/b}. \tag{6}$$

Thus, we have $\sum_{\omega \in \Omega} e^{\theta_\omega / b} = N S_{root}$ and obtain the following recursive relationship, which we refer to as *partial-normalization correctness* and is at the core of implementing **price** and **buy**:

$$S_z = \begin{cases} e^{s_z/b} \cdot (\beta_z - \alpha_z) & \text{if } z \text{ is a leaf,} \\ e^{s_z/b} \cdot \left(S_{left(z)} + S_{right(z)}\right) & \text{otherwise.} \end{cases} \tag{7}$$

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

- **price**$(I, T)$ returns the price of bundle security for $I$;
- **cost**$(I, s, T)$ returns the cost of $s$ shares of bundle security for $I$;
- **buy**$(I, s, T)$ updates $T$ to reflect the purchase of $s$ shares of bundle security for $I$.

For **cost**, it suffices to implement **price** and use Eq. (3). Since the price of $[\alpha, \beta)$ satisfies $p_{[\alpha,\beta)}(\boldsymbol{\theta}) = p_{[\alpha,1)}(\boldsymbol{\theta}) - p_{[\beta,1)}(\boldsymbol{\theta})$, it suffices to implement **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 the market ends up in the same state $\boldsymbol{\theta} + s\mathbf{1}_{[\alpha,\beta)}$. We implement **price** and **buy** for one-sided intervals $I = [\alpha, 1)$, and the remaining operations will follow.

### 3.2 Price Queries

We consider price queries for $I = [\alpha, 1)$. Let $vals(T) = \{\alpha_z : z \in T\}$ denote the set of distinct left endpoints in the tree nodes. We start by assuming that $\alpha \in vals(T)$, and later relax this assumption. We proceed to calculate $p_I(\boldsymbol{\theta})$ in two steps. *First*, we construct a set of nodes $\mathcal{Z}$ whose associated intervals $I_z$ are disjoint and cover $I$. To achieve this, we conduct a binary search for $\alpha$, putting in $\mathcal{Z}$ all of the right children of the visited nodes that have $\alpha_z > \alpha$, as well

[3] We write $\omega \in z$ to mean $\omega \in I_z$ and $z' \subseteq z$ to mean $I_{z'} \subseteq I_z$. Thus, $z' \subseteq z$ means that $z'$ is a descendant of $z$ in $T$, and $z' \subset z$ means that $z'$ is a strict descendant of $z$.

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

**Input:** Interval $I = [\alpha, 1)$, $\alpha \in \Omega$, LMSR tree $T$.
**Output:** Price of bundle security for $I$.

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

as the final node with $\alpha_z = \alpha$. Thanks to the height balance, the cardinality of $\mathcal{Z}$ is $O(\log n)$, where $n$ is the number of leaves of $T$. 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 node $z \in \mathcal{Z}$. Starting from the LMSR price in Eq. (2), we take advantage of the defined partial normalization constants $S_z$ to calculate $p_{I_z}(\boldsymbol{\theta})$:

$$p_{I_z}(\boldsymbol{\theta}) = \frac{1}{NS_{root}} \sum_{\omega \in z} e^{\theta_\omega/b} = \frac{1}{S_{root}} \cdot \frac{1}{N} \sum_{\omega \in z} \prod_{z' \ni \omega} e^{s_{z'}/b} \tag{8}$$

$$= \frac{1}{S_{root}} \cdot \frac{1}{N} \sum_{\omega \in z} \left[ \left( \prod_{z' :\, z \supseteq z' \ni \omega} e^{s_{z'}/b} \right) \left( \prod_{z' \supset z} e^{s_{z'}/b} \right) \right] \tag{9}$$

$$= \frac{S_z}{S_{root}} \underbrace{\left( \prod_{z' \supset z} e^{s_{z'}/b} \right)}_{P_z}. \tag{10}$$

In Eq. (8), we use that $NS_{root} = \sum_{\omega \in \Omega} e^{\theta_\omega/b}$ and then expand $\theta_\omega$ using Eq. (4). In Eq. (9), 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. (10) iterates over $z'$ on the path from root to $z$, and thus can be calculated along the binary-search path.

We now handle the case when $\alpha \notin vals(T)$. After the leaf $z$ on the search path is reached, we have $\alpha_z < \alpha < \beta_z$. Instead of expanding the tree, we conceptually create two children of $z$: $z'$ and $z''$ with $I_{z'} = [\alpha_z, \alpha)$ and $I_{z''} = [\alpha, \beta_z)$, and add $z''$ in $\mathcal{Z}$. Since $\theta_\omega$ 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 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 $O(\log n_{vals})$, where $n_{vals}$ denotes the number of distinct values appeared as endpoints of intervals in all the executed transactions. We defer complete proofs from this paper to the appendix, which is available in the full version of this paper on arXiv.

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

## 3.3 Buy Transactions

We next implement **buy**$([\alpha, 1), s, T)$ while maintaining the LMSR tree properties. The main challenge here is to simultaneously maintain *partial-normalization correctness* and *height balance*. We address this by adapting AVL-tree rebalancing.

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

**Input:** Quantity $s \in \mathbb{R}$, interval $I = [\alpha, 1)$, $\alpha \in \Omega$, LMSR tree $T$.
**Output:** Tree $T$ updated to reflect the purchase of $s$ shares of $\mathbf{1}_I$.

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 $z$
    $h_z \leftarrow 1 + \max\{h_{left(z)}, h_{right(z)}\}$, $S_z \leftarrow e^{s_z/b}(S_{left(z)} + S_{right(z)})$
  AddShares($z$, $s$): increase the number of shares held in $z$ by $s$
    $s_z \leftarrow s_z + s$, $S_z \leftarrow e^{s/b} S_z$
Initialize $z \leftarrow root$
**while** $\alpha_z \neq \alpha$ **and** $z$ is not a leaf **do** ▷ add $s$ shares to $z \in \mathcal{Z}$
  **if** $\alpha < \alpha_{right(z)}$ **then**
    AddShares($right(z)$, $s$)
    $z \leftarrow left(z)$
  **else**
    $z \leftarrow right(z)$
**if** $\alpha_z < \alpha$ **then** ▷ split the leaf $z$
  $left(z) \leftarrow$ NewLeaf($\alpha_z$, $\alpha$), $right(z) \leftarrow$ NewLeaf($\alpha$, $\beta_z$)
  $z \leftarrow right(z)$
AddShares($z$, $s$)
**while** $z$ is not a *root* **do** ▷ trace the binary-search path back
  $z \leftarrow par(z)$
  **if** $|h_{left(z)} - h_{right(z)}| \geq 2$ **then** ▷ restore height balance
    Rotate $z$ and possibly one of its children
    (details in Appendix A.2, Algorithm 5)
  ResetInnerNode($z$) ▷ update $h_z$ and $S_z$

We begin by considering the case $\alpha \in vals(T)$. Similar to price queries, we conduct binary search for $\alpha$ to obtain the set of nodes $\mathcal{Z}$ that covers $I = [\alpha, 1)$. We update the values of $s_z$ across $z \in \mathcal{Z}$ by adding $s$, and obtain $T'$ that has the same structure as $T$ 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

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

We then rely on the recursive relationship defined in Eq. (7) to update the partial normalization constants $S_z$. It suffices to update the ancestors of the nodes $z \in \mathcal{Z}$, all of which lie along the search path to $\alpha$, and each update requires constant time.

When $\alpha \notin vals(T)$, we split the leaf $z$ that contains $\alpha \in [\alpha_z, \beta_z)$ before adding shares to $right(z)$. This may violate the *height-balance property*. 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 show in Appendix A.2, Lemma 1, that in each rotation, only a constant number of nodes needs to be updated to preserve the *partial-normalization correctness*. Thus, the overall running time of the **buy** operation, presented in Algorithm 2, is $O(\log n_{vals})$.

Theorem 2. *Algorithm 2 implements* **buy** *in time* $O(\log n_{vals})$.

*Remarks.* We show that **price**, **cost** and **buy** can be implemented in time $O(\log n_{vals})$, which is bounded above by the log of the number of **buy** transactions $O(\log n_{buy})$ and the bit precision of the outcome $O(\log N) = O(K)$.[4] 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 affects the worst-case loss bound for the market maker, which is $O(\log N) = O(K)$. Next section presents a different construction that achieves a *constant* worst-case loss independent of the market precision.

# 4 A MULTI-RESOLUTION LINEARLY CONSTRAINED MARKET MAKER

We introduce our second design, referred to as the *multi-resolution linearly constrained market maker* (multi-resolution LCMM). The design is based on the 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. 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 maintain coherent prices, we design a matrix that imposes linear constraints to tie market prices among different submarkets to support the efficient removal of any arbitrage opportunity, following Dudík et al. [10]. We first define the multi-resolution LCMM and its properties, and show that **price**, **cost** and **buy** can be implemented in time $O(\log N)$.

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

*4.1.1 A Multi-resolution Market.* A binary search tree remains at the core of our multi-resolution market construction. Unlike a log-time LMSR that uses a self-balancing 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 $[0, 1)$. We start with an example of a market that offers interval securities at two resolutions.

Example 2 (Two-level market for $[0, 1)$). *We consider a market composed of two submarkets, indexed by $\mathcal{I}_1 = \{11, 12\}$ and $\mathcal{I}_2 = \{21, 22, 23, 24\}$, which partition $[0, 1)$ into interval events at two levels of coarseness:*

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

*The market provides six interval securities $\phi_{11}, \ldots, \phi_{24}$ associated with the corresponding interval events, i.e., $\mathcal{I} = \mathcal{I}_1 \uplus \mathcal{I}_2$ and $|\mathcal{I}| = 6$.*

We extend Example 2 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 $\omega$. 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 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 in half. Thus, level $k$ partitions $[0, 1)$ into $2^k$ intervals of size $2^{-k}$ and the final level $\mathcal{Z}_K$ contains $N = 2^K$ leaves.

We index interval securities by nodes, with their payoffs defined by $\phi_z(\omega) = 1\{\omega \in I_z\}$. We partition 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 submarket, we define the LMSR cost function $C_k$ with a *separate* liquidity parameter $b_k > 0$:

$$C_k(\boldsymbol{\theta}_k) = b_k \log\left(\sum_{z \in \mathcal{Z}_k} e^{\theta_z / b_k}\right). \tag{11}$$

*4.1.2 A Linearly Constrained Market Maker.* Following the above multi-resolution construction, the overall market has a *direct-sum cost* $\tilde{C}(\boldsymbol{\theta}) = \sum_{k \le K} C_k(\boldsymbol{\theta}_k)$, which corresponds to pricing securities in each block $\mathcal{I}_k$ independently using $C_k$. However, as there are logical dependencies between securities in different levels, independent pricing may lead to incoherent prices among submarkets and create arbitrage opportunities.

Example 3 (Arbitrage in a two-level market). *Continuing Example 2, we define separate LMSR costs, where $b_1 = 1$ and $b_2 = 1$:*

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

*The direct-sum market $\tilde{C}(\boldsymbol{\theta}) = C_1(\boldsymbol{\theta}_1) + C_2(\boldsymbol{\theta}_2)$ allows incoherent prices. For example, after buying some shares of security $\phi_{21}$ associated with $I_{21} = \left[0, \frac{1}{4}\right)$ in submarket $\mathcal{I}_2$, the market can have*

$$\tilde{p}_{11}(\boldsymbol{\theta}) = 0.5; \quad \tilde{p}_{21}(\boldsymbol{\theta}) + \tilde{p}_{22}(\boldsymbol{\theta}) = 0.6.$$

*These prices are incoherent, i.e., do not correspond to probabilities of $I_{11}$, $I_{21}$, $I_{22}$, because under any probability distribution over $\Omega$, we must have $\mathbb{P}[I_{11}] = \mathbb{P}[I_{21}] + \mathbb{P}[I_{22}]$ and $\mathbb{P}[I_{12}] = \mathbb{P}[I_{23}] + \mathbb{P}[I_{24}]$. Thus, a coherent price vector $\boldsymbol{\mu} \in \mathbb{R}^{|\mathcal{I}|}$ must satisfy linear constraints $\mu_{11} - \mu_{21} - \mu_{22} = 0$ and $\mu_{12} - \mu_{23} - \mu_{24} = 0$, which can be also written as $\mathbf{a}_1^\top \boldsymbol{\mu} = 0$ and $\mathbf{a}_2^\top \boldsymbol{\mu} = 0$ where*

$$\mathbf{a}_1 = (1, 0, -1, -1, 0, 0)^\top \quad \text{and} \quad \mathbf{a}_2 = (0, 1, 0, 0, -1, -1)^\top.$$

*We refer to $\mathbf{A} = (\mathbf{a}_1, \mathbf{a}_2) \in \mathbb{R}^{|\mathcal{I}| \times 2}$ as the constraint matrix.*

We extend Example 3 to specify price constraints in a multi-resolution market. Later we will show how the constraint matrix can be used to remove arbitrage arising from the constraint violations.

Recall that $\mathcal{M}$ denotes a *coherent price space*, where any expected payoff lies in the convex hull of $\{\boldsymbol{\phi}(\omega)\}_{\omega \in \Omega}$. For the multi-resolution market, we specify a set of *homogeneous linear equalities* describing a superset of $\mathcal{M}$.

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

We design the constraint matrix $\mathbf{A}$ to ensure that any pair of submarkets is price coherent, 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

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

For algorithmic reasons (as we will see in Section 4.3), we further tie the price of $y$ to the prices of *all* of $y$'s descendants and weight each level by its liquidity parameter $b_k$:

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

Now we can 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

[4] Clearly, $n_{vals} \le 2n_{buy}$ 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 $\Omega \cup \{1\}$.

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 $y \in \mathcal{Y}^*$:

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

Arbitrage opportunities arise if the price of bundle $\mathbf{a}_j$ differs from zero, where $\mathbf{a}_j$ denotes the $j$th column of $\mathbf{A}$. Traders profit by buying a positive quantity of $\mathbf{a}_j$ if its price is negative, and selling otherwise. Thus, the constraint matrix $\mathbf{A}$ gives a recipe for arbitrage removal. We provide the intuition for this in the two-level market, and then give the definition of the multi-resolution LCMM.

Example 4 (Arbitrage removal in a two-level market). *Continuing Example 3, the prices $\tilde{\boldsymbol{p}}(\boldsymbol{\theta})$ violate the constraint $\mathbf{A}^\top \boldsymbol{\mu} = \mathbf{0}$, because $\mathbf{a}_1^\top \tilde{\boldsymbol{p}}(\boldsymbol{\theta}) = \tilde{p}_{11}(\boldsymbol{\theta}) - \tilde{p}_{21}(\boldsymbol{\theta}) - \tilde{p}_{22}(\boldsymbol{\theta}) = 0.5 - 0.6 \neq 0$. The vector $\mathbf{a}_1$ reveals an arbitrage opportunity: buy the security $\phi_{11}$ (at the initial price 0.5) and simultaneously sell securities $\phi_{21}$ and $\phi_{22}$ (at the initial price 0.6), i.e., buy bundle $\mathbf{a}_1$. Since under any outcome $\omega$, the payout for the bundle $\mathbf{a}_1$ is 0, this is initially profitable. However, buying $\mathbf{a}_1$ will increase the price of $\phi_{11}$ and decrease the prices of $\phi_{21}$ and $\phi_{22}$. Once a sufficiently large quantity $s$ of shares of $\mathbf{a}_1$ is bought, this form of arbitrage is removed and we have $\mathbf{a}_1^\top \tilde{\boldsymbol{p}}(\tilde{\boldsymbol{\theta}}) = 0$ in a new state $\tilde{\boldsymbol{\theta}} = \boldsymbol{\theta} + s\mathbf{a}_1 = \boldsymbol{\theta} + \mathbf{A}\boldsymbol{\eta}$, where $\boldsymbol{\eta} := (s, 0)^\top$.*

A linearly constrained market maker (LCMM) [10] leverages violated constraints similarly as in Example 4 to remove arbitrage, and then returns the arbitrage proceeds to the trader. Formally, an LCMM is described by the cost function

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

It relies on the direct-sum cost $\tilde{C}$, but with each trader purchase $\boldsymbol{\delta}$ that causes incoherent 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 $\mathbf{A}\boldsymbol{\eta}$.*

We note that the purchase of bundle $\mathbf{A}\boldsymbol{\delta}_{\text{arb}}$ has no effect on the trader's payoff, since $(\mathbf{A}\boldsymbol{\delta}_{\text{arb}})^\top \boldsymbol{\phi}(\omega) = 0$ 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 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 $\mathbf{A}^\top \left(\nabla \tilde{C}(\boldsymbol{\theta} + \mathbf{A}\boldsymbol{\eta}^\star)\right) = \mathbf{0}$. 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

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

We finish this section by pointing out two favorable properties of the multi-resolution LCMM. Above, we have established that LCMM removes all arbitrage opportunities expressed by $\mathbf{A}$. The next theorem shows that this actually removes all arbitrage. The proof shows that consecutive levels are coherent, which by transitivity implies that the overall price vector is coherent (see Appendix A.3).

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 $b_k = O(1/k^{2.01})$, it can achieve a *constant* worst-case loss bound. The proof uses the fact that the overall loss is bounded by the sum of losses of level markets, which are at most $b_k \log |\mathcal{Z}_k| = k b_k \log 2$.

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

*4.1.3 A Multi-resolution LCMM Tree.* 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 only keep track of the coordinates accessed so far and represent them as an annotated subtree $T$ of $T^*$, referred to as an *LCMM tree*.

Definition 2 (LCMM Tree). *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 $T$. To calculate prices, we 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*.

## 4.2 Price Queries

There are many ways to decompose an interval $I$ in a multi-resolution market, 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 one-sided intervals of the form $I = [\alpha, 1)$.

Let $T$ be a coherent LCMM tree and $\boldsymbol{\theta} := \boldsymbol{\theta}(T)$ and $\boldsymbol{\eta} := \boldsymbol{\eta}(T)$ be 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 $\boldsymbol{\mu} := \tilde{\boldsymbol{p}}(\tilde{\boldsymbol{\theta}})$. As before, we identify a set of nodes $\mathcal{Z}$ that covers $I$, and then rely on price coherence to calculate each $\mu_z$ along the search path.

Assume that $z$ is not a root node and we know the price of its parent. Let $sib(z)$ denote the sibling of $z$ and $k = level(z)$. We can then relate the price of $z$ to the price of $par(z)$:

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

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

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

**Input:** Interval $I = [\alpha, 1)$, $\alpha \in \Omega$, coherent LCMM tree $T$.
**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: $e_l \leftarrow \exp\{(\theta_{z_l} + B_k \eta_{z_l})/b_k\}$, $e_r \leftarrow \exp\{(\theta_{z_r} + B_k \eta_{z_r})/b_k\}$, $\mu_{z_l} \leftarrow \frac{e_l}{e_l+e_r}\mu_z$, $\mu_{z_r} \leftarrow \frac{e_r}{e_l+e_r}\mu_z$ ▷ calculate prices by Eq. (19)
5: **if** $\alpha < \alpha_{right(z)}$ **then**
6: $z \leftarrow z_l$, $price \leftarrow price + \mu_{z_r}$
7: **else**
8: $z \leftarrow z_r$
9: **return** $price + \frac{\beta_z - \alpha}{\beta_z - \alpha_z} \cdot \mu_z$

Eq. (16) follows by price coherence and Eq. (17) follows by the price calculation in Eq. (1). Thus, we descend the search path to calculate each price $\mu_z$, beginning with $\mu_{root} = 1$. It remains to obtain $\tilde{\theta}_z$, for which we follow the construction of $\mathbf{A}$ in Eq. (14):

$$\tilde{\theta}_z = \theta_z + \sum_{y \in \mathcal{Y}^*} A_{zy}\eta_y = \theta_z + B_k\eta_z - b_k \sum_{y \supset z} \eta_y. \tag{18}$$

Plugging the above equation back in Eq. (17), we obtain[5]

$$\mu_z = \frac{\exp\left(\frac{\theta_z + B_k\eta_z}{b_k}\right)}{\exp\left(\frac{\theta_z + B_k\eta_z}{b_k}\right) + \exp\left(\frac{\theta_{sib(z)} + B_k\eta_{sib(z)}}{b_k}\right)} \cdot \mu_{par(z)}. \tag{19}$$

These steps yield Algorithm 3. The final line of the algorithm addresses the case when the search ends in the leaf $z$ with $\alpha_z < \alpha < \beta_z$. Rather than expanding the tree to its lowest level $K$, 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 have the same price. Therefore, the price of $[\alpha, \beta_z)$ equals $\frac{\beta_z - \alpha}{\beta_z - \alpha_z} \cdot \mu_z$.

The length of search path for $\alpha$ is $prec(\alpha)$, which denotes the bit precision of $\alpha$, defined as the smallest integer $k$ such that $\alpha$ is an integer multiple of $2^{-k}$. As the computation at each node only requires constant time, the time to price $I = [\alpha, 1)$ is $O(prec(\alpha))$, which is bounded above by $O(K)$.

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

## 4.3 Buy and Cost Operations

Different from LMSR, the cost query for a multi-resolution LCMM cannot be directly derived from prices. We instead augment **buy** to implement **cost** by executing **buy** and then reverting all the changes. We focus on $\mathbf{buy}(I, s, T)$ for $I = [\alpha, 1)$. By buying $s$ shares of $[\alpha, 1)$ and then $(-s)$ shares of $[\beta, 1)$, we obtain buying $[\alpha, \beta)$.

We summarize the procedure in Algorithm 4, which performs $\mathbf{buy}(I, s, T)$ and keeps track of $\mathbf{cost}(I, s, T)$. Similar to price queries, we start with a set of nodes $\mathcal{Z}$ that partition $I$, by searching for $\alpha$ and simultaneously calculating prices $\mu_z$ along the way (lines 3–6).

[5]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.

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

**Input:** Quantity $s \in \mathbb{R}$, interval $I = [\alpha, 1)$, $\alpha \in \Omega$, coherent LCMM tree $T$.
**Output:** Cost of $s$ shares of bundle security for $I$, the updated tree $T$.

1: Define subroutines:
NewLeaf($\alpha_0$, $\beta_0$): return a new leaf node $z$ with $I_z = [\alpha_0, \beta_0)$, $\theta_z = 0$, $\eta_z = 0$
RemoveArbitrage($y$, $\mu_{other}$): restore price coherence among submarkets $k \geq level(y)$ following Eq. (20) and update cost
Let $\ell = level(y)$, $y' = sib(y)$, $t = \frac{b_\ell}{B_{\ell-1}} \log\left(\frac{1-\mu_y}{\mu_y} \cdot \frac{\mu_{other}}{1-\mu_{other}}\right)$
$S = \mu_y e^{tB_\ell/b_\ell} + 1 - \mu_y$, $S_{other} = \mu_{other} e^{-t} + 1 - \mu_{other}$
$\eta_y \leftarrow \eta_y + t$, $\mu_y \leftarrow \mu_y e^{tB_\ell/b_\ell}/S$, $\mu_{y'} \leftarrow \mu_{y'}/S$
$cost \leftarrow cost + (b_\ell \log S) + (B_\ell \log S_{other})$
AddShares($z$, $s$): increase shares held in $z$ by $s$, update cost, and restore price coherence among submarkets $k \geq level(z)$
Let $\ell = level(z)$, $z' = sib(z)$, $\mu_{other} = \mu_z$, $S = \mu_z e^{s/b_\ell} + 1 - \mu_z$
$\theta_z \leftarrow \theta_z + s$
$cost \leftarrow cost + (b_\ell \log S)$
$\mu_z \leftarrow \mu_z e^{s/b_\ell}/S$, $\mu_{z'} \leftarrow \mu_{z'}/S$
RemoveArbitrage($z$, $\mu_{other}$)
2: Initialize $z \leftarrow root$, $\mu_z \leftarrow 1$, a global variable $cost \leftarrow 0$
3: **while** $\alpha_z \neq \alpha$ **do**
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: Calculate $\mu_{left(z)}$, $\mu_{right(z)}$, and update $z$ according to $\alpha$ (same as Algorithm 3 lines 3-8)
7: AddShares($z$, $s$)
8: **while** $z$ is not a *root* **do** ▷ remove arbitrage up the search path
9: $z' \leftarrow sib(z)$, $y \leftarrow par(z)$
10: **if** $z' = right(y)$ **then**
11: AddShares($z'$, $s$) ▷ add shares to $z \in \mathcal{Z}$
12: RemoveArbitrage($y$, $\mu_z + \mu_{z'}$)
13: $z \leftarrow y$
14: **return** *cost*

We then proceed back up the search path, adding $s$ shares to nodes within the cover $\mathcal{Z}$ (lines 7–13). Consider one of such node $y \in \mathcal{Z}$ at level $\ell := level(y)$. Increasing $\theta_y$ by $s$ creates price incoherence between the submarket at level $\ell$ and submarkets at all other levels. We design RemoveArbitrage to remove any arbitrage opportunity between level $\ell$ and *all finer levels* with $k > \ell$. We show in Appendix A.6, Lemma 3, that in order to restore coherence, it suffices to update $\eta_y$ by a closed-form amount:

$$t = \frac{b_\ell}{B_{\ell-1}} \log\left(\frac{1-\mu_y}{\mu_y} \cdot \frac{\mu_{other}}{1-\mu_{other}}\right), \tag{20}$$

where $\mu_{other} = \mu_{left(y)} + \mu_{right(y)}$ records the price of $y$ in all the finer levels. This key algorithmic step is enabled by the arbitrage bundle $\mathbf{a}_y$, which corresponds to buying $\phi_y$ on the level $\ell$ while selling securities associated with all descendants of $y$, with their shares appropriately weighted by the respective liquidity values as specified in the constraint matrix $\mathbf{A}$.

The market remains incoherent between $\ell$ and *all coarser levels* $k < \ell$. Since the updates have been localized to the subtree rooted at $y$, we use Lemma 3 again to update $\eta_{par(y)}$ and restore coherence

among all levels $k \geq \ell - 1$ (line 12). We continue in this manner back along the path to root to restore a coherent market.

The algorithm also tracks the total cost of the **buy** transaction by evaluating Eq. (3) in the component submarkets. Note that costs in all submarkets with $k > \ell$ can be evaluated simultaneously thanks to the restored coherence. Since the computations in each accessed node are constant time, Algorithm 4 runs in time $O(prec(\alpha))$.

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

*Remarks.* In Algorithms 3 and 4, we assume that each node $z$ can store a scalar $\mu_z$, which can be modified during the run to support price calculations but is disposed afterwards. The only part of our algorithms that depends on $K$ are the cumulative liquidities $B_\ell = \sum_{k=\ell+1}^{K} b_k$. To remove such dependence, we can 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 a multi-resolution LCMM with $K = L + 1$ and liquidities $b_1, b_2, \ldots, b_L, B'_L$. 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 $O(prec(\alpha) + prec(\beta))$.

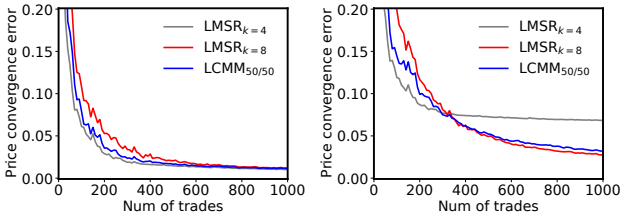
## 5 DISCUSSION AND ILLUSTRATION

We have proposed two cost-function-based market makers that support trading interval securities of arbitrary precision and execute market operations exponentially faster than previous designs. In what situations is one preferable over the other?

The log-time LMSR enjoys better storage and runtime efficiency, because search paths in LMSR tree are shorter thanks to its height-balance property. The log-time LMSR would therefore be computationally preferable, for example, when the designer expects betting interest to be concentrated on a smaller set of intervals. However, the log-time LMSR implements a standard LMSR, which faces well-known design challenges, such as the requirement to set a suitable liquidity value and the precision of bets in advance. Correctly setting these parameters often requires a good estimate of trader interest even before trading in the market starts.

On the other hand, the multi-resolution LCMM does not require a hard specification of the betting precision. Flexible pricing allows the designer to attenuate liquidity across different precisions in a way that best reflects the designer's information-gathering priorities. For example, an LMSR that operates at precision $k = 4$ with liquidity $b$ can be represented by an LCMM with the level liquidity values $\mathbf{b} = (0, 0, 0, b, 0, 0, \ldots)$. Moreover, if the market designer expects most of the information at precision 4 but also wants to support bets up to precision 8, they 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 empirically highlight such flexibility by showing how LCMM can interpolate between LMSRs at different resolutions, allowing the market to match the coarseness of traders' information. We




**(a)** $k = 4$. **(b)** $k = 8$.

**Figure 1: The price convergence error as a function of the number of trades, measured at two resolution levels.**

conduct agent-based simulation using the trader model with exponential utility and exponential-family beliefs [2, 11].[6] We defer the detailed trader model to Appendix B.1. Agents trade with either an LMSR or a multi-resolution LCMM, and we are interested in evaluating market makers' performance in terms of *price convergence error*, calculated as the relative entropy between the *market-clearing price* (that is the price reached when agents only trade among themselves) and the price maintained by the market maker.

We operate in a market over $[0, 1)$ and the outcome is specified with $K = 10$ bits. We consider budget-limited 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 $b = B/\log(2^k)$. Following our motivating example, we compare two LMSR markets at precision levels 4 and 8, denoted as LMSR$_{k=4}$ and LMSR$_{k=8}$, to an LCMM that evenly splits budget to precision levels 4 and 8, denoted as LCMM$_{50/50}$.[7]

Fig. 1 shows the price convergence as a function of the number of trades. As one may expect, LMSR$_{k=4}$ achieves a faster price convergence at the coarser precision level $k = 4$ compared to LMSR$_{k=8}$ (Fig. 1a), but fails to elicit information at any finer granularity by design.[8] The proposed LCMM$_{50/50}$, by equally splitting the budget between $k = 4$ and $k = 8$, is able to interpolate between the performance of LMSR$_{k=4}$ and LMSR$_{k=8}$ and achieves the "best of both worlds": it can elicit forecasts at the finer level $k = 8$ similarly to LMSR$_{k=8}$, but also obtain a fast convergence at the coarser level $k = 4$, almost matching the convergence speed of LMSR$_{k=4}$.

Two immediate questions arise from our work. First, do our constructions generalize to two- or higher-dimensional outcomes? One promising avenue is to combine the ideas from our log-time LMSR market maker with multi-dimensional segment trees [19] to obtain an 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. Second, does our approach extend to non-interval securities, such as call options? We leave these questions open for future research.

[6] We note that while our market makers support agents with any beliefs and utilities, the exponential trader model is convenient, as it allows a closed-form derivation of *market-clearing price* [2, 11], which can be viewed as a "ground truth" for the information elicitation.

[7] The LCMM has an infinite number of choices for its liquidity at each level. We choose LCMM$_{50/50}$ as an instance here to showcase its interpolation ability.

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

## REFERENCES


[1] Jacob Abernethy, Yiling Chen, and Jennifer Wortman Vaughan. 2011. An optimization-based framework for automated market-making. In *Proceedings of the 12th ACM Conference on Electronic Commerce*.

[2] Jacob Abernethy, Sindhu Kutty, Sébastien Lahaie, and Rahul Sami. 2014. Information aggregation in exponential family markets. In *Proceedings of the 15th ACM Conference on Economics and Computation*. 395–412.

[3] G. M. Adel'son-Vel'skiĭ and E. M. Landis. 1962. An algorithm for the organization of information. *Soviet Mathematics—Doklady* 3 (1962), 1259–1263.

[4] Mithun Chakraborty, Sanmay Das, Allen Lavoie, Malik Magdon-Ismail, and Yonatan Naamad. 2013. Instructor rating markets. In *Proceedings of the 27th AAAI Conference on Artificial Intelligence*. 159–165.

[5] Yiling Chen, Lance Fortnow, Nicolas Lambert, David M. Pennock, and Jennifer Wortman Vaughan. 2008. Complexity of combinatorial market makers. In *Proceedings of the 9th ACM Conference on Electronic Commerce*.

[6] Yiling Chen, Lance Fortnow, Evdokia Nikolova, and David M. Pennock. 2007. Betting on permutations. In *Proceedings of the 8th ACM Conference on Electronic Commerce*. 326–335.

[7] Yiling Chen, Sharad Goel, and David M. Pennock. 2008. Pricing combinatorial markets for tournaments. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing*. 305–314.

[8] Yiling Chen and David M. Pennock. 2007. A utility framework for bounded-loss market makers. In *Proceedings of the 23rd Conference on Uncertainty in Artificial Intelligence*.

[9] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. 1999. *Introduction to Algorithms*. The MIT Press.

[10] Miroslav Dudík, Sébastien Lahaie, and David M. Pennock. 2012. A tractable combinatorial market maker using constraint generation. In *Proceedings of the 13th ACM Conference on Electronic Commerce*.

[11] Miroslav Dudík, Sébastien Lahaie, Ryan M Rogers, and Jennifer Wortman Vaughan. 2017. A decomposition of forecast error in prediction markets. In *Advances in Neural Information Processing Systems*. 4371–4380.

[12] Xi Gao, Yiling Chen, and David M. Pennock. 2009. Betting on the real line. In *Proceedings of the 5th Workshop on Internet and Network Economics*.

[13] Mingyu Guo and David M. Pennock. 2009. Combinatorial prediction markets for event hierarchies. In *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems*. 201–208.

[14] Robin D. Hanson. 1999. Decision markets. *IEEE Intelligent Systems* 14, 3 (1999), 16–19.

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

[16] Robin D. Hanson. 2007. Logarithmic market scoring rules for modular combinatorial information aggregation. *Journal of Prediction Markets* 1, 1 (2007), 1–15.

[17] Donald E. Knuth. 1998. *The Art of Computer Programming, Volume 3: Sorting and Searching*. Addison Wesley.

[18] Kathryn Blackmond Laskey, Wei Sun, Robin D. Hanson, Charles Twardy, Shou Matsumoto, and Brandon Goldfedder. 2018. Graphical model market maker for combinatorial prediction markets. *Journal of Artificial Intelligence Research* 63 (2018), 421–460.

[19] Pushkar Mishra. 2016. On Updating and Querying Sub-arrays of Multidimensional Arrays. *CoRR* abs/1311.6093 (2016).

[20] Abraham Othman, David M. Pennock, Daniel M. Reeves, and Tuomas Sandholm. 2013. A practical liquidity-sensitive automated market maker. *ACM Transactions on Economics and Computation* 1, 3 (2013), 14:1–14:25.

[21] Abraham Othman and Tuomas Sandholm. 2010. Automated market-making in the large: The Gates Hillman Prediction Market. In *Proceedings of the 11th ACM Conference on Electronic Commerce*. 367–376.

[22] Abraham Othman and Tuomas Sandholm. 2012. Automated market makers that enable new settings: Extending constant-utility cost functions. In *Auctions, Market Mechanisms, and Their Applications*. 19–30.

[23] Charles R. Plott and Kay-Yut Chen. 2002. Information aggregation mechanisms: Concept, design and implementation for a sales forecasting problem. (2002). Working paper No. 1131, California Institute of Technology.

[24] Lirong Xia and David M. Pennock. 2011. An efficient Monte-Carlo algorithm for pricing combinatorial prediction markets for tournaments. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*. 452–457.