Journal of Artificial Intelligence Research 63 (2018) 421-460 Submitted 04/18; published 11/18

Graphical Model Market Maker for Combinatorial Prediction Markets

Kathryn Blackmond Laskey KLASKEY@GMU.EDU
SEOR Department, George Mason University
Fairfax, VA, 22030 USA

Wei Sun WSUN18@GMAIL.COM
Freddie Mac, McLean, VA, USA

Robin Hanson RHANSON@GMU.EDU
Economics Department, George Mason University
Fairfax, VA, 22030 USA

Charles Twardy CTWARDY@GMU.EDU
Shou Matsumoto SMATSUM2@GMU.EDU
C4I and Cyber Center, George Mason University
Fairfax, VA, 22030 USA

Brandon Goldfedder BRANDON@GOLDFEDDER.COM
Gold Brand Software, 1282 Mason Mill Court
Herndon, VA 20170, USA

Abstract

We describe algorithms for use by prediction markets in forming a crowd consensus joint probability distribution over thousands of related events. Equivalently, we describe market mechanisms to efficiently crowdsource both structure and parameters of a Bayesian network. Prediction markets are among the most accurate methods to combine forecasts; forecasters form a consensus probability distribution by trading contingent securities. A combinatorial prediction market forms a consensus joint distribution over many related events by allowing conditional trades or trades on Boolean combinations of events. Explicitly representing the joint distribution is infeasible, but standard inference algorithms for graphical probability models render it tractable for large numbers of base events. We show how to adapt these algorithms to compute expected assets conditional on a prospective trade, and to find the conditional state where a trader has minimum assets, allowing full asset reuse. We compare the performance of three algorithms: the straightforward algorithm from the DAGGRE (Decomposition-Based Aggregation) prediction market for geopolitical events, the simple block-merge model from the SciCast market for science and technology forecasting, and a more sophisticated algorithm we developed for future markets.

1. Introduction

A prediction market is a market formed for the purpose of making predictions about events of interest. Human or machine forecasters contribute either by directly editing a consensus probability distribution or by buying and selling securities whose prices can be interpreted as probabilities. It is well known that combining forecasts from multiple sources improves accuracy (Surowiecki, 2005) and that prediction markets are among the most accurate information aggregation methods (Wolfers & Zitzewitz, 2006). Prediction markets have

become a popular method for aggregating information from human forecasters (Arrow et al., 2008; Tziralis & Tatsiopoulos, 2007; Horn, Ivens, Ohneberg, & Brem, 2014) and show promise as an ensemble method for machine learning (Barbu & Lay, 2012).

Combinatorial prediction markets allow forecasts not only on base events, but also on conditional and/or Boolean combinations of events. A market-maker-based combinatorial prediction market (Hanson, 2007) allows a user to trade on any event at any time by interacting with an automated market maker which sets the price of a contingent security according to a market scoring rule. The market maker performs several functions: it processes trades on demand, manages a consistent joint probability distribution over the base events, can be queried for any user's expected assets, disallows any trade that could allow a user's total market assets to become negative, and pays off users when the state of any event becomes known.

The joint outcome space for a combinatorial market is exponential in the number of base events. Managing probabilities and assets in a general combinatorial prediction market is therefore intractable. This problem has been addressed by limiting the set of securities that can be traded and designing a market maker that can efficiently price these securities (Pennock & Xia, 2011; Abernethy, Chen, & Vaughan, 2013). The consensus probabilities share information among logically related securities in the restricted set.

While there has been considerable work on the problem of information sharing in the consensus probability distribution, less attention has been devoted to the problem of asset reuse by users trading on logically related securities. In a market without asset reuse, purchasing several logically related securities can unnecessarily tie up assets. For example, suppose a user with a total of \$10 in available cash spends \$7 for a security that pays \$10 if A occurs and \$3 for a security that pays \$10 if A does not occur. The user is guaranteed to recoup her \$10 investment whether or not A occurs. However, without asset reuse, the \$10 is unavailable to make any additional trades until the truth-value of A is revealed. Asset reuse allows users to make purchases as long as they are guaranteed to have sufficient assets in any state of the world to cover their trades.

We present a method for managing assets and probabilities in large combinatorial prediction markets. Our method allows asset reuse, and easily supported live markets with 600 questions and simulated markets with over 10,000 questions. We compare three different asset management approaches including the parallel junction tree method used in the DAGGRE (Decomposition-Based Aggregation) geopolitical forecasting market (Powell, Hanson, Laskey, & Twardy, 2013), the simple block merge method deployed in the SciCast public prediction market (Twardy et al., 2014), and a more sophisticated algorithm we developed but have not yet deployed.

Our core algorithms follow Pennock and Xia (2011) in assuming that allowable securities are structure-preserving. There are several ways to overcome this limitation if participants wish to express knowledge that does not conform to the structure-preserving restriction. In SciCast, we allowed users to add new arcs, but required a minimum investment to offset the additional computation, i.e. computationally expensive arcs required expensive bets.

The remainder of this paper is organized as follows. This section provides background on prediction markets, market makers, and combinatorial prediction markets. Section 2 describes the tasks an automated market maker must perform in a combinatorial prediction market. Section 3 describes the parallel junction tree method and shows how it accomplishes

the market maker tasks. Section 4 introduces the two methods for managing user-specific asset data structures. A numerical performance evaluation of the algorithms is presented in Section 5, followed by a concluding section.

1.1 Prediction Markets

Prediction markets make probabilistic forecasts by allowing participants to trade securities whose value is contingent on events of interest. Prices in such a market can be interpreted as probabilities: the price p in cents of a security paying \$1 contingent on event E is the market probability of E. Traders self-select to make forecasts on the events they believe they know how to forecast. Those with more knowledge tend to gain more assets, giving them greater purchasing power for making new forecasts and greater influence on market probabilities. Conversely, those who make poor forecasts have fewer assets and therefore less influence on market probabilities. Market prices inform everyone of trader information. At each moment, the market probability of an event can be considered as a consensus forecast that aggregates current information possessed by market participants.

Traditionally, prediction markets have used a double auction, where buyers list bid prices for the securities they wish to purchase and sellers list ask prices for the securities they wish to sell. Unfortunately, this double auction approach can give rise to illiquid markets in which traders with information have no incentive to trade (Chen et al., 2010). This problem can be addressed through the use of automated market makers. A market maker stands ready to buy or sell securities on any relevant event. Hanson (2003) introduced a class of market makers based on proper scoring rules (Savage, 1971). Infinitesimal trades with such a market scoring rule (MSR) are fair bets at the price offered by the market maker. Larger trades change the market probabilities. Users increase their expected assets by moving the market probabilities toward their beliefs. The prices offered by a MSR market maker can be viewed as a current trader consensus on the probabilities of those events.

The most commonly applied MSR is the logarithmic market scoring rule (LMSR) (Hanson, 2007). In a LMSR-based prediction market, the market maker varies its price exponentially with the quantity of assets it sells. The LMSR is attractive for combinatorial markets because it is uniquely modular. That is, LMSR is the only MSR for which a trade that changes the probability $p(E|F)$ of an event E given another event F leaves the probability $p(F)$ of the conditioning event unchanged. In other words, the LMSR uniquely allows users with conditional information to focus on trades related to their area of expertise, without unintentionally changing the probabilities of conditioning events about which they have no knowledge.

Most prediction markets focus on buying and selling securities, but MSRs support a formally equivalent view in which participants make direct changes to probabilities. We call such direct changes *edits* to the joint distribution. Offering an edit-based interface may be attractive to subject-matter experts who think in terms of event probabilities rather than security prices. For this reason, we use the terminology of trades and edits interchangeably.

For more information on prediction markets the reader might start with (Wolfers & Zitzewitz, 2006) and then use (Tziralis & Tatsiopoulos, 2007; Horn et al., 2014) as bibliometric guides to the field in 2007 and 2014, respectively. Our focus is combinatorial prediction markets, for which the next section provides background.

1.2 Combinatorial Prediction Markets

A combinatorial prediction market allows edits to Boolean combinations of a base set of events and/or conditional trades on base events given other base events. A market-maker-based combinatorial prediction market, therefore, declares a complete consistent probability distribution over a combinatorial space of events, and lets participants edit any part of that distribution. In a combinatorial LMSR-based market, users can make conditional bets that satisfy intuitive independence properties. For example, letting "¬" denote "not", a trader who increases the value of $p(A|B)$ gains if A and B occur and loses if $\neg A$ and B occur. Such an edit changes neither $p(B)$ nor $p(A|\neg B)$ and the trader neither gains nor loses if $\neg B$ occurs (Hanson, 2007).

With a large set of base events, the number of event combinations becomes astronomical, making it intractable in general to compute market prices and identify infeasible trades. In particular, it is in general NP-hard to maintain correct LMSR prices across an exponentially large outcome space (Chen, Fortnow, Lambert, Pennock, & Wortman, 2008).

The Predictalot public combinatorial market (Malinowski, 2010) achieved tractability by using a Monte Carlo approximation to price securities. Other authors have showed how to achieve tractable pricing without approximation by limiting the securities that can be traded: two key examples follow. Chen et al. (2008) used a Bayesian network to represent the consensus distribution for a set of securities related to outcomes in a tournament, and showed how to perform price updating efficiently when trades are restricted to securities for which updates preserve the structure. Pennock and Xia (2011) gave a general characterization of the class of structure preserving securities for a Bayesian network and proved that prices can be updated in polynomial time if edits are structure preserving.

Abernethy, et al. (2013) introduced a general framework for automated market maker design in combinatorial markets that restrict trades to a relatively small set of structured securities. From a set of intuitive axioms for a reasonable market maker, they prove the market maker must price securities using a convex cost function. Their framework uses optimization over the convex hull of security payoffs to achieve efficient algorithms for pricing securities. Their framework includes LMSR markets as a special case. Li et al. (2013) extended this framework to a parameterized class of markets with adaptive liquidity, and studied the relationship of market parameters to the market's ability to aggregate information and make a profit.

Dudík et al. (2012) proposed a market maker that achieves some information sharing among related securities, without restricting the securities that can be traded. Convex optimization and constraint generation are used to detect and eliminate some kinds of arbitrage opportunities. Loss bounds were derived. While falling short of a full combinatorial market, this approach allows trading on the full combinatorial space while balancing information propagation with tractability. The approach was applied in a large-scale predication market for United States elections (Dudík, Lahaie, Rothschild, & Pennock, 2013).

1.3 Reusing Assets

In the LMSR framework of Pennock and Xia (2011), each edit takes the form of a participant paying cash to a market maker to obtain a contingent security that pays cash if a certain event happens. A trader who has run out of cash is not permitted to make any more trades.

Yet the assets one is sure to obtain from prior trades are often sufficient to guarantee many more trades.

As a simple example, suppose a user buys a security "Pays \$10 if A," and then later buys a security "Pays \$10 if $\neg A$." Because one of these is guaranteed to pay off, the two are together worth \$10 in cash. However, in a naïve implementation, this \$10 cannot be used to make additional trades until the truth-value of A is resolved.

As another example, consider a user who wishes to trade on conditional probabilities for an event A given N mutually exclusive conditions B_i, $i = 1 \ldots N$. In the Pennock and Xia framework, such conditional probabilities can be established by trading securities that depend on joint states. A market probability $p(A|B_i) = x_i$ corresponds to a market price of $\$x_i$ to purchase two separate securities, one paying \$1 if B_i and A both occur and the other paying $\$x_i$ if $\neg B_i$ occurs. Trading on A given all N of the B_i requires purchasing N separate pairs of securities, where the i^{th} pair costs $\$x_i$ and pays \$1 if A and B_i occur, 0 if $\neg A$ and B_i occur, and $\$x_i$ if $\neg B_i$ occurs. Without asset reuse, the total purchase price of $\$\sum x_i$ would be tied up until one of the B_i occurred. Note, though, that because no more than one of the B_i can occur, the collection of securities is guaranteed to pay off at least $\$(\sum x_i - \max\{x_i\})$. If N is large and the probabilities are non-negligible, a considerable sum could be unnecessarily tied up.

While we might imagine that a market maker could be designed to notice and account for these kinds of asset reuse opportunities, combinations of trades worth a guaranteed amount of cash are difficult to identify in general. The market maker needs to be able to ensure that each participant has sufficient assets to cover any losses that could occur. Thus, in order to allow asset reuse, the market maker must be able to determine whether an attempted edit could result in negative assets in some state of the world, and if so, disallow the attempt. When there are complex combinations of conditional trades, this can be challenging. The market maker algorithms presented in Sections 3 and 4 provide computationally efficient ways to identify and exploit asset reuse opportunities.

2. Market Maker Tasks

Consider a LMSR market maker market with base events of the form $V = v_k$, where V is a question posed to market participants and $\{v_1, \ldots v_m\}$ is a set of of mutually exclusive and collectively exhaustive possible answers to the question. For example, V might be the question of who will win a given election, $\{v_1, \ldots v_m\}$ might be the list of candidates running for the position, and $V = v_k$ would represent the event that the k^{th} candidate wins the election. Following standard terminology, V is called a *random variable* and the set $\{v_1, \ldots v_n\}$ consists of the states or possible outcomes of V. A traditional or "flat" prediction market assigns a probability to each base event $V = v_k$.

In a combinatorial prediction market with n random variables $V_1, \ldots, V_n$, the market consensus distribution assigns a probability to each of the $\prod_k n_k$ possible joint outcomes of the random vector $\mathbf{V} = (V_1, \ldots, V_n)$.[1] From this joint probability assignment can be derived probabilities for the base events $V_j = v_{j_k}$, as well as probabilities of more complex events such as logical combinations of events or conditional events.

1. We use capital letters for random variables, lowercase letters for variable values, and bold letters for vectors.

At any time, a user u possesses assets consisting of the collection of securities u has purchased up to that time together with the cash u has remaining after purchases. The value of these assets depends on the outcomes of the events on which u has traded. Once the joint state $\mathbf{v}$ of all market variables becomes known, all u's securities will be paid out for their cash value in state $\mathbf{v}$. At any point during trading, the amount of cash the user would receive if a given state $\mathbf{v}$ were revealed to be the true state, denoted by $a^u_{\mathbf{v}}$, is called the user's current assets for that joint state. For example, suppose a user starts with \$100 and spends \$0.70 to purchase a security paying \$1 if A occurs. This user will have assets \$100.30 in any state in which A occurs and \$99.30 in any state in which A does not occur. If the user then spends an additional \$0.40 to purchase a conditional security that pays \$1 for B given A, then the user's new assets will be \$100.90 if A and B both occur, \$99.90 if A occurs and B does not, and \$99.30 if A does not occur.

The market maker maintains the current consensus joint market probability distribution, sets prices on contingent securities, and maintains an account of each user's state-dependent assets. This section discusses six tasks the market maker must perform in order to manage probabilities and assets.

Task 1: First, the market maker needs to maintain a consistent consensus distribution $p_{\mathbf{v}} = p(\mathbf{V} = \mathbf{v})$ over joint states $\mathbf{v}$ of $\mathbf{V}$ and respond to queries about features of the current consensus distribution. In addition to queries $p(V = v)$ about the probabilities of base events, a combinatorial market may allow queries $p(T|\mathbf{H} = \mathbf{h})$ for the distribution of a target variable T given that assumption (or hypothesis) variables $\mathbf{H}$ have values $\mathbf{h}$. Queries may also involve Boolean combinations, *e.g.,* $p((W = w) \vee (U = u))$. While intractable for general combinatorial markets, this task has been accomplished either by restricting the securities that can be priced (Pennock & Xia, 2011; Abernethy et al., 2013) or by approximation (Malinowski, 2010; Dudík et al., 2012).

Task 2: The second market maker task is to process edits. A user who disagrees with the current consensus distribution $p_{\mathbf{v}}$ can change it by making edits, provided the edits can be covered by her current assets. If user u begins with assets $a^u_{\mathbf{v}} \geq 0$ in joint state $\mathbf{v}$ and makes an edit changing $p(\mathbf{v})$ to $x(\mathbf{v})$, the market maker needs to update u's assets according to the LMSR asset updating rule. User u's post-edit assets in joint state $\mathbf{v}$ are given by:

$$a'^u_{\mathbf{v}} = a^u_{\mathbf{v}} + b \ln \frac{x(\mathbf{v})}{p(\mathbf{v})}, \tag{1}$$

where b is a liquidity parameter. That is, u's assets change in proportion to the logarithm of the ratio of new to old probabilities. Edits by u leave other users' assets unchanged.

Edits typically involve only small subsets of the base events. For example, u might make an edit to change the conditional distribution $p(T|\mathbf{H} = \mathbf{h})$ to a new distribution $x(T|\mathbf{H} = \mathbf{h})$. Such an edit changes only the conditional distribution of T given $\mathbf{H} = \mathbf{h}$. The marginal distribution of the variables other than T remains unchanged, as does the distribution of T given $\mathbf{H} \neq \mathbf{h}$. Specifically, $p(\mathbf{v})$ becomes $\frac{x(T|\mathbf{H}=\mathbf{h})}{p(T|\mathbf{H}=\mathbf{h}))} p(\mathbf{v})$ in states $\mathbf{v} = (t, \mathbf{h}, \mathbf{w})$ and remains unchanged in states $\mathbf{v}' = (t, \mathbf{h}', \mathbf{w})$ for which $\mathbf{h}' \neq \mathbf{h}$. From (1) we see that this conditional edit changes u's assets $a^u_{(t,\mathbf{h},\mathbf{w})}$ to

$$a'^u_{(t,\mathbf{h},\mathbf{w})} = a^u_{(t,\mathbf{h},\mathbf{w})} + b \ln \frac{x(t|\mathbf{H} = \mathbf{h})}{p(t|\mathbf{H} = \mathbf{h})}. \tag{2}$$

where $\mathbf{w}$ consists of all variables other than t and $\mathbf{h}$. For $\mathbf{h}' \neq \mathbf{h}$, assets $a'^u_{(t,\mathbf{h}',\mathbf{w})} = a^u_{(t,\mathbf{h}',\mathbf{w})}$ remain unchanged. Thus, u's assets increase in states $(t, \mathbf{h}, \mathbf{w})$ for which the edit has increased the probability of t given $\mathbf{h}$; decrease in states $(t, \mathbf{h}, \mathbf{w})$ for which the edit has decreased the probability of t given $\mathbf{h}$; and remain unchanged in states $(t, \mathbf{h}', \mathbf{w})$ for which $\mathbf{h}' \neq \mathbf{h}$.

In summary, the market maker's edit processing task is to update the consensus joint distribution after each edit and to update assets of the user making the edit according to (1), which becomes (2) for conditional edits. As with the first task, this task is intractable in full generality, but can be approximated, or made tractable by restricting the allowable securities.

Task 3: The third market maker task is to ensure that users can cover their bets. The market maker has to disallow any edit for which the new asset value $a'^u_{\mathbf{v}}$ would be negative in any state $\mathbf{v}$. Therefore, the market maker should be able to calculate the minimum asset value $c^u = \min_{\mathbf{v}} a^u_{\mathbf{v}}$, also known as the user's cash, in order to ensure that it always remains non-negative. When a user attempts an edit, the market maker can provisionally update the asset structure, calculate the user's post-edit cash, and process the edit only if the result is non-negative.

Rather than having users attempt edits that may be disallowed, it is convenient for the market maker to provide upper and lower bounds in advance of edits. Edit limits are straightforward to calculate if the market maker is able to find conditional minimum assets. Denote the conditional cash given $\mathbf{Y} = \mathbf{y}$ as:

$$c^u_{\mathbf{y}} = \min_{\mathbf{v}:\mathbf{Y}=\mathbf{y}} a^u_{\mathbf{v}}$$

Suppose the user wants to make an edit to change $p(T|\mathbf{H} = \mathbf{h})$, denoted as $p_{t|\mathbf{h}}$, to $x(T|\mathbf{H} = \mathbf{h})$, denoted as $x_{t|\mathbf{h}}$. The updated assets will remain non-negative as long as both $c^u_{(t,\mathbf{h})} + b \ln\{x_{t|\mathbf{h}}/p_{t|\mathbf{h}}\}$ and $c^u_{(\neg t,\mathbf{h})} + b \ln\{(1 - x_{t|\mathbf{h}})/(1 - p_{t|\mathbf{h}})\}$ are non-negative. That is, the new probability $x_{t|\mathbf{h}}$ must fall within the edit limits $x_* \leq x_{t|\mathbf{h}} \leq x^*$, where:

$$\begin{aligned} x_* &= p_{t|\mathbf{h}} e^{-\left(c^u_{(t,\mathbf{h})}/b\right)}, \text{ and} \\ x^* &= 1 - (1 - p_{t|\mathbf{h}}) e^{-\left(c^u_{(\neg t,\mathbf{h})}/b\right)}. \end{aligned} \tag{3}$$

Thus, the ability to calculate conditional minimum assets allows the market maker to give users upper and lower bounds on probability changes that can be supported by their assets.

The problem of finding a user's minimum assets has received little attention in the literature. Market makers for combinatorial prediction markets typically do not consider asset reuse.

Task 4: In addition to responding to probability queries, processing edits and computing unconditional and conditional minimum assets, the fourth market maker task is to compute the market value, or expected value, of any user's portfolio of assets:

$$\bar{a}^u = \sum_{\mathbf{v}} p_{\mathbf{v}} a^u_{\mathbf{v}}.$$

Note that if a user u makes an edit $x(t|\mathbf{H} = \mathbf{h})$ to move the consensus distribution in the direction of her subjective probability for t given $\mathbf{h}$, her subjective expected assets will increase. Thus, users are incentivized to move the market distribution toward their subjective beliefs.

Task 5: Over time, as the values of some components of $\mathbf{V}$ become known, the market maker needs to update its representations of $p_{\mathbf{v}}$ and $a^u_{\mathbf{v}}$ accordingly. When it becomes known that the random variable W has value w, the system can throw away all representations associated with states where $W \neq w$. The remaining probabilities must then be renormalized so that $\sum_{W=w} p_{\mathbf{v}} = 1$. Assets $a^u_{\mathbf{v}}$ for states $\mathbf{v}$ with $W = w$ do not change. The cash for user u changes from c^u to c^u_w. This fifth market maker operation, removing a random variable and renormalizing when its value becomes known, is called *resolving* W to the value w.

Task 6: Finally, new questions of interest arise from time to time. To allow forecasts on new questions, the market maker must perform a sixth task of adding new random variables and specifying a joint distribution for the new and old random variables. When a new random variable W is added to the system, each previous state $\mathbf{v}$ is replaced by a set of states $\{(\mathbf{v}, w)\}$, one for each of the possible values w of W. Assets are directly transferred via $a^u_{(\mathbf{v},w)} = a^u_{\mathbf{v}}$. Probabilities $p_{(\mathbf{v},w)} = p_{\mathbf{v}} p_{w|\mathbf{v}}$ are assigned, where $p_{w|\mathbf{v}}$ is the initial conditional probability for state w of W given $\mathbf{v}$, and $\sum_w p_{w|\mathbf{v}} = 1$. This initial probability distribution is arbitrary and can be set by the market maker. For simplicity, it is commonly set to a uniform distribution. This bounds the loss suffered by a LMSR market maker from adding the new random variable to $-b \ln \#W$, where $\#W$ is the number of possible values of W. If the market maker has knowledge about W, the market maker's expected loss is minimized by setting $p_{w|\mathbf{v}}$ to the market maker's conditional probability of w given $\mathbf{v}$.

To summarize, a combinatorial prediction market needs data structures and algorithms to accomplish the following tasks:

1. *Perform probability query* to find the current conditional distribution $p(T|\mathbf{H} = \mathbf{h})$ for target variable T given assumption $\mathbf{H} = \mathbf{h}$.

2. *Apply edit* by user u to the distribution of one or more variables, e.g., a change from $p(T|\mathbf{H} = \mathbf{h})$ to $x(T|\mathbf{H} = \mathbf{h})$.

3. *Determine allowable edits.* This task includes finding unconditional and conditional cash c^u and c^u_w, determining upper and lower edit limits, and checking whether an attempted edit is allowable.

4. *Find expected assets* $\bar{a}^u$ of user u with respect to the current market distribution.

5. *Resolve variable* W to value w.

6. *Add variable* W with possible values $\{w_1, \ldots, w_{\#W}\}$.

3. Parallel Junction Tree Market Maker

Hanson's (2007) combinatorial market maker algorithm employed a naïve brute-force enumeration approach that quickly becomes intractable as the number of market questions grows. In a combinatorial market with n base events, each of which has two possible outcomes (e.g.,

true and *false*), the market maker must maintain a consensus probability distribution over 2^n joint states. With 20 base events, there are over a million probabilities; with 30 base events, there are over a billion probabilities; with 40 base events, there are over a trillion probabilities. Clearly, brute force enumeration does not scale to combinatorial markets with more than a few base questions.

The problem of efficiently representing and computing with joint distributions over large numbers of random variables arises in a wide range of applications and has been studied extensively. The standard approach is to represent the joint distribution as a product of factors, where each factor involves only a small number of random variables. The number of parameters required to specify the joint distribution then scales as the number of factors times the number of parameters required to specify the largest factor. This generally provides a drastic reduction in the number of parameters needed to specify the joint distribution. In addition to representation economy, factored representations also typically admit efficient query processing and probability updating algorithms.

Pennock and Xia (2011) introduced a factored representation for probabilities in a combinatorial prediction market, but did not provide a means to allow asset reuse and did not discuss how to compute expected assets. This section presents a probability and asset management approach, based on probabilistic graphical models, that allows asset reuse and supports computation of expected values. Section 4 introduces a new asset management approach that is more efficient when trades are sparse in the full joint space.

3.1 Probabilistic Graphical Models for Market Joint Distribution

Probabilistic graphical models have emerged as a general framework for efficient representation of and inference with joint probability distributions over many random variables. A probabilistic graphical model $\mathcal{M} = (\mathcal{G}, \mathcal{P})$ consists of a graph $\mathcal{G}$ representing direct dependencies among related random variables and a set $\mathcal{P}$ of functions, defined on clusters of random variables, representing numerical probability information. For example, Figure 1 shows the graph $\mathcal{G}$ for a directed graphical model, also called a Bayesian network, representing the dependency structure for a group of eighteen related questions that appeared in the SciCast (Twardy et al., 2014) combinatorial prediction market in 2014. The graph $\mathcal{G}$ is an acyclic directed graph in which each node represents a random variable. The nodes are shown as rounded boxes labeled with short phrases; full question text is provided in Table 1. Arcs in the graph, shown as arrows, represent direct dependence relationships, e.g., Arctic Sea ice volume depends on Arctic Sea ice thickness and Arctic Sea ice extent.

In a Bayesian network $\mathcal{B} = (\mathcal{G}, \{\mathcal{P}_k\})$ defined on random variables $\mathbf{V} = (V_1, \ldots, V_n)$, each random variable V_k is conditionally independent of its non-descendents given its parents. With each node V_k is associated a set

$$\mathcal{P}_k = \{p(V_k = v_k | \mathbf{V}_{Pa(V_k)} = \mathbf{v}_{Pa(V_k)})\} \tag{4}$$

of local distributions, where $\mathbf{V}_{Pa(V_k)}$ denotes the parents of V_k in $\mathcal{G}$ and $p(V_k = v_k | \mathbf{V}_{Pa(V_k)} = \mathbf{v}_{Pa(V_k)})$ denotes the conditional probability that V_k has outcome v_k given that its parents $\mathbf{V}_{Pa(V_k)}$ have outcomes $\mathbf{v}_{Pa(V_k)}$. For a root node, the local distribution assigns a probability to each possible value; otherwise it assigns a conditional distribution given each combination of values of its parents.

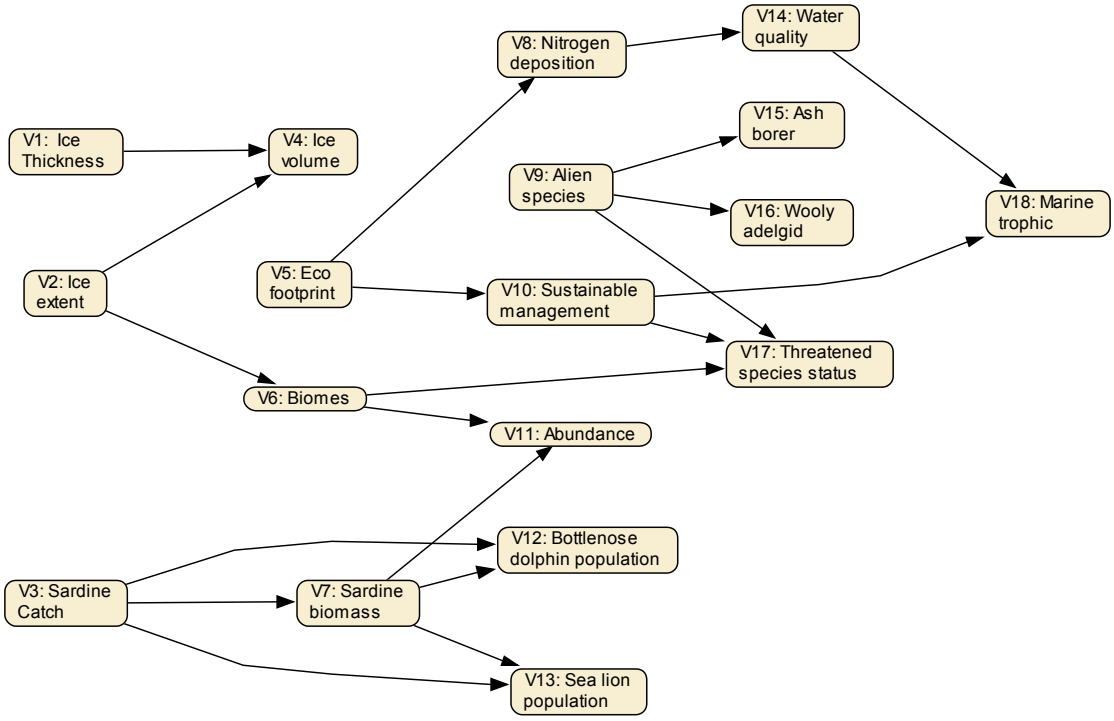

Figure 1: Bayesian Network for a Subset of SciCast Questions.

The graph $\mathcal{G}$ and local distributions $\{\mathcal{P}_k\}$ together represent a joint probability distribution over $\mathbf{V}$. This joint distribution can be written in factored form as:

$$p(\mathbf{V} = \mathbf{v}) = \prod_{1 \leq k \leq n} p(V_k = v_k | \mathbf{V}_{Pa(V_k)} = \mathbf{v}_{Pa(V_k)}). \tag{5}$$

If no node has more than r states and s parents, then specifying the joint distribution requires no more than $n(r-1)^s$ probabilities. This is typically a vast reduction: for 40 random variables with 4 states each, brute force enumeration requires more than 10^{24} probabilities, whereas a Bayesian network with no more than four parents per node can be specified fewer than $3,300$ probabilities. A general joint distribution for the random variables of Figure 1 would require over 200 billion probabilities to specify by brute force, whereas this Bayesian network requires about 1,500 probabilities to specify.

In addition to simplifying specification, probabilistic graphical models admit inference algorithms that exploit the independence relationships for efficient computation. When the graph is singly connected–that is, when there is only one path between any two nodes in the graph–there are elegant message-passing schemes for managing probability updates and query processing. For multiply connected graphical models such as Figure 1, the most popular inference method is the junction tree algorithm (Lauritzen & Spiegelhalter, 1988). This algorithm first transforms the original graphical model into a new singly connected graphical model and then applies a message-passing scheme on the transformed model. The transformation preserves the joint probability distribution represented by the original model.

Table 1: Full Question Text for SciCast Question Subset

Variable	Full Text of Question Appearing on SciCast Market
V1	What will be the daily average Arctic sea ice thickness for September 2014?
V2	What will the average Arctic sea ice extent be for September 2014?
V3	How will the U.S. catch of Pacific sardines in 2014 change from the 2013 estimated catch of 112,296 metric tons?
V4	What will be the monthly averaged Arctic ice volume for September 2014?
V5	Which of the following changes will be reported about "Ecological footprint and related concepts" in the fourth edition of the Global Biodiversity Outlook report?
V6	Which of the following changes will be reported about "trends in extent of selected biomes, ecosystems, and habitats" in the fourth edition of the Global Biodiversity Outlook report?
V7	How will NOAA Fisheries 2015 stock biomass estimate of the Pacific sardine change compared to the 2014 estimate of 378,120 metric tons?
V8	Which of the following changes will be reported on the topic of Nitrogen deposition in the fourth edition of the Global Biodiversity Outlook report?
V9	Which of the following changes will be reported about "Trends in invasive alien species" in the fourth edition of the Global Biodiversity Outlook report?
V10	Which of the following changes will be reported about "Area of forest, agricultural and aquaculture ecosystems under sustainable management" in the fourth edition of the Global Biodiversity Outlook report?
V11	Which of the following changes will be reported about "trends in abundance and distribution of selected species" in the fourth edition of the Global Biodiversity Outlook report?
V12	How will NOAA Fisheries' next minimum population estimate of California/Oregon/Washington offshore common bottlenose dolphins change from the 2010 stock assessment of 684 bottlenose dolphins?
V13	How will NOAA Fisheries' next minimum population estimate of the U.S. stock of California sea lions change from the 2011 stock assessment?
V14	Which of the following changes will be reported about "Water quality of aquatic ecosystems" in the fourth edition of the Global Biodiversity Outlook report?
V15	Will any new infestations of Emerald Ash Borer, an invasive insect species, be observed in the U.S. in 2014?
V16	Will the number of U.S. counties with infestations of Hemlock Woolly Adelgids, an invasive insect species, decrease in 2014 from the previous year?
V17	Which of the following will be reported about "Change in status of threatened species" in the fourth edition of the Global Biodiversity Outlook report?
V18	Which of the following changes will be reported about "The Marine Trophic Index" in the fourth edition of the Global Biodiversity Outlook report?

The graph for the transformed model is an undirected graph called a *junction tree*. The nodes of the junction tree represent clusters of related random variables from the original Bayesian network.

Figure 2 shows a junction tree constructed from the Bayesian network of Figure 1. The clusters, shown as ovals in the figure, are arranged in a tree. Each arc in the graph is labeled with the intersection, or *separator*, of its endpoints.

For a probabilistic graphical model on a junction tree composed of a set of clusters $\mathcal{C}$ and separators $\mathcal{S}$, the joint probability distribution $p(\mathbf{v})$ factors according to:

$$p(\mathbf{V} = \mathbf{v}) = \frac{\prod_{c \in \mathcal{C}} p(\mathbf{V}_c = \mathbf{v}_c)}{\prod_{s \in \mathcal{S}} p(\mathbf{V}_s = \mathbf{v}_s)}. \tag{6}$$

Here, $\mathbf{V}_c$ and $\mathbf{V}_s$ denote the random variables in cluster c and separator s, respectively, and $p(\mathbf{V}_c = \mathbf{v}_c)$ and $p(\mathbf{V}_s = \mathbf{v}_s)$ are the marginal distributions for the cluster and separator variables, respectively.

The junction tree algorithm works as follows. Each cluster c [separator s] is initialized with a *potential function*, denoted by $\psi(\mathbf{v}_c)$ [$\psi(\mathbf{v}_s)$], such that the ratio

$$\frac{\prod_{c \in \mathcal{C}} \psi_c(\mathbf{v}_c)}{\prod_{s \in \mathcal{S}} \psi_s(\mathbf{v}_s)}. \tag{7}$$

is proportional to $p(\mathbf{V} = \mathbf{v})$. After initialization, the cluster and separator potentials are updated by passing messages along the arcs. Each updating step leaves the product (7) invariant. At termination of the algorithm, each cluster and separator potential function is proportional to the joint marginal distribution of its random variables, i.e., at termination, $\psi(\mathbf{v}_c) \propto p_c(\mathbf{V}_c = \mathbf{v}_c)$ and $\psi(\mathbf{v}_s) \propto p(\mathbf{V}_s = \mathbf{v}_s)$ for all clusters c and separators s.

When evidence is obtained, the junction tree algorithm can be used to propagate the effects of this new information to other related random variables. For example, suppose we learn that a random variable V has value v. First, we choose a cluster c containing V (if it belongs to more than one cluster, we can choose one arbitrarily). Then, we modify the potential function for this cluster by setting it to zero for any state $\mathbf{v}_c$ in which $V \neq v$, leaving it unchanged for states $\mathbf{v}_c$ in which $V = v$. Then message passing is applied to propagate the changes to the other clusters. At termination, each potential is proportional to the conditional distribution of its random variables given $V = v$.

The computational complexity of the junction tree algorithm is exponential in the treewidth of the junction tree, defined as the size of the largest cluster in the junction tree. Although inference in arbitrary graphical models is NP-hard (Cooper, 1990), models with tractable treewidth have been defined for many applications.

Henceforth, we assume a LMSR combinatorial prediction market in which the consensus probability distribution $p(\mathbf{v})$ is represented as a probabilistic graphical model that can be compiled into a junction tree.

The parallel junction tree (PJT) method maintains a set of junction trees, all having the same clusters and separators. One junction tree represents the consensus joint distribution; a separate junction tree of the same structure represents each user's assets. The following subsections describe how the PJT method performs the six market maker tasks. Section 4 then introduces a more efficient approach to asset management.

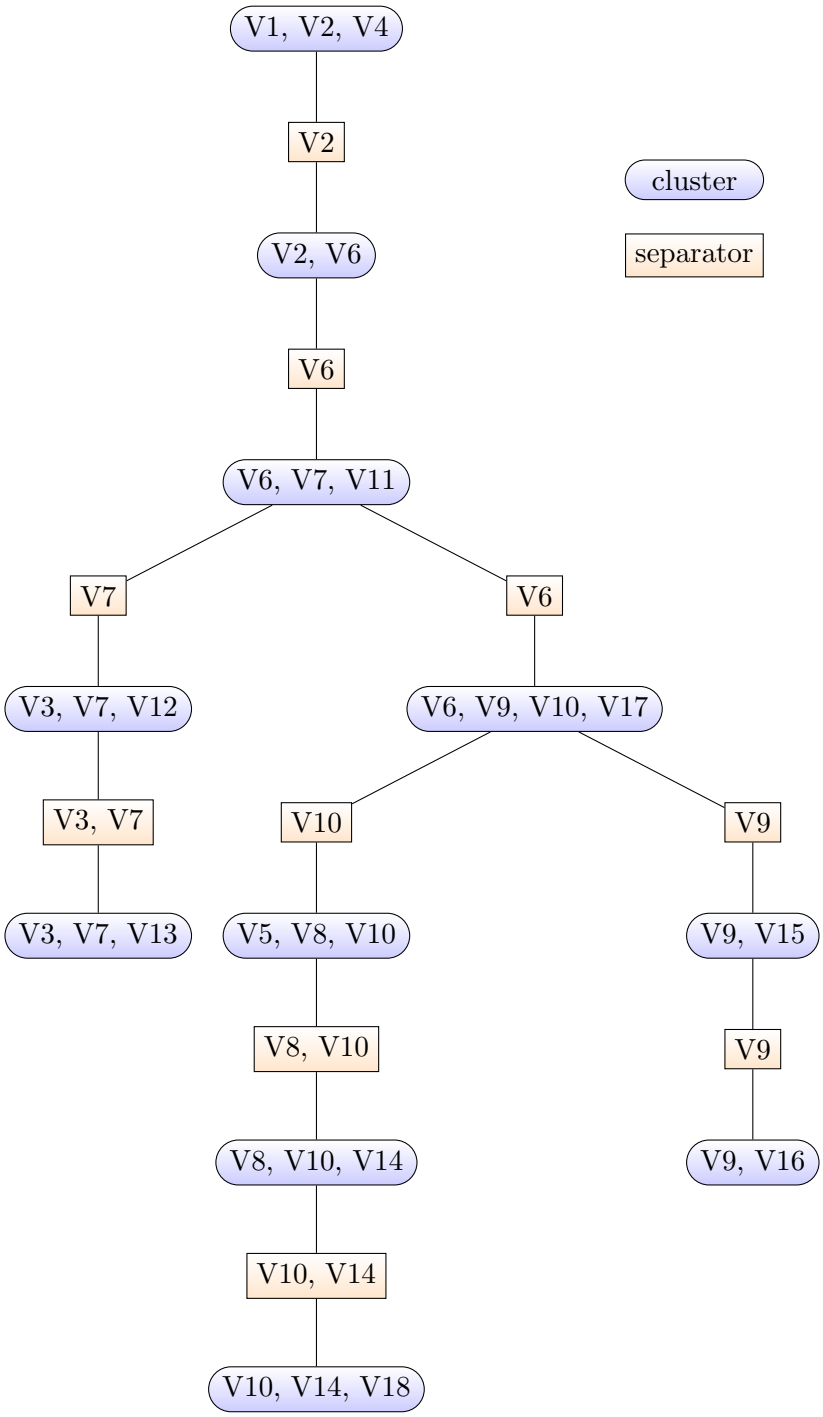

Figure 2: Junction Tree for Bayesian Network of Figure 1.

3.2 Probability Queries

The first market maker task is to process probability queries. If the current consensus distribution is maintained as a junction tree (6), the junction tree algorithm (Lauritzen & Spiegelhalter, 1988) is applied directly to perform probability calculations such as finding the marginal distribution of a random variable T or the conditional distribution of T given $\mathbf{H} = \mathbf{h}$.

3.3 Structure Preserving Edits: Probability Updating

When a user makes an edit to change the conditional probability $p(t|\mathbf{H} = \mathbf{h})$ to a new probability $x(t|\mathbf{H} = \mathbf{h})$, the market maker needs to update the distributions of all the other random variables to account for the new information. This operation makes use of a generalization of Bayes' rule, known as Jeffrey's rule (Jeffrey, 1990), to handle evidence declaring a new non-extreme probability distribution. This kind of evidence is known as *soft* evidence, in contrast to traditional (hard) evidence asserting a definite value for one of the random variables.

The junction tree algorithm is adapted in a straightforward way to handle soft evidence (Koski & Noble, 2009; Langevin & Valtorta, 2008; Valtorta, Kim, & Vomlel, 2002). To update the consensus distribution when the user changes $p(t|\mathbf{h})$ to $x(t|\mathbf{h})$, the market maker asserts soft evidence to set a new conditional probability $x(t|\mathbf{h})$ of $T = t$ given $\mathbf{H} = \mathbf{h}$ and update the remaining probabilities according to Jeffrey's rule. Formally, asserting soft evidence is equivalent to introducing a hidden "dummy" random variable D with parents T and H, setting the conditional probabilities $p(D = 1|t, \mathbf{h})$ to values proportional to $x(t|\mathbf{h})/p(t|\mathbf{h})$, and then declaring $D = 1$ as ordinary evidence (Pearl, 1990).

Probability updating after an edit is straightforward if there is at least one cluster containing all the variables mentioned in the edit. Pennock and Xia (2011) call such an edit *structure preserving* for the junction tree. In general, clusters in the junction tree contain variables that are closely related to each other. In particular, if the joint distribution is represented as a Bayesian network, and if two variables in the Bayesian network are connected directly, then there will be a cluster in the junction tree containing both variables. Thus, conditional edits of directly connected variables are structure preserving, as are edits on Boolean combinations of directly connected variables. Variables related only through a chain of connections may not be structure preserving. For example, $V5$ and $V14$ are indirectly related via $V9$ in the Bayesian network of Figure 1, but are not in a common cluster in Figure 2. Thus, edits involving both $V5$ and $V14$ are not structure preserving with respect to the junction tree of Figure 2. Further discussion about non structure preserving edits is in Section 3.9.

Suppose the user wishes to make a structure preserving edit to: (1) set $p(t|\mathbf{h})$ to $x(t|\mathbf{h})$; (2) proportionally adjust probabilities of $t' \neq t$ given $\mathbf{h}$; and (3) leave $p(t|\mathbf{h}')$ unchanged for $\mathbf{h}' \neq \mathbf{h}$. This task can be accomplished by using a standard probability updating algorithm to apply soft evidence to the consensus distribution junction tree. To do this, the market maker can add a new random variable D with possible values 1 and 0, add the cluster $(D, T, \mathbf{H})$ to the junction tree, and define the potential function $\psi(T, \mathbf{H}, D)$ of the new cluster

as follows:

$$\begin{aligned}
&\psi(t, \mathbf{h}, 1) = m\frac{x(t|\mathbf{h})}{p(t|\mathbf{h})} \\
&\psi(t', \mathbf{h}, 1) = m\frac{1 - x(t|\mathbf{h})}{1 - p(t|\mathbf{h})}p(t'|\mathbf{h}) && t' \neq t \\
&\psi(t'', \mathbf{h}', 1) = m && \text{all } t'', \mathbf{h}' \neq \mathbf{h} \\
&\psi(t'', \mathbf{h}'', 0) = 1 - \psi(t'', \mathbf{h}'', 1) && \text{all } t'', \text{ all } \mathbf{h}'',
\end{aligned}$$

where $m = max\{\frac{p(t|\mathbf{h})}{x(t|\mathbf{h})}, \frac{1-p(t|\mathbf{h})}{1-x(t|\mathbf{h})}p(t'|\mathbf{h})\}$ is chosen so that $\psi(T, \mathbf{H}, D)$ is strictly non-negative.

The market maker then declares $D = 1$ as evidence and applies the standard junction tree algorithm to update the probabilities of all random variables. To see that this operation results in the desired new probabilities, first note that declaring $D = 1$ as evidence will set $\psi(\cdot, \cdot, 0)$ to zero for all values of T and $\mathbf{H}$. After declaring evidence, the algorithm will choose a cluster containing T and $\mathbf{H}$ and multiply its potential by $\psi(\cdot, \cdot, 1)$. This operation will multiply cluster table entries by the constant m for states $(t'', \mathbf{h}')$ for which $\mathbf{h}' \neq \mathbf{h}$, leaving the conditional distribution of T given $\mathbf{h}'$ unchanged. The cluster table entry for $(t, \mathbf{h})$ will be multiplied by $m\frac{x(t|\mathbf{h})}{p(t|\mathbf{h})}$ and entries for $(t', \mathbf{h})$ for $t' \neq t$ will be multiplied by $m\frac{1-x(t|\mathbf{h})}{1-p(t|\mathbf{h})}p(t'|\mathbf{h})$. Therefore: (1) the old marginal probability $p(t|\mathbf{h})$ is multiplied by $m\frac{x(t|\mathbf{h})}{p(t|\mathbf{h})}$; and (2) the old marginal probability $p(t'|\mathbf{h})$ is multiplied by $m\frac{1-x(t|\mathbf{h})}{1-p(t|\mathbf{h})}p(t'|\mathbf{h})$. The resulting conditional probabilities given $\mathbf{H} = \mathbf{h}$ are therefore proportional to $mx(t|\mathbf{h})$ and $m\frac{1-x(t|\mathbf{h})}{1-p(t|\mathbf{h})}p(t'|\mathbf{h})$. After normalizing, the new conditional probability of t given $\mathbf{h}$ will be equal to $x(t|\mathbf{h})$ and the probabilities of $t' \neq t$ will be multiplied by $\frac{1-x(t|\mathbf{h})}{1-p(t|\mathbf{h})}$. The impact of the edit will be propagated to the rest of the network by the junction tree algorithm.

More efficiently, a market maker working directly with the junction tree representation need not explicitly represent the dummy variable D. The market maker can simply multiply the potential function for a cluster containing both T and $\mathbf{H}$ by $\psi(T, \mathbf{H}, 1)$ and update probabilities accordingly.

In the foregoing, the edit was assumed to be structure preserving. Non structure preserving edits can be viewed as changing the structure of the Bayesian network, an operation that requires separate treatment (see Section 3.9 below).

3.4 Structure Preserving Edits: Asset Updating

The previous section addressed probability updating by applying standard algorithms for graphical models. This section introduces a new method for using a junction tree to represent assets and process structure preserving edits. This new method enables asset reuse as well as providing tractable probability and asset updating.

Consider a user making a structure preserving edit of $p(t|\mathbf{H} = \mathbf{h})$ to $x(t|\mathbf{H} = \mathbf{h})$. As discussed above, the effect of this edit is to change the probability of the full joint state $\mathbf{v}$

from $p(\mathbf{v})$ to $x(\mathbf{v})$ as follows:

$$\begin{aligned} x(t, \mathbf{h}, \mathbf{w}) &= \frac{x(t|\mathbf{H}=\mathbf{h})}{p(t|\mathbf{H}=\mathbf{h})} p(t, \mathbf{h}, \mathbf{w}) \\ x(t', \mathbf{h}, \mathbf{w}) &= \frac{1 - x(t|\mathbf{H}=\mathbf{h})}{1 - p(t|\mathbf{H}=\mathbf{h})} p(t', \mathbf{h}, \mathbf{w}) && t' \neq t \\ x(t', \mathbf{h}', \mathbf{w}) &= p(t', \mathbf{h}', \mathbf{w}), && \mathbf{h}' \neq \mathbf{h} \end{aligned}$$

where $\mathbf{w}$ denotes the joint state of the random variables other than T and $\mathbf{H}$. To implement the edit, a LMSR market maker will update u's assets $a^u_{\mathbf{v}}$ according to Equation (2). It is convenient to work with an exponential transformation $q^u(\mathbf{v}) = (a^u_{\mathbf{v}})^b$ of u's assets.

Proposition 3.1. Let the transformation q^u on a user's assets be given by:

$$q^u(\mathbf{v}) = (a^u_{\mathbf{v}})^b \,, \tag{8}$$

If the user makes a structure preserving edit of $p(t|\mathbf{H}=\mathbf{h})$ to $x(t|\mathbf{H}=\mathbf{h})$, then the ratio of new to old transformed assets satisfies:

$$\frac{q'^u(\mathbf{v})}{q^u(\mathbf{v})} = \frac{(a'^u_{\mathbf{v}})^b}{(a^u_{\mathbf{v}})^b} = \frac{x(\mathbf{v})}{p(\mathbf{v})}, \tag{9}$$

where $(a'^u_{\mathbf{v}})$ is the user's updated assets in joint state $\mathbf{v}$ as defined in (2) and $q'^u(\mathbf{v})$ is the updated transformed assets.

Proof. The expression (9) can be derived as follows:

$$\begin{aligned} a'^u_{\mathbf{v}} &= a^u_{\mathbf{v}} + \Delta a^u_{\mathbf{v}} \\ &= a^u_{\mathbf{v}} + b \ln \frac{x(\mathbf{v})}{p(\mathbf{v})} \\ &= b \ln(q^u(\mathbf{v})) + b \ln \frac{x(\mathbf{v})}{p(\mathbf{v})} \\ &= b \ln(q'^u(\mathbf{v})) \end{aligned}$$

$$\begin{aligned} &\Rightarrow \quad b \ln(q'^u(\mathbf{v})) = b \ln(q^u(\mathbf{v})) + b \ln \frac{x(\mathbf{v})}{p(\mathbf{v})} \\ &\Rightarrow \quad \ln(q'^u(\mathbf{v})) - \ln q^u(\mathbf{v}) = \ln \frac{x(\mathbf{v})}{p(\mathbf{v})}. \\ &\Rightarrow \quad \frac{q'^u(\mathbf{v})}{q^u(\mathbf{v})} = \frac{x(\mathbf{v})}{p(\mathbf{v})}. \end{aligned}$$

This establishes (9). □

To interpret the constant b, note that if the user changes the probability of state $\mathbf{v}$ from $p(\mathbf{v})$ to $x(\mathbf{v}) = 1$ and $\mathbf{v}$ turns out to be true, the user gains $\Delta a^u_{\mathbf{v}} = -b \ln p(\mathbf{v})$. The maximum possible gain is therefore $-b \ln p(\mathbf{v}_*)$, where $\mathbf{v}_*$ is the minimum probability state. If all states start out equally likely, i.e., $p(\mathbf{v}) = 1/L$ for all $\mathbf{v}$, then the maximum gain to

users, and the maximum loss to the market maker, is $b \ln L$, where L is the total number of states. Therefore, to bound losses to be no more than M, we can initialize all market states to be equally likely at the start of trading and set $b = M/\ln L$.

If q^u starts out independent of the state and changes in proportion to changes in p, we can decompose q^u similar to the decomposition of p in Equation (6). Specifically,

$$q^u(\mathbf{v}) = \frac{\prod_{c \in \mathcal{C}} q_c^u(\mathbf{v}_c)}{\prod_{s \in \mathcal{S}} q_s^u(\mathbf{v}_s)}, \tag{10}$$

where q_c^u and q_s^u are local asset components defined on the cluster and separator variables, respectively. Notice the similarity between (10) and (6). This factored representation for assets is preserved as long as initial assets are state-independent and all edits are structure preserving.

Any structure preserving edit can be implemented by changing the potential function for just one cluster. All factors of (9) cancel except for the joint space of the edited cluster. That is, assets can be updated simply by choosing a cluster c containing the variables being traded and multiplying $q_c^u(\mathbf{v}_c)$ by the probability ratio:

$$q_c'^u(\mathbf{v}_c) = q_c(\mathbf{v}_c)\frac{x_c(\mathbf{v}_c)}{p_c(\mathbf{v}_c)}. \tag{11}$$

Combining 11 with Equation (10) yields the same result as Equation (9) in the case of a structure preserving edit.

Thus, a representation of the user's assets can be stored as an asset junction tree of the same structure as the junction tree representing the consensus distribution. Each edit can be processed locally by modifying a cluster containing the edited variables. The user's asset junction tree can then be used to compute cash and expected assets as described below.

3.5 Determining Allowable Edits

Allowable edits must respect the rule that the user's cash c^u may not become negative in any joint state with non-zero consensus probability. Equivalently, any edit that changes $x(t|\mathbf{H} = \mathbf{h})$ to $p(t|\mathbf{H} = \mathbf{h})$ must fall within the edit limits $x_* \leq x(t|\mathbf{H} = \mathbf{h}) \leq x^*$ as defined by (3). The factored asset representation (10) can be exploited to compute cash, conditional cash, and edit limits. Dawid's (1992) generalization of the junction tree algorithm can be applied to find the minimum value of a function that can be represented in factored form as in (10). The market maker can verify that the user's cash is non-negative by finding the minimum of the transformed assets q^u as defined in Equation (8) and checking that it is no smaller than 1. Dawid's algorithm can be easily adapted to calculate conditional minima, providing a way to calculate conditional cash $c_{\mathbf{y}}^u$ and consequently edit limits (8).

3.6 Expected Value of User's Assets

A user typically wants to know the expected value of her assets given the current market consensus prices. If she is contemplating an edit to event A, she would want to know what her expected assets will be if A happens and if $\neg A$ happens. These expectations can be calculated given the factorization represented in the junction tree, as shown by Dawid (1992).

User u's expected assets are given by:

$$\bar{a}^u = \sum_{c \in \mathcal{C}} \sum_{\mathbf{v}_c} a_c^u(\mathbf{v}_c) p_c(\mathbf{v}_c) - \sum_{s \in \mathcal{S}} \sum_{\mathbf{v}_s} a_s^u(\mathbf{v}_s) p_s(\mathbf{v}_s), \tag{12}$$

where

$$a_c^u(\mathbf{v}_c) = b \ln(q_c^u(\mathbf{v}_c)),$$

and

$$a_s^u(\mathbf{v}_s) = b \ln(q_s^u(\mathbf{v}_s)).$$

Note that the asset updating rule (11) updates only to the cluster tables, leaving the separator tables unchanged. Because all separator tables start out independent of state and are not changed by updating, the second term of (12) is a constant.

3.7 Resolving Variables

When the value of a random variable $W = w$ becomes known, the distributions of the remaining random variables $\mathbf{U} = \mathbf{V} \setminus W$ are updated to $p(\mathbf{U}|W = w)$ according to Bayes' rule. The junction tree algorithm is a computationally efficient way to perform Bayesian updating. After updating, we can remove W from the representation. To do this, the consensus probability distribution is updated by removing W from all clusters and separators to which it belongs, and then replacing all cluster and separator marginal distributions $p_c(\mathbf{v}_c)$ and $p_s(\mathbf{v}_s)$ with their conditional distributions given $W = w$. Each user's asset structure is then updated by removing W from all the asset clusters and separators, and replacing all the cluster and separator asset tables with the conditional tables given $W = w$.

3.8 Adding Variables

To add a new random variable Y, we first decide whether it should be linked to any of the other market random variables. If so, we need to add arcs connecting Y to these random variables. This may necessitate recompiling the junction tree. It is possible that recompilation will assign variables formerly in the same cluster to different clusters. In this case, any edits involving these variables, while structure preserving with respect to the original junction tree, would no longer be structure preserving in the new representation. This issue can be addressed by modifying the junction tree construction algorithm to force variables connected by edits to remain in the same cluster.

After updating the junction tree, each user's asset structure must be updated – an expensive operation that will be avoided by the algorithms in Section 4. The asset tables remain unchanged for any clusters left unchanged in the probability representation. If the only change to a cluster is to add the variable Y, the cluster table is updated by simply duplicating the existing table for each state y of Y. More extensive changes to the junction tree require re-processing edits. Recall that each edit is processed by applying (11) to a cluster c containing the edited variables. An edit must be reprocessed unless it is associated with an unchanged cluster or a cluster that changes only by adding Y.

3.9 Non Structure Preserving Edits

So far we have considered edits that follow equation (1). In such an edit, a user who changes $p(T|H = h)$ also changes her assets $a^u_{\mathbf{v}}$ in any joint state in which $H = h$, and leaves unchanged the probabilities $p(H = h)$ and $p(T|H \neq h)$, as well as her assets in any state for which $p(H \neq h)$. No one else's assets change in any way. These relationships hold for the general but computationally intractable LMSR. Because the algorithms presented in this paper employ LMSR, they also respect these relationships, but are limited to structure preserving edits.

What should a system do if a user requests to make an edit that violates the structural assumptions of the current graphical model? If we thought there was a strong consensus that the graphical model is correct, we might interpret this user request as wanting to leave the graphical model unchanged, and yet still make the requested change to $p(T|H = h)$. It would be interesting further research to seek a general method to satisfy the user's request as closely as possible while making a minimum change to $p(T|H \neq h)$, $p(H = h)$, and assets in states with $H \neq h$. However, any such algorithm (like minimizing KL divergence) could involve the user in any number of bets they might disavow, so we chose a more direct approach. Our response to an edit request that violates structural assumptions is to change the graphical model. The simplest way to do this is to extend the model by adding links between variables. Direct links to T from all the variables in H would be sufficient to support a change to $p(T|H = h)$, but not all of these links may be necessary. By using this extended model, one can preserve all the features of the algorithms in this paper.

However, adding links can increase computational cost. For example, doing this could increase the treewidth of the junction tree, and as we've seen the computational complexity of probability updating is exponential in that treewidth. In the final version of our SciCast system, when a user indicated that they wanted to edit a particular conditional probability that violated our structural assumptions, the system required an edit whose cost increased with the new computational load. Specifically, the larger the treewidth, the larger the minimum required edit.

If new links are sometimes added to the model, yet never removed (except by questions resolving), then eventually the treewidth will become large enough to prevent adding more links. Thus for long term adaptability, one should consider removing links. But cutting links raises two new problems: stranded assets and user incentives for cutting links.

Regarding stranded assets, consider the simplest case of two binary variables A and B, with one link between them. In this case there are three independent probabilities, four possible states, and users can have a different asset amount in each of the four states. If the one link between these two variables is cut, then there are now only two independent probabilities, and a parallel junction tree representation of assets now assumes that the added assets that a user has if the variable A has state true, relative to the state false, is independent of the value of the variable B. To convert assets from the previous asset representation into this new representation, some assets in the original representation will in general be "stranded", in that they are not directly expressed in the new representation. Stranded assets will still be paid out properly when variables are resolved, but cannot be reused.

As variables are resolved, stranded assets involving those variables become simpler, and eventually turn into cash. Currently stranded assets are not considered when calculating cash values for feasible edit limits, but this is exactly as it should be, and so edit limits are correctly calculated. It is straightforward to include stranded assets in expected value calculations. Thus the problem of stranded assets seems to be a manageable one.

There is a potential incentive problem, however, that seems more difficult. Consider that in a graphical model, the lack of a link between two variables corresponds to an exact conditional independence relationship. Therefore, in order to remove an arc, the probabilities must be adjusted to remove that particular conditional dependence. The problem is that profit-maximizing users have no obvious incentive to exactly balance the probability numbers. A user interface could include a macro operator that enables users to easily produce these exact values, and thus allow links to be cut from the graph. However, some other incentive will need to be found and offered to induce users to make such edits. Alternatively, the system can periodically remove weak links that induce compute costs, ideally by making regular edits and absorbing the associated costs. In SciCast we never needed to invoke arc deletion.

4. Trade-based Asset Management

In the PJT representation, each user has an asset junction tree with the same structure as the consensus belief junction tree. A typical user in a market with a large number of questions will make edits on only a small proportion of the questions. Because a user's assets do not depend on the state of any random variables she has not edited, it is highly inefficient to maintain a user-specific asset structure on the entire joint space. Further, any changes (e.g., adding questions, resolving questions, or adding links) to the structure of the global junction tree must be propagated to each user's asset structure. With large numbers (e.g., thousands) of users and questions, space and time complexity of the parallel junction tree representation can become prohibitive. Adequate performance for large-scale prediction markets requires a more efficient approach to asset management.

To address this challenge, trade-based asset management uses a data structure for each user's assets that involves only the random variables on which that user has traded. Each user's asset structure consists of a set of asset blocks constructed from the user's trade history and updated incrementally with each trade. When most users trade on only a small fraction of the market variables, the asset blocks consume much less space and support more efficient asset management algorithms than the parallel junction tree representation. The trade-based asset structure no longer mirrors the probability structure, which means that changes to the structure of the joint distribution no longer need to be propagated to all user-specific asset structures.

For example, consider a combinatorial prediction market in which the consensus distribution is represented by the junction tree of Figure 2. Consider a user who makes structure preserving edits involving only four variables, $V2$, $V6$, $V11$, and $V17$. To determine this user's allowable edits and expected assets, the market maker needs to consider only joint configurations of these four variables. The asset block representations described in this section would represent this user's asset structure with the junction tree of Figure 3. This is much smaller than the asset junction tree maintained under the parallel junction tree

Figure 3: Trade-Based Asset Structure for Sparse Trader.

representation. Further, in computing prices and expected assets for this user, we can remove all clusters from the consensus distribution junction tree except the three clusters at the top of Figure 2. The time and space complexity of computation with junction trees is the size of the largest cluster times the number of clusters. Across a large number of users who make sparse edits, we can achieve a large savings in both storage and computation by moving to a user-specific asset block representation.

Section 4.1 describes the basic asset representation. Section 4.2 describes how the market maker tasks described in Section 2 are accomplished with the asset block representation. Section 4.3 provides algorithms for updating the asset block representation as trades and resolutions occur. Sections 4.4 and 4.5 describe two different methods for calculating cash with the asset block representation, a simple block merge approach and a method based on an asset junction tree constructed dynamically from a user's current asset structure.

4.1 Representing Gains and Losses with Asset Blocks

The basic idea of our trade-based asset representation is to partition the user's trades into subsets, such that each subset involves only a small number of variables. Each subset of trades is summarized by a data structure called an *asset block*, which represents a user's gains and losses from its associated trades.

Definition 4.1. An *asset block* $B = (\mathbf{V}_B, \delta_B)$ consists of a vector $\mathbf{V}_B$ of block variables and a block asset function δ_B that maps states $\mathbf{v}_B$ of $\mathbf{V}_B$ to real numbers $\delta_B(\mathbf{v}_B)$.

The block variables $\mathbf{V}_B$ are those involved in the trades covered by the asset block. The value δ_B of the block asset function at the state $\mathbf{v}_B$ is the net amount the user gains or loses from the covered trades if the state occurs.

A single asset block B summarizes gains or losses from a subset of trades. A collection of asset blocks covering all the user's trades forms a compact representation of gains or losses in any joint state.

Definition 4.2. A user's assets $a^u_{\mathbf{v}}$ are *additively decomposable* with respect to the set $\mathcal{B}$ of asset blocks if

$$a^u_{\mathbf{v}} = \sum_{B \in \mathcal{B}} \delta_B(\mathbf{v}_B). \tag{13}$$

For each user u, the market maker maintains a user-specific collection $\mathcal{B}^u$ of asset blocks such that u's assets are additively decomposable with respect to $\mathcal{B}^u$. These asset blocks $\mathcal{B}^u$ are updated incrementally with each trade and each new question resolution to ensure that (13) holds.

To support space and time efficient asset management, we require that the user's asset blocks satisfy the following properties.

Definition 4.3. The asset block B is *structurally congruent* with the junction tree (6) if there exists a cluster $c \in \mathcal{C}$ such that every variable in $\mathbf{V}_B$ is an element of c.

Definition 4.4. The set $\mathcal{B}$ of asset blocks is *frugal* if for any pair of blocks $B \neq B' \in \mathcal{B}$, the block variable vector $\mathbf{V}_B$ includes at least one variable not in $\mathbf{V}_{B'}$.

Structural congruence allows computing the block variable marginal distribution $p(\mathbf{V}_B)$ easily from the cluster marginal distribution $p(\mathbf{V}_c)$, which is maintained by the market maker. The block marginal distribution is needed to compute expected gains and losses $\sum_{\mathbf{v}_B} \delta_B(\mathbf{v}_B)p(\mathbf{v}_B)$. Frugality is desirable to avoid unnecessary proliferation of blocks. Together, these two requirements allow the asset management computations to be performed in a natural and computationally efficient way using the consensus probability distribution.

4.2 Market Maker Tasks with Asset Block Representation

For each user u, the trade-based asset manager maintains a collection $\mathcal{B}^u$ of asset blocks that represent u's trading history. This section describes how the asset block representation can be used to perform the market maker tasks defined in Section 2. The representation supports efficient computation of edit limits and expected assets in the typical case where a user's trades are sparse in the full joint space. The asset structure $\mathcal{B}^u$ is updated each time u makes an edit to reflect gains and losses from the edit. When a question W resolves to state w, every user's asset structure is updated to remove W from $\mathcal{B}^u$ and update all the block asset functions to reflect the resolution. Because the asset block representation includes only the random variables on which the user has traded, resource savings can be large when trades are sparse.

Each user's asset structure is initialized to a null asset structure representing initial assets a_0^u that are constant across all states. As shown in Section 4.3 below, the properties of frugality, structural congruence and additive decomposability are trivially satisfied by the initial asset structure and are preserved by the asset updating algorithms.

4.2.1 Probability Queries

Probability queries are performed exactly as described in Section 3.2 and are unaffected by changes to the asset representation.

4.2.2 Structure Preserving Edits

An edit by user u requires both an update to the consensus joint distribution and a change to u's asset structure. The probability update is accomplished via soft evidence update to the consensus probability distribution as described in Section 3.3 and is unaffected by the asset representation. The asset structure $\mathcal{B}^u$ is updated by either creating a new asset block or updating an existing asset block. This must be done in a way that preserves additive decomposability, frugality and structural congruence. Details are given in Section 4.3 below.

4.2.3 Determining Allowable Edits

In addition to the requirement that edits must be structure preserving, no edit is allowed if the user's assets can become negative in any state. In order to ensure that no edit will allow the user's assets to become negative, the market maker must be able to compute conditional and unconditional minimum assets c^u and $c^u_{\mathbf{y}}$. Section 4.4 below presents a simple cash calculation method that works well when asset blocks have few overlaps. Section 4.5 presents

a more general method that constructs a junction tree from the user's asset blocks and uses min propagation to find the user's cash.

4.2.4 Expected Value of User Assets

The expected assets of user u are given by:

$$\bar{a}^u = \sum_{B \in \mathcal{B}} \sum_{\mathbf{v}_B} \delta_B(\mathbf{v}_B) p_B(\mathbf{v}_B), \tag{14}$$

Structural congruence implies that each block variable vector $\mathbf{V}_B$ is contained in at least one of the cluster variable vectors $\mathbf{V}_c$ of the consensus distribution junction tree. Thus, the block marginal distribution $p_B(\mathbf{V}_B)$ can be computed by marginalizing the cluster marginal distribution $p_c(\mathbf{V}_c)$. Therefore, it is straightforward to calculate the sum (14) from the marginal distributions and block asset functions.

4.2.5 Resolving Variables

When a variable W is resolved to one of its values w, the consensus probability junction tree is updated as described in Section 3.7 and is unaffected by the asset representation. If user u has made no edits involving the variable W, then u's asset structure does not change. Otherwise, u's asset structure $\mathcal{B}^u$ must be updated as follows. Find all blocks $B \in \mathcal{B}$ such that W is one of the variables in $\mathbf{V}_B$. For each such B, we write the block variables as $\mathbf{V}_B = (W, \mathbf{Y})$, where $\mathbf{Y}$ consists of the block variables other than W. The block B is replaced by a new block B^* with block variables $\mathbf{V}_{B^*} = \mathbf{Y}$ and block asset function $\delta_{B^*}(\mathbf{y}) = \delta_B(w, \mathbf{y})$. This process of removing W may result in a new asset structure that is no longer frugal. That is, $\mathbf{V}_{B^*}$ may be entirely contained in the block variables of another block. If so, then B^* is then merged with one such block.

It is straightforward, as shown in Section 4.3 below, to show that this resolution method preserves frugality, structural congruence and additive decomposability.

4.2.6 Adding Variables

When a new random variable Y is added, the probability structure is updated as described in Section 3.8. If all asset blocks are structurally congruent with the new probability structure, no change is needed to the asset structure. Otherwise, any previous edits associated with non structurally congruent blocks need to be re-processed.

4.3 Asset Block Updating

This section provides details on how asset blocks are updated when a user makes edits. We assume each user u begins with state-independent assets represented as an asset structure $\mathcal{B}^u = \{B_\emptyset\}$ containing a single "null" block $B_\emptyset$ for which the block variable vector $\mathbf{V}_{B_\emptyset} = ()$ is empty and the block asset function $\delta_{B_\emptyset} = a_0^u$ is a constant equal to u's initial assets. It is clear from Definitions 4.2, 4.3 and 4.4 that the initial asset structure $\{B_\emptyset\}$ is frugal and structurally congruent with the junction tree (5), and that u's initial assets are additively decomposable with respect to $\{B_\emptyset\}$.

The asset structure $\mathcal{B}^u$ is updated after each of u's edits and after every resolution of any variable u has edited. In this section, we show that the properties of frugality, structural congruence and additive decomposability are preserved when assets are updated.

At a certain point during trading, the user u has asset structure $\mathcal{B}^u = \{B_1^u, \ldots, B_k^u\}$ consisting of k blocks. We assume $\mathcal{B}^u$ is frugal and structurally congruent with the junction tree (5), and that u's current assets are additively decomposable with respect to $\mathcal{B}^u$. When u makes an edit involving variables $\mathbf{Y}$, u's pre-edit asset structure $\mathcal{B}^u$ is updated as follows.

1. Create a new asset block $B_{k+1}^u = (\mathbf{V}_{B_{k+1}^u}, \delta_{B_{k+1}^u})$, with block variables $\mathbf{V}_{B_{k+1}^u} = \mathbf{Y}$ and block asset function $\delta_{B_{k+1}^u}(\mathbf{y}) = b \ln(\frac{x(\mathbf{y})}{p(\mathbf{y})})$, where $p(\mathbf{y})$ and $x(\mathbf{y})$ are the pre- and post-edit probability distributions on $\mathbf{Y}$. The new block asset function $\delta_{B_{k+1}^u}(\mathbf{y})$ represents gains and losses from the edit.

2. Perform zero or more block merges. To merge two blocks B_i and B_j into a single block B, set block variables $\mathbf{V}_B$ to include all variables occurring in either of $\mathbf{V}_{B_i}$ or $\mathbf{V}_{B_j}$, and block asset function $\delta_B(\mathbf{v}_B) = \delta_B(\mathbf{v}_{B_i}) + \delta_B(\mathbf{v}_{B_j})$ equal to the sum of the original block asset functions applied to their respective sub-vectors of $\mathbf{v}_B$. Any block whose variables are all included in those of another block must be merged into one such block. No merge is allowed if the resulting block would violate structural congruence. Otherwise, merges are at the discretion of the asset management algorithm.

Algorithm 4.1 summarizes the process of updating a user's asset blocks after a trade.

The following two propositions are immediate consequences of Definition 4.2, and together imply that edit updates preserve additive decomposability of the editing user's assets.

Proposition 4.5. Suppose u's assets $a_{\mathbf{v}}^u$ are additively decomposable with respect to the asset structure $\mathcal{B} = \{B_1, \ldots, B_k\}$ and let B_{k+1} be an asset block with block variables $\mathbf{V}_{B_{k+1}} = \mathbf{Y}$ and block asset function $\delta_{B_{k+1}}(\mathbf{y}) = b \ln(\frac{x(\mathbf{y})}{p(\mathbf{y})})$, where all variables $\mathbf{Y}$ are contained in $\mathbf{V}$. Let

$$a_{\mathbf{v}}^* = a_{\mathbf{v}}^u + b \ln \left(\frac{x(\mathbf{y})}{p(\mathbf{y})} \right).$$

Then $a_{\mathbf{v}}^*$ is additively decomposable with respect to $\mathcal{B}^* = \{B_1, \ldots, B_{k+1}\}$.

Proposition 4.6. Suppose u's assets $a_{\mathbf{v}}^u$ are additively decomposable with respect to the asset structure $\mathcal{B} = \{B_1, \ldots, B_k\}$, where $k \geq 2$. Let B be an asset block with block variables $\mathbf{V}_B$ consisting of all variables in either of $\mathbf{V}_{B_1}$ or $\mathbf{V}_{B_2}$, and block asset function $\delta_B(\mathbf{v}_B) = \delta_B(\mathbf{v}_{B_1}) + \delta_B(\mathbf{v}_{B_2})$. Let $\mathcal{B}^* = \{B, B_3, \ldots, B_k\}$ be a new asset structure replacing B_1 and B_2 with the merged block. Then u's assets $a_{\mathbf{v}}^u$ are additively decomposable with respect to $\mathcal{B}^*$.

The next proposition is also an immediate consequence of Definition 4.2. Ir implies that resolution preserves additive decomposability of assets.

Proposition 4.7. Suppose u's assets $a_{\mathbf{v}}^u$ are additively decomposable with respect to the asset structure $\mathcal{B} = \{B_1, \ldots, B_k\}$. Let W be one of the variables in $\mathbf{V}$, and let w be one of the possible states of W. Let $\mathcal{B}^* = \{B_1^*, \ldots, B_k^*\}$ be a new asset structure obtained as follows.

- If W is not one of the variables in $\mathbf{V_{B_i}}$, then $B_i^* = B_i$.
- Otherwise, writing $\mathbf{V_{B_i}} = (W, \mathbf{Y})$, the updated block B_i^* has block variables $\mathbf{V}_{B^*} = \mathbf{Y}$ and block asset function $\delta_{B^*}(\mathbf{y}) = \delta_B(w, \mathbf{y})$.

Then u's assets $a^u_{\mathbf{v}}$ are additively decomposable with respect to the asset structure $\mathcal{B}^*$.

Algorithm 4.1 (Update User's Asset Blocks after a Trade)**.** This algorithm is called when a user makes a trade, to update the user's asset model by either creating a new asset block or absorbing the trade into existing blocks.

Require: user's asset structure $\mathcal{B} = \{(\mathbf{V}_i, \delta_i)\}$. Here, $\mathbf{V}_i$ is the vector of domain variables in the i^{th} block, δ_i is the asset function of the i^{th} block, represented as a table of dimension equal to the product of the number of states of the variables in $\mathbf{V}_i$.

Require: trade information: target question T; set of assumption questions $\mathbf{A}$, and their given states $\mathbf{a}$; current conditional probability $p(T|\mathbf{A} = \mathbf{a})$; intended target probability $x(T|\mathbf{A} = \mathbf{a})$

Require: domain sizes of variables involved in the trade, namely, an array of number of states of $T, \mathbf{A}$. This is used to calculate the size of the new asset table.

Require: list $\mathcal{C}$ of clusters of variables, with which it is assumed $\mathcal{B}$ is structurally congruent, and must remain so at termination.

1. create the asset table δ^* associated with the trade, using market scoring rule $\delta^* = b * \log([x(T|\mathbf{A} = \mathbf{a})]/[p(T|\mathbf{A} = \mathbf{a})])$
2. $flag = 0, k =$ number of asset blocks in $\mathcal{B}$
3. for $i = 1$ to k, do
   $\mathbf{D} = \mathbf{V}_i$; $\tau = \delta_i$;
   if $\mathbf{D} == \{T, \mathbf{A}\}$
      $\delta_i = \delta^* + \tau$;
      $flag = 1$; break
   if $\mathbf{D} \supset \{T, \mathbf{A}\}$
      Extend δ^* to the size of $\mathbf{D}$
      $\delta_i = \delta^* + \tau$; $flag = 1$; break
   if $\mathbf{D} \subset \{T, \mathbf{A}\}$
      Extend τ to the size of $\{T, \mathbf{A}\}$
      $\mathbf{V}_i = \{T, \mathbf{A}\}$; $\delta_i = \tau + \delta^*$; $flag = 1$; break
   if $\mathbf{D} \cup \{T, \mathbf{A}\} \subset c$ for some cluster $c \in \mathcal{C}$ then optionally
      Extend τ and δ^* to the size of $\{T, \mathbf{A}\} \cup \mathbf{D}$
      $\mathbf{V}_i = \{T, \mathbf{A}\} \cup \mathbf{D}$; $\delta_i = \tau + \delta^*$; $flag = 1$; break
   endfor
4. if $flag == 0, \mathbf{V}_{k+1} = \{T, \mathbf{A}\}, \delta_{k+1} = \delta^*$.
5. return $\mathcal{B} = \{(\mathbf{V}_i, \delta_i)\}$

Together, Propositions 4.5, 4.6, and 4.7 imply the following result:

Proposition 4.8. Suppose u begins with initial asset structure $\mathcal{B}^u_0 = \{B_\emptyset\}$ consisting of a single asset block having zero-length block variable vector $\mathbf{V}_{B_\emptyset} = ()$ and constant block asset function $\delta_{B_\emptyset} = a^u_0$ with value equal to u's initial assets. Suppose u makes a sequence of

structure preserving edits to the joint distribution (6). Interspersed among u's edits may be structure preserving edits by other users and resolutions of some variables. Let $\mathcal{B}^u$ denote u's asset structure after all edits and resolutions have been processed as described above. Then:

1. All blocks $B \in \mathcal{B}^u$ are structurally congruent with (6);
2. $\mathcal{B}^u$ is frugal; and
3. u's assets $a^u_{\mathbf{v}}$ are additively decomposable with respect to $\mathcal{B}^u$.

Proof. The initial asset structure $\mathcal{B}^u_0 = \{B_0\}$ trivially satisfies structural congruence, frugality and additive decomposability of a^u_0. As an induction hypothesis, suppose u's asset structure $\mathcal{B}^u_{t-1}$ just prior to the t^{th} edit satisfies structural congruence, frugality and additive decomposability. The first step in processing the t^{th} edit is adding a new block to represent gains and losses from the edit. Proposition 4.5 implies that this step preserves additive decomposability of u's asset structure. If the edit is structure preserving, the new block is structurally congruent with (6). Proposition 4.6 implies that any block merges that occur will not destroy additive decomposability. If updating operations result in a non-frugal asset structure, block merges will be performed to restore frugality. Therefore, edits preserve the properties of structural congruence, frugality and additive decomposability of u's asset structure. Any edits by other users between u's t^{th} and $(t+1)^{st}$ edits do not affect u's asset structure. Further, Proposition 4.7 implies that any resolutions between edits t and $t+1$ preserve additive decomposability. By Proposition 4.6, any post-resolution merges to restore frugality also preserve additive deomposability. Therefore, the asset structure $\mathcal{B}^u_t$ satisfies structural congruence, frugality and additive decomposability, and the result follows by induction. □

4.4 Global Separator (GS) Method for Cash Calculation

In the PJT approach, the user's conditional or unconditional minimum assets are calculated using min-propagation in the user's asset junction tree (Section 3.5). Having replaced the parallel junction tree with a user-specific asset structure, we need a method to find minimum assets using this data structure. This section describes a simple method, called the *global separator* algorithm, that works well when the asset blocks do not have many variables in common.

The collection of a user's asset blocks is a compact representation of the user's trading history. Each asset block represents a modular set of trades involving only that block's variables. The associated asset functions record local asset changes and collectively represent the user's assets in any joint state. We would like to exploit the modularity of the asset block representation to do asset management with local computations involving only a few variables at a time.

From an asset structure $\mathcal{B} = \{B_1, \ldots, B_k\}$, we can construct an undirected graph in which nodes correspond to trade blocks, an arc connects any pair of blocks that share a variable, and each arc is labeled with variables shared between its terminal nodes. An example is shown in Figure 4, where u's gains and losses from trades involving variables A through H are represented in an asset structure consisting of five asset blocks. The variables

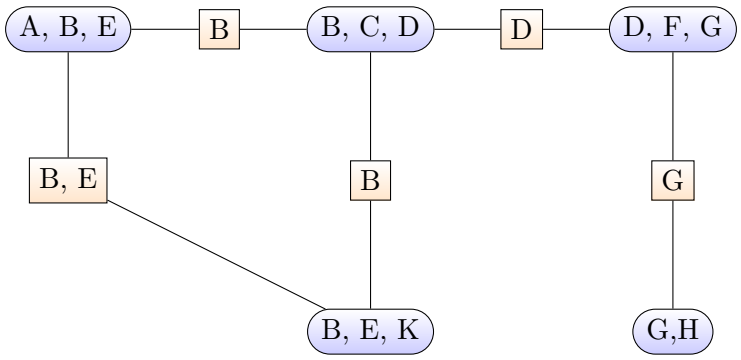

Figure 4: Example asset block graph.

B, D, E, and G are shared between blocks, while each of A, C, F, and H belongs to only one block.

As shown in Figure 5, we can transform the graph of Figure 4 into a junction tree with the shared variables as the root node and the five original trade blocks as leaf nodes. The variables at the root, which all appear in more than one block, are called the *global separator*, because they separate all the leaf nodes from each other.

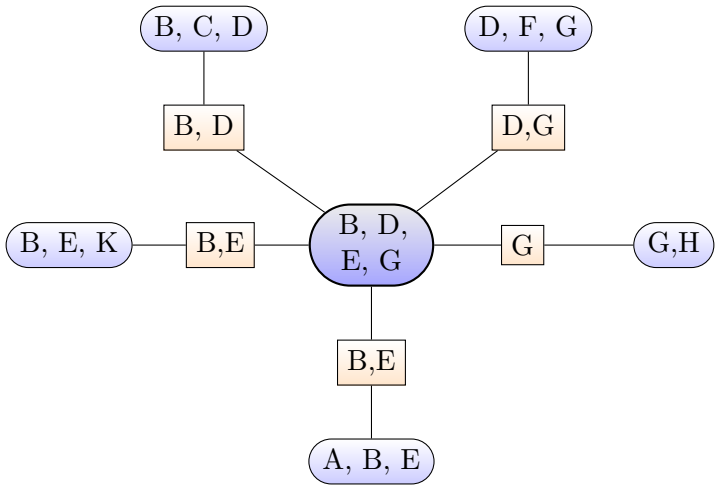

Figure 5: Global Separator Tree for Figure 4 Asset Blocks.

Definition 4.9. The *global separator (GS)* $\mathbf{V}_{\mathcal{B}}$ for asset structure $\mathcal{B} = \{B_1, \ldots, B_k\}$ consists of all variables contained in more than one of the $\mathbf{V}_{B_i}$.

The GS algorithm iterates through states $\mathbf{v}_{\mathcal{B}}$ of the global separator $\mathbf{V}_{\mathcal{B}}$, summing the minima of the asset blocks for states in which $\mathbf{V}_{\mathcal{B}} = \mathbf{v}_{\mathcal{B}}$. The minimum over $\mathbf{v}_{\mathcal{B}}$ of these sums is the cash c^u.

Proposition 4.10. Let $a^u_{\mathbf{v}}$ be additively decomposable with respect to asset structure $\mathcal{B} = \{B_1, \ldots, B_k\}$, and let $\mathbf{V}_{\mathcal{B}}$ be the global separator for $\mathcal{B}$. Then u's cash $c^u = \min_{\mathbf{v}} a^u_{\mathbf{v}}$ is

given by:

$$c^u = \min_{\mathbf{v}_\mathcal{B}} \sum_{i=1}^{k} \min_{\{\mathbf{v}_{B_i} : \mathbf{V}_\mathcal{B} = \mathbf{v}_\mathcal{B}\}} \delta_{B_i}(\mathbf{v}_{B_i}). \tag{15}$$

Proof. From additive decomposability of u's assets with respect to $\mathcal{B} = \{B_1, \ldots, B_k\}$ we have

$$\begin{aligned} c^u &= \min_{\mathbf{v}} a^u_{\mathbf{v}} \\ &= \min_{\mathbf{v}} \sum_{i=1}^{k} \delta_{B_i}(\mathbf{v}_{B_i}) \\ &= \min_{\mathbf{v}_\mathcal{B}} \min_{\{\mathbf{v} : \mathbf{V}_\mathcal{B} = \mathbf{v}_\mathcal{B}\}} \sum_{i=1}^{k} \delta_{B_i}(\mathbf{v}_{B_i}). \end{aligned}$$

Conditional on $\mathbf{v} : \mathbf{V}_\mathcal{B} = \mathbf{v}_\mathcal{B}$, the conditional minima of the asset blocks are independent of each other. Thus, the inner minimum can be brought inside the sum:

$$\begin{aligned} c^u &= \min_{\mathbf{v}_\mathcal{B}} \sum_{i=1}^{k} \min_{\{\mathbf{v} : \mathbf{V}_\mathcal{B} = \mathbf{v}_\mathcal{B}\}} \delta_{B_i}(\mathbf{v}_{B_i}) \\ &= \min_{\mathbf{v}_\mathcal{B}} \sum_{i=1}^{k} \min_{\{\mathbf{v}_{B_i} : \mathbf{V}_\mathcal{B} = \mathbf{v}_\mathcal{B}\}} \delta_{B_i}(\mathbf{v}_{B_i}), \end{aligned}$$

which establishes (15). □

For conditional minimum assets, u's cash conditional on state $\mathbf{y}$ of variable $\mathbf{Y}$ is:

$$c^u_{\mathbf{y}} = \min_{\{\mathbf{v}_\mathcal{B} : \mathbf{Y} = \mathbf{y}\}} \sum_{i=1}^{k} \min_{\{\mathbf{v}_{B_i} : \mathbf{V}_\mathcal{B} = \mathbf{v}_\mathcal{B}\}} \delta_{B_i}(\mathbf{v}_{B_i}). \tag{16}$$

Algorithm 4.2 summarizes how to compute a user's unconditional minimum assets. The modification to calculate conditional minimum assets is straightforward.

Algorithm 4.2 (Calculate Cash using Global Separator)**.** This algorithm is called to find the minimum assets in any state.

Require: user's asset structure $\mathcal{B} = \{(\mathbf{V}_i, \delta_i)\}$. Here, $\mathbf{V}_i$ is the vector of domain variables in the i^{th} block, δ_i is the asset function of the i^{th} block, represented as a table of dimension equal to the product of the number of states of the variables in $\mathbf{V}_i$.

1. initialize global separator $\mathbf{G} = \emptyset$; other traded variables $\mathbf{W} = \emptyset$; k=number of asset blocks in $\mathcal{B}$
2. for $i = 1$ to k do
   for each variable $V \in \mathbf{V}_i$
   if $V \in \eta$ then set $\mathbf{W} = \mathbf{W} \setminus \{V\}$ and $\mathbf{G} = \mathbf{G} \cup \{V\}$
   else if $V \notin \mathbf{G}$ then set $\mathbf{W} = \mathbf{W} \cup \{V\}$
   Set global separator $\mathbf{G}$ to variables in γ
3. initialize $m_* = 0$
4. for all states $\mathbf{g}$ of the variables in $\mathbf{G}$
   for $i = 1$ to k do
   $m_* = m_* + \min_{\mathbf{G}=\mathbf{g}} \delta_i(\mathbf{v}_i)$
   endfor
5. return m_*

A common trading pattern is to make edits to multiple disjoint clusters of variables. This would happen, for example, if u expresses knowledge about multiple topics that are unrelated to each other. In this case, we can represent u's asset structure as $\mathcal{B} = \mathcal{B}_1 \cup \cdots \mathcal{B}_r$, where there is no overlap among the variables involved in different $\mathcal{B}_j$. Graphically, this corresponds to multiple disjoint global separator trees. When this happens, u's unconditional [conditional] assets are obtained by applying (15) [(16)] to each of the $\mathcal{B}_j$ separately and summing the results.

GS becomes intractable as the number of overlapping variables in a user's asset blocks becomes large. The SciCast market (Twardy et al., 2014) addressed this issue with a conservative approximation. When the algorithm finds an asset block containing variables that increase the size of the global separator beyond a threshold, it simply ignores the overlap, treating the new trade as a separate investment for the purpose of cash computation. The approximation still guarantees that the user's cash will not become negative, but may fail to exploit some opportunities for asset reuse. The next section describes an alternative method that improves upon GS when there are more than a few variables in the global separator.

4.5 Dynamic Asset Cluster Model

The simple approach of iterating over states of the global separator variables breaks down as a user makes more complex patterns of interrelated trades. For example, if a user makes a linked set of conditional trades on binary variables V_{i+1} given V_i, for $i = 1, \cdots, n$, the global separator will have size 2^n. Clearly, GS becomes intractable as n grows large. For example, a user who makes a series of 20 such linked edits on binary variables will have a

Algorithm 4.3 (Construct DAC Asset Junction Tree)**.** This algorithm is called to build *DAC* asset junction tree for computing asset related values. The built junction tree can be cached as long as the user does not make any new trades.

Require: user's asset structure $\mathcal{B} = \{(\mathbf{V}_i, \delta_i)\}$. Here, $\mathbf{V}_i$ is the vector of domain variables in the i^{th} block, δ_i is the asset function of the i^{th} block, represented as a table of dimension equal to the product of the number of states of the variables in $\mathbf{V}_i$.

1. if $\mathcal{B}$ is empty, return $NULL$;
2. set $k =$ number of asset blocks in $\mathcal{B}$;
3. initialize list of clusters $\mathcal{C} = NULL$; list of asset tables $\mathcal{D} = NULL$, cluster adjacency matrix $\mathbf{J} = \mathbf{0}$;
4. if $k = 1$ then set $\mathcal{C} = (\mathbf{V}_1)$; return $\mathcal{C}, \mathcal{D}$, and $\mathbf{J}$;
5. if $k = 2$, then set $\mathcal{C} = (\mathbf{V}_1, \mathbf{V}_2)$; $\mathcal{D} = (\delta_1, \delta_2)$; $\mathbf{J}(1, 2) = 1$; return $\mathcal{C}$, $\mathcal{D}$, and $\mathbf{J}$;
6. set $n =$ number of variables on which user has traded;
7. initialize $n \times n$ trade adjacency matrix $\mathbf{H} = \mathbf{0}$;
8. for $i = 1$ to k, do
   $\mathbf{d} =$ domain indices for variables in $\mathbf{V}_i$,
   $\mathbf{H}(\mathbf{d}, \mathbf{d}) = 1$;
   endfor
9. set the diagonal of $\mathbf{H}$ to be zeros, indicating no self-connections.
10. add edges to $\mathbf{H}$ to form a triangulated graph using a standard triangulation algorithm;
11. identify cliques of the graph from the triangulated adjacency matrix $\mathbf{H}$ and set list $\mathcal{C}$ of clusters to the set of cliques;
12. use a standard algorithm to find a junction tree for the clusters $\mathcal{C}$; set $\mathbf{J}$ to the adjacency matrix $\mathbf{H}$ for the junction tree;
13. set $z =$ number of clusters in $\mathcal{C}$;
14. initialize cluster asset tables $\varphi(j), j = 1, \ldots, z$ to zero ;
15. for $i = 1$ to k, do
    for $j = 1$ to z, do
    set $\mathbf{C}$ to the cluster variables for the j^{th} cluster ;
    if $\mathbf{V_i} \subseteq \mathbf{C}$, then
    merge asset block table δ_i into cluster asset table $\varphi(j)$;
    break ; (one block can go into only one cluster)
    endif
    endfor
    endfor
16. return $\mathcal{C}$, $\mathcal{D} = (\varphi(1), \ldots, \varphi(z))$, and $\mathbf{J}$;

global separator table of size greater than 1 million entries. A more robust approach to calculating minimum assets is needed for such cases.

To develop such an approach, we first note that the global separator tree of Figure 5 is a junction tree, and that the GS algorithm is a special case of min-propagation in a junction tree. It is well known that the complexity of the junction tree algorithm is dominated by the treewidth, or size of the largest cluster. The junction tree of Figure 6 is a smaller treewidth junction tree for the same asset block graph of Figure 4. This alternative junction tree can be used to find the user's cash more efficiently.

The Dynamic Asset Cluster (DAC) algorithm is based on this insight. DAC dynamically creates an asset junction tree from the trade block graph whenever a user makes a query for conditional or unconditional cash. Junction tree construction must ensure that the original asset blocks are never split when new cliques are formed. Asset tables in the junction tree are built from the asset tables of the original asset blocks. Min-propagation in this dynamically constructed junction tree gives the user's cash. The complexity difference is small for the example of Figures 5 and 4, but the experimental comparison of Section 5 demonstrates that the resource savings of DAC over the global separator method can be substantial.

The steps of the DAC algorithm are:

1. Create an undirected trade graph $\mathcal{G}$ by pairwise connecting all variables in each asset block. (This guarantees that all variables in each asset block will be contained in at least one clique in the asset junction tree.)

2. Triangulate $\mathcal{G}$ to make a triangulated graph $\mathcal{T}$, and identify all cliques from $\mathcal{T}$. An undirected graph is *triangulated* if every cycle with four or more nodes has a chord. A *clique* of a graph is a maximal set of nodes, all of which are connected to each other. Triangulation is performed because the cliques of a triangulated graph can always be arranged into a junction tree (Lauritzen & Spiegelhalter, 1988).

3. Use a standard algorithm to form a junction tree $\mathcal{J}$ from the triangulated graph $\mathcal{T}$.

4. Assign the asset function for each asset block to exactly one cluster in the junction tree that contains all the block variables for the asset block. The junction tree formation process ensures that none of the original blocks is split, thus guaranteeing that at least one clique will contain all the block variables.

5. Create an asset table for each clique by adding the block asset functions for blocks assigned to the clique.[2]

Algorithm 4.3 gives pseudo-code for constructing the asset junction tree from a user's asset blocks and filling in the clique asset tables.

2. Note that we do not fill in asset tables for the separators of the junction tree

Algorithm 4.4 (Min-propagation Protocol between two cliques)**.** Assume that two cliques $\mathcal{C}_i$ and $\mathcal{C}_j$ are neighbors in the junction tree, and the separator $\mathcal{S}_{ij}$ is associated with the edge between $\mathcal{C}_i$ and $\mathcal{C}_j$. The asset tables for $\mathcal{C}_i, \mathcal{C}_j$ and $\mathcal{S}_{ij}$ are $\varphi(\mathcal{C}_i), \varphi(\mathcal{C}_j)$ and $\varphi(\mathcal{S}_{ij})$ respectively. Min-propagation from $\mathcal{C}_i$ to $\mathcal{C}_j$ along the separator $\mathcal{S}_{ij}$ follows the min-propagation protocol presented below.

1. Let $\varphi(\mathcal{S}_{ij})' = \min_{\mathcal{C}_i \setminus \mathcal{S}_{ij}} \varphi(\mathcal{C}_i)$, — minimizing $\varphi(\mathcal{C}_i)$ onto the domain of the separator $\mathcal{S}_{ij}$.
2. Let $\mathcal{L}(\mathcal{S}_{ij}) = \varphi(\mathcal{S}_{ij})' - \varphi(\mathcal{S}_{ij})$, — subtracting the old asset in the separator $\mathcal{S}_{ij}$ with the projected minimization value from its neighbor. The result is called the separator gain.
3. Let $\varphi(\mathcal{C}_j) = \varphi(\mathcal{C}_j) + \mathcal{L}(\mathcal{S}_{ij})$, — summing up with the separator gain to update the assets $\varphi(\mathcal{C}_j)$.

Given the asset junction tree, local propagation can be used to perform the following tasks:

- *Calculate conditional minimum assets.* The min-propagation algorithm (Dawid, 1992) returns both the minimum global value and the joint state where the minimum occurs. To calculate trade limits, we need only the minimum global value and not the joint state where the minimum is attained. For this, one-way min-propagation is sufficient. We can choose any clique as the root, and a one-way propagation order from all leaf cliques to the chosen root can be determined. The propagation protocol between cliques is presented in Algorithm 4.4.

- *Calculate expected assets.* Expected assets are calculated by finding the joint consensus probability for each clique, calculating clique expected assets, and summing the expected values for all the cliques.

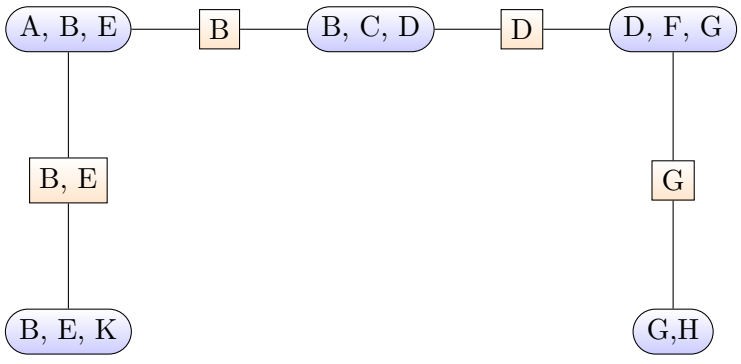

Figure 6: Junction Tree Constructed from Figure 4 Asset Blocks.

5. Evaluation

This section analyzes resource usage by the different asset management algorithms and reports on experiments comparing their performance.

5.1 Resource Usage

The introduction of graphical models enables tractable probability and asset management for combinatorial prediction markets with hundreds to thousands of base variables. All the algorithms presented in this paper are based on junction trees. This section considers time and space complexity of operations needed for probability and asset management. The following operations must be supported

1. *Construct consensus distribution junction tree.* This operation needs to be performed any time the structure of the junction tree changes, and is typically handled offline. Incremental modification can save resources over complete recompilation, as well as ensuring that all existing trades remain structure preserving. However, incremental modification can result in a junction tree with larger treewidth than compiling from scratch.

2. *Update consensus distribution after edit.* This operation requires probability propagation in the consensus distribution junction tree. It is performed after any edit is made. On restarting the system, all edits must be processed. For efficiency, as long as the structure does not change, all evidence can be posted first, and then probability propagation need only be performed once.

3. *Process assets for an edit.* This operation includes determining whether an edit is allowable, creating a new asset block to represent gains and losses from the edit, and integrating the new asset block into the existing representation. This operation is needed only for the user making the edit, and involves that user's asset structure. For DAC it includes constructing the user's asset junction tree; for GS it includes updating the GS tree; for PJT it involves manipulating the common junction tree.

4. *Calculate expected assets.* This operation involves both the consensus probability junction tree and the asset junction tree. It is performed every time expected assets are updated. Ideally, it would be performed after each edit for all users, but performance constraints may make this infeasible when there are many users.

Each of the above operations requires representing and manipulating tables in one or both of the probability and asset junction trees. Relevant parameters for time and space complexity are given in Table 2. Table 3 shows space and time complexity values for the PJT and trade-based methods.

When there are many users, the space complexity is dominated by the complexity of the asset junction tree, which is the same as the probability junction tree for the PJT method. If there are many sparsely trading users, so that most users' asset junction trees have smaller treewidth and fewer cliques than the probability junction tree, space savings for trade-based asset management can be large.

Table 2: Notation for Complexity Analysis

Parameter	**Definition**
u	# users
s	# states per variable
c	# cliques in probability junction tree
w	treewidth of probability junction tree
a	# cliques in asset junction tree
b	# asset blocks
z	treewidth of asset junction tree

The time for a probability update is independent of the asset management method. Cash computation is typically faster for the trade-based approach because min-propagation is faster than probability propagation. However, the time to perform cash updates is small relative to the time for probability updates, so even a large improvement in time to compute cash does not have much impact on the overall time to process a trade.

Table 3: Complexity of Operations

Method	**Operation**	**Complexity**
PJT	Space (# values stored)	$\mathcal{O}(ucs^w)$
Trade-based	Space (# values stored)	$\mathcal{O}(cs^w + ubs^z)$
All	Time for probability update	$\mathcal{O}(cs^w)$
PJT	Time to compute cash	$\mathcal{O}(cs^w)$
Trade-based	Time to compute cash	$\mathcal{O}(as^z)$
PJT	Time to compute expected score	$\mathcal{O}(ucs^w)$
Trade-based	Time to compute expected score	$\mathcal{O}(uas^w)$

The expected score computation must be performed for every user each time scores are updated. Ideally, this would happen after every edit, but resource constraints may necessitate less frequent updates. Equation (14) involves iterating over asset blocks and computing the expectation with respect to the clique in the probability junction tree that contains the asset block. Therefore, the computation is linear in the number of users and the number of asset blocks, and exponential in the treewidth of the probability junction tree. Achieving good performance in the expected score computation requires keeping the number of asset blocks from growing too large. Recall that the asset block update operation gives the algorithm designer discretion in combining asset blocks subject to the required conditions of structural congruence and frugality. We have found that good performance of the expected value computation requires more than the minimum amount of merging. We use the heuristic of assigning each new asset block to a clique in the probability junction tree that contains the asset block, and merging all asset blocks assigned to the same clique. Using this heuristic, when edits are confined to only a few cliques in the asset junction tree, there will be fewer summands in (14) than in (12), resulting in better performance. The difference can be substantial when there are many users.

5.2 Experimental Procedure

We performed an experiment to compare performance of the different asset management approaches. Our experiment used one of the most natural applications of combinatorial prediction markets, forecasting the outcome of a tournament. Figure 7 shows a Bayesian network representing outcomes of a tournament with eight teams. Chen et al. (2008) used an m-level network of this structure to represent outcomes of a tournament with 2^m teams. Each node represents the outcome of a game. The leaf nodes have two states, each representing the outcome of a game in the first round. The parent of a pair of nodes represents the outcome of a game between the winners of the games represented by the child nodes. The root node, which has $2 * m$ possible values, represents the winner of the tournament.

A junction tree for this Bayesian network is shown in Figure 8. In our simulation, each user was assigned a favorite team T_u, and made edits to change $p(V_k = t_u|V_j = t_u)$ to $x(V_k = t_u|V_j = t_u)$, where V_k is the parent of V_j in the Bayesian network. That is, u edits the probability that her team wins a round given that it won the previous round. It is clear from Figure 8 that these edits are structure preserving.

Our simulation varied the depth of the tree from $m = 4$ to $m = 6$. We assumed a market of 30 users with 240 trades in each round, or an average of 8 trades per user. For each of 50 runs, the simulation proceeded as follows.

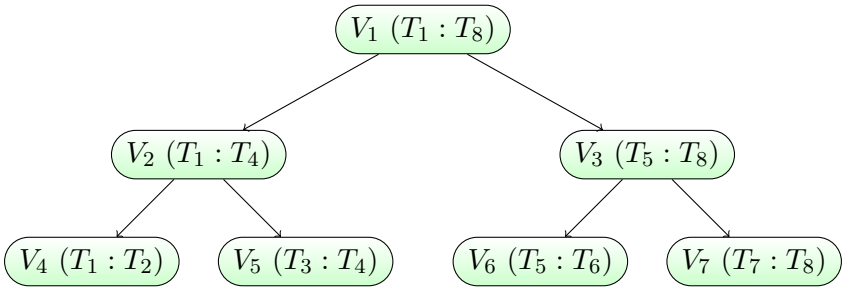

Figure 7: Bayesian Network for Three-Level Tournament

1. Generate a random depth m and an m-level Bayesian network to represent a tournament with 2^m teams.
2. For each user, select a random team T_u. Select all nodes containing T_u as the nodes u edits.
3. For each of the 240 trades, perform the following steps.
   (a) Select a user to make the trade. For the first 30 trades, cycle through the users, so that each user has at least one trade. For the rest of the trades, select a trader at random.
   (b) Generate a parent-child pair from the possible trades involving u's favorite team. Generate a random trade for user u to change the probability of a win of the parent node given a win of the child node.
   (c) Update the consensus probability distribution given the edit, recording the computation time.

(d) Calculate u's new cash using each of the cash calculation methods, recording the computation time.

(e) Calculate the expected score for all users using each of the expected score calculation methods, recording the computation time.

4. Record the storage required for the consensus probability distribution and all users' asset structures for PJT and each of the trade-based asset management methods.

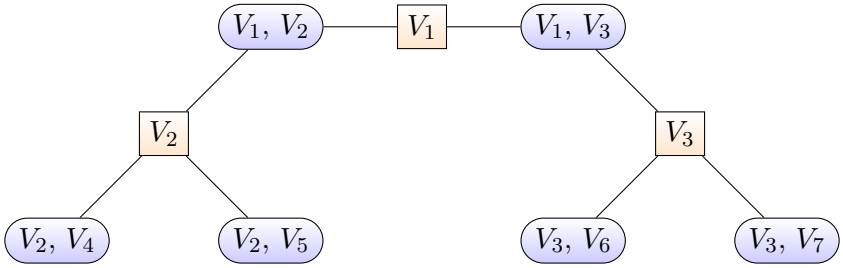

Figure 8: Junction Tree for Figure 7 Bayesian Network.

5.3 Experimental Results

Figure 9 shows results of the experiment. The upper left panel shows the logarithm of the time to compute cash for each of the three methods as a function of the number of levels in the tournament. The upper right panel shows the time to compute the expected score for the trade-based asset block and the parallel junction tree representations, again as a function of the number of levels in the tournament. The third panel compares the total time to perform an entire update, which includes making an edit (common to all three methods), calculating cash (different for all three methods), and computing expected score (common for GS and DAC, different for PJT). The bottom right panel shows memory usage in bytes to store the asset structures for the trade asset block and parallel junction tree methods. In all these graphs, the error bars span a range from the 5th to the 95th percentile of the observations.

The cash computation comparison shows that both trade-based representations outperform the parallel junction tree method for small m, but as the depth of the tree increases, the global separator method degrades until at $m = 6$ it is considerably worse than the parallel junction tree representation. This happens because the asset blocks for each user's trades form a chain of clusters starting at the leaf node containing the team's first round game, passing through the intermediate levels, ending at the cluster containing the final game. For example, with the model of Figure 8, a user who likes T_3 will trade on the clusters $\{V_5, V_2\}$ and $\{V_2, V_1\}$. All $m-2$ intermediate clusters along this chain (V_2 in our T_3 fan's asset model) will be in the global separator for the chain of trade blocks. By contrast, the DAC asset junction tree will have a chain of clusters of size 2 and length $m-1$ with separators of size 1. This means that the treewidth z_{DAC} of the DAC asset junction tree is 2, whereas the treewidth z_{GS} of the GS asset junction tree is $m-1$. Because cash computation is exponential in the treewidth of the asset junction tree (see Table 3, performance degrades rapidly as m increases.

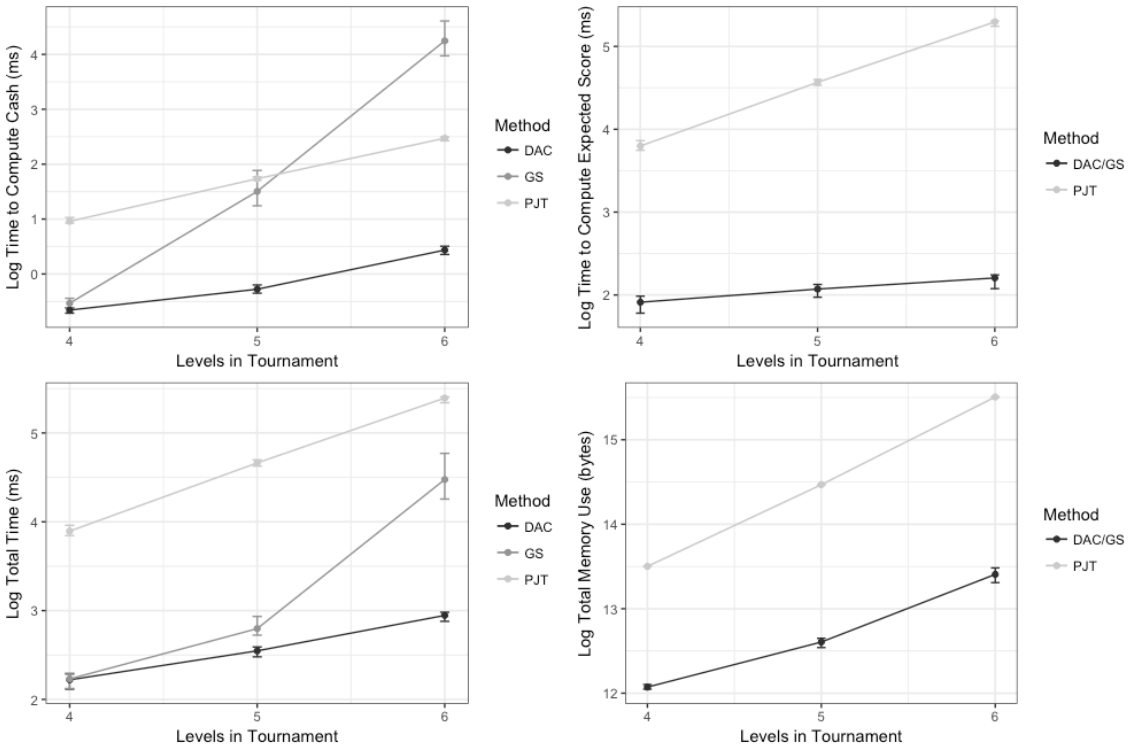

Figure 9: Results of Experiment

The expected score computation is the same for both DAC and GS, and takes much less time than for PJT. In total time, PJT performance is worse than both GS and DAC, but as the number of levels increases, GS performance begins to approach PJT performance.

The PJT method averaged about 8 times the computation time taken by DAC. Broken down by m, the average was 5.4 times for $m = 4$, 8.3 times for $m = 5$, and 11.5 times for m=6.

Clearly, trade-based asset management can save computation time over PJT. The global separator method is simpler to implement than DAC, and is faster when there are few overlaps in the user's trade blocks, but becomes much more expensive as the number of overlaps increases. In extreme cases, the GS cash computation can dominate and degrade performance to the point where PJT becomes competitive. The DAC method avoids this problem.

The plot of memory usage shows a clear dominance of the trade-based asset data structure over the parallel junction trees. On average, PJT uses about 6 times as much memory as the trade block representation. Broken down by m, PJT uses 4.2 times as much storage for $m = 4$, 6.5 times as much storage for $m = 5$, and 8.2 times as much storage for $m = 6$.

6. Conclusion

Combining forecasts from multiple sources has been shown to improve accuracy, and prediction markets are among the most accurate methods for achieving this task. Combinatorial

prediction markets allow forecasting agents to arrive at a consensus joint distribution over multiple related propositions by making conditional trades and/or trades on Boolean combinations of events. This paper presented methods based on probabilistic graphical models for managing probabilities and assets in a logarithmic market scoring rule combinatorial prediction market, and experiments comparing their performance. These methods make it feasible to easily run large-scale combinatorial prediction markets with hundreds of questions and thousands of forecasting agents.

The methods presented in this paper were implemented in the DAGGRE and SciCast prediction markets, which were open to forecasters in the general public over a four-year period from 2011 through 2015. The DAGGRE public LMSR prediction market (Powell et al., 2013) ran from October of 2011 through May of 2013 as part of an Intelligence Advanced Research Projects Activity (IARPA) program to improve aggregate forecast accuracy (Tetlock, Mellers, Rohrbaugh, & Chen, 2014, p.17). The market focused on forecasting world events, including events with extended time horizons (usually 1-12 months) and considerable uncertainty. DAGGRE began as a flat prediction market. A combinatorial feature launched in October of 2012. At any given time, there were on the order of 100 active questions on the market. Over the 20 months the market was open, more than 3000 participants contributed at least one forecast, with an average of about 80 forecasts by 20 users on any given day. The market averaged about 2 edits per question per day. Questions were removed as their outcomes became known, and new questions were introduced on a regular basis.

SciCast, also sponsored by IARPA, was a public LMSR combinatorial prediction market focused on forecasting science and technology innovation (Twardy et al., 2014). SciCast was designed to handle at least a 10-fold increase in the number of live questions and active forecasters. Performance limitations of the DAGGRE market necessitated a move to a user-specific, trade-based asset system. The market opened in November of 2013 using the global separator asset management algorithm. The plan was to move to the dynamic asset cluster algorithm if performance became an issue. An approximation was implemented to compute cash if the global separator became too large to handle. The approximation was conservative, ensuring that no user would run out of assets but sometimes disallowing edits the user could have covered. There were very few instances where the system had to resort to approximation, and consequently the dynamic asset cluster method was never introduced into the public market.

Acknowledgments

This research was supported by the Intelligence Advanced Research Projects Activity (IARPA) via Department of Interior National Business Center contract number D11PC20062. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, DoI/NBC, or the U.S. Government. The authors thank the anonymous reviewers for thoughtful comments that have helped us to strengthen the paper.

References

Abernethy, J., Chen, Y., & Vaughan, J. W. (2013). Efficient market making via convex optimization, and a connection to online learning. *ACM Transactions on Economics and Computation*, *1*(2), 12:1–12:39. doi: 10.1145/2465769.2465777

Arrow, K., Forsythe, R., Gorham, M., Hahn, R., Hanson, R., Ledyard, J. O., ... Zitzewitz, E. (2008, May). The promise of prediction markets. *Science*, *320*(5878), 877-878.

Barbu, A., & Lay, N. (2012). An introduction to artificial prediction markets for classification. *Journal of Machine Learning Research*, *13*, 2177–2204.

Chen, Y., Dimitrov, S., Sami, R., Reeves, D., Pennock, D., Hanson, R., ... Gonen, R. (2010). Gaming prediction markets: Equilibrium strategies with a market maker. *Algorithmica*, *58*(4), 930-969. doi: 10.1007/s00453-009-9323-2

Chen, Y., Fortnow, L., Lambert, N., Pennock, D. M., & Wortman, J. (2008). Complexity of combinatorial market makers. In *Proceedings of the 9th ACM conference on electronic commerce (EC)* (pp. 190–199).

Chen, Y., Goel, S., & Pennock, D. M. (2008). Pricing combinatorial markets for tournaments. In *Proceedings of the 40th annual ACM symposium on theory of computing (STOC-2008)* (pp. 305–314).

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

Dawid, A. P. (1992). Applications of a general propagation algorithm for probabilistic expert systems. *Statistics and Computing*, *2*, 25–36.

Dudík, M., Lahaie, S., & Pennock, D. M. (2012). A tractable combinatorial market maker using constraint generation. In *Proceedings of the 13th acm conference on electronic commerce* (pp. 459–476). New York, NY, USA: ACM. doi: 10.1145/2229012.2229047

Dudík, M., Lahaie, S., Rothschild, D., & Pennock, D. (2013). A combinatorial prediction market for the U.S. elections. In *Proceedings of the fourteenth acm conference on electronic commerce* (pp. 341–358). New York, NY, USA: ACM. doi: 10.1145/2482540.2482601

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

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

Horn, C. F., Ivens, B. S., Ohneberg, M., & Brem, A. (2014, September). Prediction markets: A literature review 2014. *The Journal of Prediction Markets*, *8*(2), 89–126. doi: 10.5750/jpm.v8i2.889

Jeffrey, R. C. (1990). *The logic of decision* (2nd ed.). Chicago: University Of Chicago Press.

Koski, T., & Noble, J. (2009). *Bayesian networks: An introduction* (1st ed.). Wiley.

Langevin, S., & Valtorta, M. (2008). Performance evaluation of algorithms for soft evidential update in Bayesian networks: First results. In *Proceedings of the second international conference on scalable uncertainty management (SUM-08)* (pp. 294–297).

Lauritzen, S. L., & Spiegelhalter, D. J. (1988). Local computations with probabilities on graphical structures and their applications to expert systems. *Journal of the Royal Statistical Society, Series B*, *50*, 157–224.

Li, X., & Vaughan, J. W. (2013). An axiomatic characterization of adaptive-liquidity market makers. In *Proceedings of the Fourteenth ACM Conference on Electronic Commerce*

(pp. 657–674). New York, NY, USA: ACM. doi: 10.1145/2482540.2482575

Malinowski, E. (2010, March). Predictalot brings Wall Street to March Madness. *WIRED*. (Retrieved 2018/09/25 from https://www.wired.com/2010/03/predictalot-brings-wall-street-to-march-madness/)

Pearl, J. (1990). Jeffrey's rule, passage of experience, and neo-Bayesianism. In *Knowledge representation and defeasible reasoning* (pp. 245–265). Kluwer Academic Publishers.

Pennock, D., & Xia, L. (2011). Price updating in combinatorial prediction markets with Bayesian networks. In *Proceedings of the twenty-seventh conference annual conference on uncertainty in artificial intelligence (UAI-11)* (pp. 581–588). Corvallis, Oregon: AUAI Press.

Powell, W. A., Hanson, R., Laskey, K. B., & Twardy, C. (2013). Combinatorial prediction markets: An experimental study. In W. Liu, V. Subrahmanian, & J. Wijsen (Eds.), *Scalable uncertainty management* (Vol. 8078, pp. 283–296). Berlin Heidelberg: Springer.

Savage, L. J. (1971). Elicitation of personal probabilities and expectations. *Journal of the American Statistical Association*, *66*(336), pp. 783-801.

Surowiecki, J. (2005). *The wisdom of crowds* (Reprint edition ed.). New York: Anchor.

Tetlock, P. E., Mellers, B. A., Rohrbaugh, N., & Chen, E. (2014, August). Forecasting Tournaments: Tools for Increasing Transparency and Improving the Quality of Debate. *Current Directions in Psychological Science*, *23*(4), 290–295. doi: 10.1177/0963721414534257

Twardy, C., Hanson, R., Laskey, K., Levitt, T. S., Goldfedder, B., Siegel, A., ... Maxwell, D. (2014). SciCast: collective forecasting of innovation. In *Proceedings of the 2014 collective intelligence conference.* Cambridge, MA, USA.

Tziralis, G., & Tatsiopoulos, I. (2007). Prediction markets: An extended literature review. *Journal of Prediction Markets*, *1*(1), 75–91.

Valtorta, M., Kim, Y.-G., & Vomlel, J. (2002). Soft evidential update for probabilistic multiagent systems. *International Journal of Approximate Reasoning*, *29*(1), 71-106.

Wolfers, J., & Zitzewitz, E. (2006, March). *Prediction markets in theory and practice* (Working Paper No. 12083). National Bureau of Economic Research. doi: 10.3386/w12083