ICE: An Iterative Combinatorial Exchange*

David C. Parkes[†] [‡] Ruggiero Cavallo[†] Nick Elprin[†] Adam Juda[†] Sébastien Lahaie[†]
Benjamin Lubin[†] Loizos Michael[†] Jeffrey Shneidman[†] Hassan Sultan[†]

ABSTRACT

Combinatorial exchanges (CEs) facilitate trade between multiple buyers and sellers that need to express complex preferences on bundles of items. We present the first design for an iterative combinatorial exchange (ICE). The exchange incorporates a tree-based bidding language that is concise and expressive for CEs. Winner-determination is directly formulated in terms of the structure of this language. The main innovation is that bidders interaction with proxy agents, and specify lower and upper valuations for trades by annotating the tree with value intervals. The design is entirely symmetric, handling buyers, sellers and mixed buyers and sellers. The proxy approach facilitates early price discovery and ensures useful preference elicitation even in early rounds. Constraint generation is used to generate linear prices without enumerating all possible trades. A proxied interpretation of a revealed-preference activity rule ensures progress. At termination, a VCG-based payment scheme that has been shown to mitigate opportunities for bargaining and strategic behavior is used to determine payments. The exchange is fully implemented, running, and in a validation phase.

1. INTRODUCTION

Combinatorial exchanges combine and generalizes two different mechanisms: double auctions and combinatorial auctions. In a double auction (DA), multiple buyers and sellers trade units of an identical good [21]. In a combinatorial auction (CA), a single seller has multiple heterogeneous items up for sale [12]. Buyers may have complementarities or substitutabilities between goods and have an expressive bidding language. A common goal in both market designs is to determine the efficient allocation that maximizes total value.

A combinatorial exchange is a combinatorial double auction that brings together multiple buyers and sellers to trade multiple heterogeneous goods. For example, in an exchange for wireless spectrum, a bidder may declare that she is willing to pay $1 million for a trade where she obtains licenses for New York City, Boston, and Philadelphia, and loses her license for Washington DC. Thus, unlike a DA, a combinatorial exchange allows for participants with complex valuations and supports expressive bids. Unlike a CA, a combinatorial exchange allows for multiple buyers and multiple sellers, as well as mixed agents that are both buying and selling.

Combinatorial exchanges (CEs) have received recent attention both in the context of wireless spectrum allocation [19] and for airport takeoff and landing slot allocation [3]. In both of these domains there are incumbents with property rights, and it is important to facilitate a complex multi-way reallocation of resources. Another potential application domain for CEs is to the problem of resource allocation in federated distributed systems, such as PlanetLab [14]. Our exchange is general purpose by design, and can be instantiated for different domains. This instantiation is an extremely compelling direction for the next step in our research.

This paper presents the first design for an iterative combinatorial exchange (ICE). The genesis of this project was a class, CS 286r "Topics at the Interface between Economics and Computer Science," taught at Harvard University in Spring 2004.[1] The entire class was dedicated to the design and prototyping of an iterative combinatorial exchange. The ICE design problem is multifaceted, and in general very hard. The four main challenges are:

Bidding language. CEs require an expressive and concise bidding language, that extends to buyers, sellers, and mixed agents that are both buying and selling items.

Elicitation. Iterative CEs are essential because they allow for adaptive preference elicitation. It is unreasonable to expect a bidder to provide her complete value for all possible trades in most realistic applications of CEs.

Winner determination. Winner determination is the problem of determining a set of trades given bids. It is important that winner determination scales to domains of interest.

Payment and Activity Rules. Bidders are self-interested, and any design must provide a *payment rule* to promote design goals (such as allocative efficiency), and together with an *activity rule* to promote continual and straightforward bidding.

*Thanks to all participants in CS 286r, Spring 2004.

[†]Corresponding author. Remaining authors in alphabetical order. parkes@eecs.harvard.edu

[‡]Division of Engineering and Applied Sciences, Harvard University, Cambridge MA 02138.

EC'05, June 5–8, 2005, Vancouver, British Columbia, Canada.

[1]http://www.eecs.harvard.edu/~parkes/cs286r/ice.html

In the design for our *Iterative Combinatorial Exchange* (ICE), we focused mainly on the problems of bidding languages, elicitation and activity rules. Current evidence from one-sided CAs [10, 29] suggests that elicitation is typically more of a bottleneck in expressive markets than winner determination (WD). We expect this to continue to hold for CEs, even though there is some evidence to suggest that WD is harder in CEs than in CAs. For a payment rule, we stick with the general framework of VCG-based payments as introduced by Parkes et al. [25].

There are a number of innovations in the design. First, we introduce an expressive and concise tree-based bidding language. The language describes values for *trades*, such as "my value for selling AB and buying C is \$100," or "my value for selling ABC is -\$50," with negative values indicating that a bidder must receive a payment for the trade to be acceptable. The language can represent the powerful XOR/OR language [22], but is much more general. The language shares some structural elements with the $\mathcal{L}_{GB}$ language [8], but has very different semantics.

Second, we advocate a proxied design in which bidders interact with the exchange by stating and refining bounds on their values for trades. In CEs, the proxy approach is important to allow for price discovery and useful elicitation in early rounds. Bidders state upper and lower valuation bounds by placing value intervals at nodes in a bid tree.

Third, the design is multi-round, with linear prices computed in each round to guide preference elicitation. We adopt linear rather than non-linear prices because they are easier for bidders to interpret, and because they are computationally efficient to handle within the exchange when enforcing activity rules. Here, we extend earlier ideas due to Kwasnica et al. [18] and Dunford et al. [13] to compute prices that approximately support the current allocation. Our methods use constraint generation to extend previous linear programming formulations, to allow for expressive bidding languages without requiring that individual bundles are enumerated and to allow for markets with buyers and sellers.

Fourth, the design adopts the revealed-preference activity rules of Ausubel et al. [1] to a proxied iterative exchange. We formulate a variation on the winner determination problem to generate a *witness* trade, that is used to determine whether or not the proxy agent has enough information for her best-response to be well defined given current prices. The methods that we adopt to determine whether activity is satisfied must carefully account for shared uncertainty across trades, where the value of a trade is not independent of the value of another trade because of structure in the bid tree.

The exchange is fully coded, and runs, and is currently in validation. Later in the paper we present some examples of prices generated within the exchange, and also some initial scalability examples on a simple instance generator.

1.1 Outline

In Section 2 we provide a simple example of a small exchange instance that we will return to later in the paper. Section 3 provides preliminaries and defines the problem. Section 4 motivates the proxy-based design. Section 5 describes the tree-based bidding language. Section 6 defines the winner-determination step and price-feedback step that concludes each round. Section 7 defines the revealed-preference activity rule. Section 8 gives termination conditions. Section 9 makes some comments about the systems infrastructure issues in developing our prototype. We conclude in Section 10. The Appendix provides examples of bids that can be formulated in our bidding language, some additional MIP formulations, and the results of two simple scalability tests.

1.2 Related Work

Many ascending-price one-sided CAs are known in the literature [26, 30, 11]. Direct elicitation approaches have also been proposed for one-sided CAs in which queries are generated of agents [15, 9, 20]. Especially relevant for this work is that a number of ascending CAs are with linear prices [18, 13]. The price generation methods that we use in ICE are generalizations of the methods in these earlier papers.

Parkes et al. [25] studied sealed-bid combinatorial exchanges and introduced the Threshold payment rule. Subsequently, Krych demonstrated experimentally that Threshold promotes efficient allocations. We are not aware of any previous studies of iterative CEs.

Dominant strategy DAs are known for unit demand [21] and also for single-minded agents [2]. No dominant strategy mechanisms are known for the general CE problem.

Voucher-based schemes [19] have been proposed as a method to extend one-sided CAs to exchanges. For instance, the FCC spectrum is not highly fragmented and it is important to design mechanisms to facilitate efficient reallocation. Incumbents must be reallocated at the same time as new spectrum is allocated. Vouchers are proposed as a simple scheme to allow revenue to be shared across incumbents. Our ICE design is much more expressive, and provides equal and symmetric expressiveness to all sides of the market.

The ICE design that we propose is a hybrid design in the sense that we couple linear prices with a combinatorial winner determination problem and Threshold payments to compute the final outcome. Recently, Ausubel et al. [1] proposed a hybrid one-sided CA, that they call *clock proxy*. Clock proxy also combines linear prices in early rounds with winner determination and non-linear prices in the final round. We adopt the revealed-preference activity rule in clock-proxy, providing a proxied interpretation.

2. WORKING EXAMPLES

In this section, we provide two simple examples of instances that we will use to illustrate various components of the exchange.

Example A

		value
Seller	A OR B OR C OR D	0
Buyer 1	A AND B	6
Buyer 2	A XOR B	4
Buyer 3	C AND D	3
Buyer 4	C XOR D	2

Goods: A, B, C, D.
Initial allocation: (ABCD,$\emptyset, \emptyset, \emptyset, \emptyset$).
Efficient allocation: $(\emptyset, AB, \emptyset, CD, \emptyset)$.

In Example A, the seller is willing to sell any number of the goods (OR), buyers 1 and 3 want both goods (AND) and buyers 2 snd 4 want at most one good (XOR). In the bid tree language, the OR is represented with a [0,4] IC constraint,

the AND with a [2,2] IC constraint and the XOR with a [1,1] constraint.

Example B

		value
Seller	A AND B AND C AND D	-18
Buyer 1	A AND B	11
Buyer 2	C AND D	8

Goods: A, B, C, D.
Initial allocation: (ABCD,$\emptyset, \emptyset$).
Efficient allocation: $(\emptyset, AB, CD)$.

In the Example B, the seller wants to sell all four items and must receive a payment of at least \$18. Buyer 1 wants both of A and B, and buyer 2 wants both of C and D.

3. PRELIMINARIES

In our model, we consider a set of goods, indexed $\{1, \ldots, m\}$ and a set of bidders, indexed $\{1, \ldots, n\}$. The initial allocation of goods is denoted $x^0 = (x_1^0, \ldots, x_n^0)$, with $x_i^0 = (x_{i1}^0, \ldots, x_{im}^0)$ and $x_{ij}^0 \geq 0$ for good j indicating the number of units of good j held by bidder i. A trade $\lambda = (\lambda_1, \ldots, \lambda_n)$ denotes the change in allocation, with $\lambda_i = (\lambda_{i1}, \ldots, \lambda_{im})$ where $\lambda_{ij} \in \mathbb{Z}$ is the change in the number of units of item j to agent i. So, the final allocation $x^1 = x^0 + \lambda$.

Each bidder has a value $v_i(\lambda_i) \in \mathbb{R}$ for a trade λ_i. This value can be positive or negative, and represents the *change in value* between the final allocation $x_i^0 + \lambda_i$ and the initial allocation x_i^0. Utility is quasi-linear, with $u_i(\lambda_i, p) = v_i(\lambda_i) - p$ for trade λ_i at price p. Price p can be negative, indicating the bidder receives a payment for the trade. We use the term *payoff* interchangeably with *utility*.

Our goal in the ICE design is to implement the efficient allocation. The efficient trade, λ^*, maximizes the total increase in value across bidders.

Definition 1 (Efficient trade). *The efficient trade λ^* solves*

$$\max_{(\lambda_1,\ldots,\lambda_n)} \sum_i v_i(\lambda_i) \qquad (V(n))$$

$$\text{s.t.} \quad \lambda_{ij} + x_{ij}^0 \geq 0, \quad \forall i, \forall j \qquad (1)$$

$$\sum_i \lambda_{ij} \leq 0, \quad \forall j \qquad (2)$$

$$\lambda_{ij} \in \mathbb{Z} \qquad (3)$$

Constraints (1) ensure that no agent sells more items that it has in its initial allocation. Constraints (2) provides free disposal, and allows feasible trades to sell more items than are purchased (but not vice versa). It is useful to use $Feas(x^0)$ to denote the feasible trades, given these constraints.

The bidding language allows a bidder to report both lower and upper bounds on its value to the exchange. Lower bounds are *pessimistic* values, and denoted $\underline{v}(\lambda)$. Upper bounds are *optimistic* values, and denoted $\overline{v}(\lambda)$. In each round, we will compute linear prices $p = (p_1, \ldots, p_m)$. Given these prices, then the price on trade λ for agent i is $p(\lambda) = \sum_j p_j \cdot \lambda_{ij}$.

Definition 2 (competitive equilibrium prices). *Prices p^* are competitive equilibrium if the efficient trade λ^* is supported at prices p^*, so that for each bidder*

$$\lambda_i^* = \arg \max_{\lambda \in Feas(x^0)} \{v_i(\lambda_i) - p^*(\lambda_i)\} \qquad (4)$$

Linear competitive equilibrium prices will not always exist for the exchange [5]. In these cases we will compute approximate equilibrium prices to guide progress in the exchange.

The Threshold payment rule is based on the payments in the VCG mechanism [16], which itself is truthful, efficient but does not satisfy budget balance.

Definition 3 (budget-balance). *The total payments to the exchange are non-negative for all bids.*

In VCG, the payment paid by agent i is

$$p_{\text{vcg},i} = \hat{v}(\lambda_i^*) - (V^* - V_{-i}) \qquad (5)$$

where λ^* is the efficient trade, V^* is the reported value of this trade, and V_{-i} is the reported value of the efficient trade that would be implemented without bidder i. We call $\Delta_{\text{vcg},i} = V^* - V_{-i}$ denote the *VCG discount*.

The Threshold payment rule [25] allocates available surplus when the exchange clears to minimize the maximal across all agents to the VCG outcome.

Definition 4. *The Threshold payment scheme implements the surplus-maximizing trade λ^* given bids, and sets payments $p_{\text{Thresh},i} = \hat{v}_i(\lambda_i^*) - \Delta_i$, where $\Delta = (\Delta_1, \ldots, \Delta_n)$ is set to minimize $\max_i(\Delta_{\text{vcg},i} - \Delta_i)$ subject to $\Delta_i \leq \Delta_{\text{vcg},i}$ and $\sum_i \Delta_i \leq V^*$.*

Example 1. *In Example A, the efficient trade gives AB to agent 1 and CD to agent 3. Agent 1 makes payment \$4, agent 3 makes payment \$2 and the seller receives payment \$9. Threshold would take the available surplus of \$9 and divide it so that \$1 goes to agent 1, 0 goes to agent 2 and \$8 goes to the seller. This minimizes the worst-case error to the VCG payoff across all payments.*

Threshold payments minimize the maximal *ex post* opportunity for manipulation, given bids from others, across all simple CEs that satisfy budget-balance and participation. The result of Threshold is that each bidder's ex post regret after price-adjustment is $\Delta_{\text{vcg},i} - \Delta_i$. Krych [17] confirmed that Threshold promotes allocative efficiency in restricted and approximate Bayes-Nash equilibrium.

4. MAKING THE CASE FOR A PROXIED DESIGN

A common design for one-sided iterative CAs is to design an *ascending price* auction [26, 30, 11]. Here, one proposes a set of prices and collects best-responses from bidders. The price on bundles for which there is over demand is increased. Eventually, the prices are in competitive equilibrium and the demand of every buyer can be satisfied. These prices support the efficient allocation.

It is natural to ask whether a single monotonic price trajectory can be effective for iterative CEs. For increasing prices, $p_j^{t+1} \geq p_j^t$ for all goods, j, and all rounds t, vice versa for decreasing prices. Thus, with increasing prices a pure buyer would see decreasing utility across rounds while a pure seller would see increasing utility across rounds. Neither increasing and decreasing price trajectories are useful

because they are asymmetric for buyers and sellers, requiring monotonic utility concessions on only one side of the market.

One can allow for a pair of monotonic price trajectories, for instance increasing for buyers and sellers so that both sides of the market forfeit utility across rounds. However, when prices are increasing to both buyers and sellers the aggregate demand across rounds does not change monotonically (it depends on whether the rate at which demand decreases is greater or less than the rate at which supply decreases). A similar problem occurs for decreasing prices on both sides of the market, and here an additional problem is that there are no bids in early rounds.[2]

In both cases, effective price discovery and effective elicitation seems difficult. For these, and other reasons, we advocate a *proxied* design [27]. A single price trajectory is maintained, but need not be adjusted monotonically. The design is entirely symmetric, and facilitates focused elicitation in all rounds. Crucially, we require that a bidder provides both *upper* and *lower* bounds on valuation to her proxy.[3]

In each round, the provisional allocation and prices are computed based on a linear combination, $\alpha\underline{v} + (1-\alpha)\overline{v}$, for $0 \leq \alpha \leq 1$, of the pessimistic and optimistic valuations. Parameter α is initialized to zero, so that the exchange operates with optimistic values in early rounds. Later, the exchange clears with pessimistic values and these are the values that define the final allocation and payments.

EXAMPLE 2. *In Example A, initial bounds could be* $[2, 8]$, $[1, 10]$, $[2, 6]$ *and* $[3, 5]$ *for the buyers and* $[0, 0]$ *for the seller. Now, if clearing at the optimistic outcome then the trade would be to buyers 2 and 3. The trade at the pessimistic outcome would be to allocate to buyer 1 and buyer 4.*

Proxied designs also restrict the bidding strategy of participants to *incremental revelation of value information.* This restriction brings a useful strategic simplification to the design of iterative auctions [27, 1].

One drawback to requiring explicit upper and lower bounds is that they can limit the opportunity for learning in settings with correlated value across bidders. If needed, the consistency rule that requires tighter bounds across rounds can be relaxed in early rounds to facilitiate this behavior.

5. TREE-BASED BIDDING LANGUAGE

Central to our design is a concise and expressive bidding language. The language defines the interface between bidders and the proxy agents, and represents lower and upper bounds on values for trades. The bidding language is designed to be entirely symmetric with respect to buyers and sellers, and naturally extends to capture bids from mixed buyers and sellers, ranging from simple swaps to highly complex trades.

Bids are expressed as annotated *bid trees*, and define a bidder's value for all possible trades. While the language shares some structural elements with the $\mathcal{L}_{gb}$ language of Boutilier and Hoos [6], the semantics of our language are very different. Boutilier and Hoos provide a logic-based interpretation, where the same items in an allocation can satisfy a tree in multiple places. In comparison, our language works by allocating items to satisfy one particular part of the tree. Boutilier and Hoos demonstrate that their language, with this ability to trigger shared parts of the tree, provides new conciseness in some settings. However, we believe that it is important that trees have a *locality* in meaning so that the value of a component that a bidder adds to a tree can be understood independently of the rest of the tree.

The main feature of the language is that it has a general "choose between x and y" logical connective, together with careful semantics for propagating values in the tree. Given this, it is a simple matter to capture XOR/OR bidding languages as a bid tree in our language.[4]

Let $T = (N, \mathit{leaf}, \mathit{child}, \overline{v}, \underline{v}, IC, \mathit{trade})$ denote a bid tree. $N = (n_0, \ldots, n_k)$ denotes the set of nodes, with n_0 the root. $\mathit{child} : N \rightarrow 2^L$ is the operator that defines the children $\mathit{child}(n) \subseteq L$ of a node n. $\mathit{leaf}(n) \in \{0, 1\}$ indicates whether or not node n is a leaf. Let $L = \{n : n \in N, \mathit{leaf}(n)\} \subseteq N$.

A well-formed tree must be connected and acyclic, and have $\overline{v}(n) \geq \underline{v}(n)$ for all nodes $n \in N$. Function $\overline{v} : N \rightarrow \mathbb{R}$ defines the upper-bound on a node. Function $\underline{v} : N \rightarrow \mathbb{R}$ defines the lower-bound on a node. For internal nodes $n \in N \setminus L$, function $IC(n) = [a, b]$ defines the *interval choose* operator, with $IC_x(n) = a$ and $IC_y(n) = b$. Interval choose is not defined for leaves. For leaves $l \in L$, then $\mathit{trade}(l) = [j, Q_j]$ with $j \in \{1, \ldots, m\}$ and $Q_j \in \mathbb{Z}$. This defines a trade in which the agent's allocation changes by Q_j units (perhaps negative) of item j. We write $\mathit{trade}_G(l) = j$ and $\mathit{trade}_Q(l) = Q_j$.

Trees are used to evaluate the change in value for trades. Given trade λ_i to agent i, we can define a tree in which a subset of nodes are *satisfied*. Let $\mathit{sat}(n) \in \{0, 1\}$ denote whether or not node n is satisfied.

DEFINITION 5 (VALID). *Solution sat is valid, written* $\mathit{sat} \in \mathit{valid}(T, \lambda_i)$, *given tree* T *and trade* λ_i *if and only if*

1. $\sum_{n \in L, \mathit{trade}_G(n)=j} \mathit{sat}(n) \cdot \mathit{trade}_Q(n) \leq \lambda_{ij}, \quad \forall j$
2. $\mathit{sat}(n) \Leftrightarrow IC_x(n) \leq \sum_{n' \in \mathit{child}(n)} \mathit{sat}(n') \leq IC_y(n)$, *for all* $n \in N \setminus L$.

In words, satisfied nodes $\mathit{sat}(n)$ define a valid solution to a tree, given a trade, if the following hold. First, the total trade as defined across the satisfied children must be consistent with the items traded (subject to free-disposal). Second, an internal node is satisfied if and only if between x and y children of the node are also satisfied. Given this, we can define the semantics for a bidder's value, given tree T and a trade.

DEFINITION 6 (TREE VALUES). *Given trade* λ_i, *the upper-value defined by tree* T *is*

$$\overline{v}_i(T, \lambda_i) = \max_{\mathit{sat} \in \mathit{valid}(T, \lambda_i)} \sum_{n \in N} \mathit{sat}(n) \cdot \overline{v}(n) \qquad (6)$$

Given trade λ_i, *the lower-value defined by tree* T *is*

$$\underline{v}_i(T, \lambda_i) = \max_{\mathit{sat} \in \mathit{valid}(T, \lambda_i)} \sum_{n \in N} \mathit{sat}(n) \cdot \underline{v}(n) \qquad (7)$$

[2] A familiar design for one-sided auctions is the Dutch auction, which employs decreasing prices and stops as soon as the first bid is received [16].

[3] The idea of upper and lower bound queries was suggested in previous work [23, 24, 15] but the idea of using an explicit lower and upper bound representation of values seems to be new.

[4] The problem of whether XOR/OR dominates OR* in terms of expressiveness, or vice versa remains open [22].

The tree-based language generalizes existing languages. For instance: $IC(2,2)$ on a node with 2 children is equivalent to an AND operator; $IC(1,3)$ on a node with 3 children is equivalent to an OR operator; and $IC(1,1)$ on a node with 2 children is equivalent to an XOR operator.

LEMMA 1. *Given a well-formed tree, then* $\underline{v}(\lambda_i) \leq \overline{v}(\lambda_i)$ *for all trades.*

PROOF. Suppose some λ_i for which $\underline{v}(\lambda_i) > \overline{v}(\lambda_i)$. Then, $\max_{sat \in valid(T,\lambda_i)} \sum_{n \in N} sat(n) \cdot \underline{v}(n) > \max_{sat \in valid(T,\lambda_i)} \sum_{n \in N} sat(n) \cdot \overline{v}(n)$. But, this is a contradiction because the trade λ' that defines $\underline{v}(\lambda_i)$ is still feasible with upper bounds $\overline{v}$, and $\overline{v}(n) \geq \underline{v}(n)$ for all nodes in a well-formed tree. □

Given a tree, we define a partial order $T' \prec T$, that determines whether tree T is tighter than tree T'.

DEFINITION 7 (TIGHTER). *Tree T is is tighter than tree T', denoted $T' \prec T$, if and only if T and T' share the same structural components $(N, leaf, child, IC, trade)$, are well-formed, and differ only in upper and lower bounds, with $\overline{v}(n) \leq \overline{v}'(n), \quad \forall n$, $\underline{v}(n) \geq \underline{v}'(n), \quad \forall n$, and for which there is a node $\tilde{n} \in N$, for which $\overline{v}(\tilde{n}) < \overline{v}'(\tilde{n})$ or $\underline{v}(\tilde{n}) > \underline{v}'(\tilde{n})$.*

The symmetry in the language, with negative and positive values, and sells and buys in leaves, is important to make it expressive and natural. The use of the general $[x, y]$ operator makes the language concise. Appendix **A1** provides many examples of valuations expressed in this bid-tree language.

To highlight one design decision, notice that we choose to require a bidder to specify her upper and lower valuations on the same tree. The bidder simply annotates nodes with upper and lower values. This proves to be very important in defining an efficient method to check the revealed-preference activity rule, and also to ensure consistency holds across rounds.

For now, we allow a bidder to change the value information but not the bounds or other structure within the tree. This extension is useful in practice, but requires some careful thought. For instance, any new structure should be limited so that an agent cannot circumvent consistency or revealed preference.

6. WINNER DETERMINATION AND PRICE FEEDBACK

This section defines the winner determination problem and the pricing problem solved by the exchange at the end of each round. The proxy agents pass the bid trees $T = (T_1, \ldots, T_n)$ for each agent. Winner determination, denoted WD(v), takes valuations from bidders (in the tree language) and computes a surplus maximizing trade. The winner determination problem is reused many times within the exchange, e.g. for column generation in pricing and for checking revealed preference.

Pricing, denoted CE(v, λ), takes a trade λ and valuations v (in the tree language) and computes approximate competitive equilibrium prices. While the exchange is still open:

1. Solve WD($\underline{v}$), to get trade $\underline{\lambda}$ and WD($\overline{v}$), to get trade $\overline{\lambda}$.
2. Define $\alpha = \frac{\sum_i \underline{v}_i(\overline{\lambda})}{\sum_i \underline{v}_i(\underline{\lambda})}$. Define $v^\alpha = (v_1^\alpha, \ldots, v_n^\alpha)$, where v_i^α is defined by setting bounds $\underline{v}(n)$ and $\overline{v}(n)$ on every node in the bid tree to $\alpha \underline{v}_i(n) + (1-\alpha)\overline{v}_i(n)$.
3. Solve WD(v^α) to find trade λ^α, and compute prices $p^\alpha = \text{CE}(v^\alpha, \lambda^\alpha)$.
4. When $\alpha > cutoff$ go to *last-and-final* round.

Both the winner determination and pricing problems are defined as terms of mixed-integer programs and linear programs, and solved in our system through repeated calls to a commercial solver.[5] The solver uses branch-and-bound search with dynamic cut generation and branching heuristics to solve large mixed integer programs (MIP)s in an economically feasible run time.

6.1 Winner Determination

The purpose of winner determination in an iterative combinatorial exchange is to calculate the surplus-maximizing trade. In defining the mixed-integer program (MIP) representation, which is generated on-the-fly in our ICE system, we are careful to avoid an XOR-based enumeration of all bundles. Rather, as in Boutilier [7] we form a concise representation that directly encodes the tree-based structure. The MIP formulation is defined for a particular fixed value at each node, which could be $\overline{v}(n), \underline{v}(n)$, or $v^\alpha(n)$. The constant p_β is set to what the value is for a node.

Constants

- $x_{ij} \in \{0\} \cup \mathbb{N}$, initial quantity of good j held by agent i.
- $x_j = \sum_i x_{ij}$, for all $j \in \{1, \ldots, m\}$
- $x_\beta \in \{0\} \cup \mathbb{N}$: The satisfaction lower bound for node β.
- $y_\beta \in \{0\} \cup \mathbb{N}$: The satisfaction upper bound for node β.
- $p_\beta \in \mathbb{R}$: The value annotated within the tree for node β.
- $q_\beta \in \mathbb{Z}$: The quantity of the item requested (for purchase or sale) at **leaf** node β.

Variables

- $b_{ij} \in \{0\} \cup \mathbb{N}$: Agent i **bought** this many units of good j.
- $s_{ij} \in \{0\} \cup \mathbb{N}$: Agent i **sold** this many units of good j.
- $s_\beta \in \{0, 1\}$: Indicator variable for the satisfaction of node β.

Objective Function

Maximize the sum of values of all agents:

$$\max_{b,s,s_\beta} \sum_i \sum_\beta p_\beta \cdot s_\beta$$

where s_β is an indicator to state whether node β is satisfied or not in the tree.

[5] CPLEX, www.ilog.com

Supply constraints

FOR EVERY GOOD j: Total demand must not exceed total supply:

$$\sum_i b_{ij} \leq \sum_i s_{ij}$$

This assumes free disposal. For goods that have negative value (e.g. toxic waste), this must be set to an equality explicitly in order to force efficient trades.

FOR EVERY AGENT i, EVERY GOOD j: Agents cannot sell more than they had initially:

$$s_{ij} \leq x_{ij}$$

Satisfaction Rules

Let Buy_{ij} denote the set of leaf nodes of agent i that request a certain quantity of good j to be bought. Let Sell_{ij} similarly denote the set of leaf nodes of agent i that request a certain quantity of good j to be sold. Let $\text{child}(\beta)$ denote the children of node β. We have $\text{child}(\beta) = \emptyset$ if and only if node β is a leaf.

FOR ALL LEAF NODES β:

Set appropriate quantity of buy nodes:

$$\sum_{\beta \in \text{Buy}_{ij}} q_\beta \cdot S_\beta \leq b_{ij}, \quad \forall j$$

Set appropriate quantity of sell nodes:

$$\sum_{\beta \in \text{Sell}_{ij}} q_\beta \cdot S_\beta \geq s_{ij}, \quad \forall j$$

FOR ALL NON-LEAF (INTERNAL) NODES β:

At least x_β children must be satisfied for parent to be satisfied:

$$\sum_{\beta_i \in \text{child}(\beta)} S_{\beta_i} \geq x_\beta \cdot s_\beta$$

At most y_β children can be satisfied if the parent is satisfied, otherwise **none** can be satisfied:

$$\sum_{\beta_i \in \text{child}(\beta)} S_{\beta_i} \leq y_\beta \cdot s_\beta$$

Taken together, these constraints ensure strict propagation on the tree with a parent satisfied iff the right number of children are satisfied, and vice versa.

Final allocation

The final allocation can be backed-out from the trade, with $x^1_{ij} = x_{ij} + b_{ij} - s_{ij}$, for all agents i and all goods j and an initial quantity x_{ij} of each good.

Some goods may go unassigned because free disposal is allowed within the clearing rules of winner determination.[6] Any unassigned items can be allocated back to agents that sold the items, i.e. for which $b_{ij} - s_{ij} < 0$.

6.2 Linear Price-Feedback

The exchange design relies on price feedback to guide progress, since it is the prices that define the revealed preference activity rule. In each round, we have an allocation and reported values. A set of informative prices would satisfy:

1. (accurate) The provisional trade maximizes the surplus for each bidder at the current prices.
2. (accurate, balanced) An agent that is not engaged in trade can understand how to adjust her bid values to begin to trade.
3. (linear) Prices should be compact and easy to express to make them useful in defining activity rules and guiding elicitation.
4. (fair) Prices should approximate the final payments in the exchange.

In some settings, such as CAs with substitutes valuations, prices— the linear competitive equilibrium prices we defined in Section 3 —will exist. However, ideal prices fail to exist in general CEs. Instead, we compute approximate prices that are as accurate as possible, and then break ties to favor *balance* and *fairness*. We borrow ideas due to Rassenti et al. [28], Kwasnica et al. [18] and Dunford et al. [13] but introduce the following innovations:

1. Approximate competitive equilibrium prices are defined for an *expressive* bidding language. Previous approaches (implicitly) assume an additive-or bidding language, and can handle full expressiveness only with the addition of *dummy goods* that lead to non-anonymous prices.
2. Constraint generation is used to avoid enumerating all feasible trades in the formulation for approximate competitive equilibrium prices.
3. Prices are computed through a sequence of linear programs, first to achieve *accuracy*, then to achieve *fairness*, and third to achieve *balance*.

Defining Linear Competitive Equilibrium Prices: Accuracy

Prices are computed as a sequence of linear programs (LPs), with the winner determination MIP called as a subroutine. We assume that bidder trees have been instantiated with a fixed value at each node, which will be $v^\alpha(n)$ in each round of ICE. Let $\lambda^t = (\lambda^t_1, \ldots, \lambda^t_n)$ denote the current provisional allocation, in round t. An empty trade is denoted $\lambda^t_i = \emptyset$ and has value $v_i(\lambda^t_i) = 0$ by definition.

Let I denote the set of bidders, with W denoting the set of winners with $\lambda^*_i \neq \emptyset$ for all $i \in W$. Let $p = (p_1, \ldots, p_m)$ denote linear prices, with $p(\lambda_i)$ to denote the total price on trade λ_i to bidder i. Recall that we assume quasi-linear utility, with $u_i(\lambda, p) = v_i(\lambda) - p$ to denote the utility for a trade. Ideal (CE) prices would satisfy the following conditions:

$$v_i(\lambda) - p(\lambda) \leq 0, \quad \forall i \in I \setminus W \tag{8}$$

$$v_i(\lambda) - p(\lambda) \leq v_i(\lambda^t) - p(\lambda^t), \quad \forall i \in W \tag{9}$$

In general, ideal prices will not exist and we compute approximate prices by using an LP to minimize the maximal error across all bidders:

$$\min_{p,\delta} \delta \qquad \text{[PD]}$$

$$\text{s.t.} \quad v_i(\lambda) - p(\lambda) \leq v_i(\lambda^t_i) - p(\lambda^t_i) + \delta, \quad \forall i, \forall \lambda \tag{10}$$

$$\delta \geq 0$$

$$p(k) \geq 0, \quad \forall k \in G$$

[6] If goods are not subject to free disposal then this can be changed by simply writing the supply constraints with a strict equality.

Eqs. (8) and (9) are represented as a single set of constraints (10), because $\lambda_i^t = \emptyset$ and $v_i(\lambda_i^t) = p(\lambda_i^t) = 0$ for losers $i \in I \setminus W$.

Notice that prices can be perfectly accurate even when the price on some trade $\lambda_i \neq \lambda_i^*$ is less than value $v_i^\alpha(\lambda_i)$, as long as $v_i^\alpha(\lambda_i) - p(\lambda_i) \leq v_i^\alpha(\lambda_i^*) - p(\lambda_i^*)$. The formulation also treats all bidders symmetrically, both winners and losers.[7]

A Constraint Generation Approach

We adopt constraint generation, to dynamically generate a sufficient subset of the exponential constraints (10) in [PD]. This is essential, because to do otherwise would undo the benefits of the concise bid-tree language and require enumeration of values for all feasible trades. The subproblem, to generate new constraints, is a simple variation on winner determination.

Constraint generation considers a relaxed program that only contains a manageable subset of the constraints, and solves this to optimality [4]. An optimal solution to the relaxed program is also optimal for the original program, provided it is feasible for the latter. Once we solve the relaxed program, we therefore need a way to test for feasibility in the original. This can usually be done by solving another LP or integer program.

In our case, let $\mathbb{F}_i$ denote a manageable subset of all possible feasible trades to bidder i. The relaxed pricing problem is simply:

$$\min_{p,\delta} \quad \delta \qquad \text{[ACC]}$$

$$\text{s.t. } v_i(\lambda) - p(\lambda) \leq v_i(\lambda_i^t) - p(\lambda_i^t) + \delta, \ \forall i, \forall \lambda \in \mathbb{F}_i \qquad (11)$$

$$\delta \geq 0$$

$$p(k) \geq 0, \quad \forall k \in G$$

The solution (p^*, δ^*) must be tested for feasible in the original problem [PD]. To test for feasibility we solve the following for each bidder i. Note that the prices and are now constants, and the goal is to determine the surplus maximizing trade at these prices.

$$\max_{\lambda} \quad v_i(\lambda) - p^*(\lambda) \qquad \text{[R-WD]}$$

$$\text{s.t.} \quad \lambda \in Feas \qquad (12)$$

where *Feas* is the set of all feasible trades for bidder i.

Problem [R-WD] is a simple refinement of the winner determination problem. The objective that maximized the total value to bidders is replaced with the new objective. The other bidders are kept round in the MIP to define the space of feasible trades.

Let $\hat{\lambda}$ denote the solution to [R-WD]. Check condition $v_i(\hat{\lambda}_i) - p^*(\hat{\lambda}) \leq v_i(\lambda_i^t) - p^*(\lambda_i^t) + \delta^*$, and if this is satisfied then (p^*, δ^*) satisfies the full set of constraints in [PD] that refer to bidder i. When these constraints hold for *all* bidders, we can terminate because we know that solution (p^*, δ^*) is feasible. Otherwise we add $\hat{\lambda}_i$ to the set $\mathbb{F}_i$, and re-solve [ACC] with the new constraints.

[7]Earlier work on approximate competitive equilibrium prices considers all bids on trades separately, implicitly assuming an additive-or logic across trades [13, 18]. The formulations impose a constraint that the price should be exact on trades executed and minimize the error on other trades. This ignores the structure that follows when one considers that multiple bids come from a single bidder.

Prices are lexicographically optimized, with an initial solution to [ACC] refined to minimize the maximal error across other bidders, but without degrading the accuracy of the solution.[8] In each stage in this lexicographic minimization, constraint generation is used as an inner-loop to continue to generate new trades for $\mathbb{F}_i$. Details of the lexicographical minimization are presented in Appendix 2. At the end of this phase, solution $(\delta_1^*, \ldots, \delta_n^*)$ is carried through to the first tie-breaking stage.

Example 3. *Consider Example A, and suppose that the proxy agents have true agent values. The efficient trade is to allocation AB to bidder 1 and CD to bidder 3. Accuracy will seek prices $p(A), p(B), p(C)$ and $p(D)$ to minimize the worst-case δ across these constraints:*

$$\begin{aligned} p(A) + p(B) &\leq 6 + \delta \\ p(A) + \delta &\geq 4 \\ p(B) + \delta &\geq 4 \\ p(C) + p(D) &\leq 3 \\ p(C) + \delta &\geq 2 \\ p(D) + \delta &\geq 2 \\ p(A) + p(B) + p(C) + p(D) &\geq 0 \end{aligned}$$

First, setting $p(A) = p(B) = 10/3$ and $p(C), p(D)$ so that the error on bidders 3 and 4 is $\leq 2/3$ defines a space of solutions. However, with lexicographical minimization we fix $\delta_1^* = \delta_2^* = 2/3$ and then continue to drive down the error to bidders 3 and 4. Finally, this gives $p(A) = p(B) = 10/3$ and $p(C) = p(D) = 5/3$, with accuracy $2/3$ to bidders 1 and 2 and $1/3$ to bidders 3 and 4.

Pricing: Breaking Ties for Fairness

Second, we break remaining ties to prefer fair prices. All formulations include the constraints from *accuracy*, to ensure that the prices are just as accurate as at the end of the accuracy phase. We define *fairness* as follows: *all else equal, we prefer prices that minimize the maximal surplus to bidders that trade.*

Example 4. *In Example B, accuracy can select prices $p = (0, 11, 8, 0)$ for items A, B, C and D. Prices $(10.66, 0, 0, 7.66)$ are just as accurate, but divide the surplus equally across the buyers and the sellers in the trade. These prices are selected when breaking ties for fairness.*

We adopt *fairness* because it is likely to push prices close to approximating the payments in the Threshold rule.[9]

As an LP, we formulate the *fairness* tie-breaking method as follows:

$$\min_{p,\pi} \quad \pi \qquad \text{[FAIR]}$$

$$\text{s.t.} \quad v_i(\lambda) - p(\lambda) \leq v_i(\lambda_i^t) - p(\lambda_i^t) + \delta_i^*, \forall i, \forall \lambda \in \mathbb{F}_i \qquad (13)$$

$$\pi \geq v_i(\lambda_i^t) - p(\lambda_i^t), \quad \forall i \in W \qquad (14)$$

$$\pi \geq 0$$

$$p(k) \geq 0, \quad \forall k \in G$$

[8]This technique was also used in Kwasnica et al. for the pricing problem in a one-sided CA.

[9]The methods of Dunford et al. [13], that use a nucleous approach, are also closely related.

where λ^* is the efficient trade and δ_i^* are the optimal errors, as computed in [ACC]. As before, we solve [FAIR] with constraint generation, to check that a solution (p^*, π^*) is feasible in the original program: we solve [R-WD] for every bidder to check that constraints (13) are not violated (and grow $\mathbb{F}_i$ otherwise). Once an optimally fair solution is found, subject to accuracy as defined in [ACC], lexicographical minimization is again adopted to break ties to sequentially minimize the maximal payoff. This is presented in Appendix 2. Eventually, information $\{\pi_i^* : i \in W\}$ and $(\delta_1^*, \ldots, \delta_n^*)$ is carried through to the final tie-breaking stage.

Pricing: Breaking Ties for Balance

This time, we break the remaining ties to prefer equal prices, again without undoing any of the progress made up to this point. We define *balance* as follows: *all else equal, prefer prices that minimize the maximal price across all items.*

To motivate this, consider a simple example with two items and a choice of prices from accuracy and fairness of $[0, 10], [10, 0]$ and $[5, 5]$. Given knowledge that items are more likely to be alike than dissimilar, then we would prefer prices $[5, 5]$ and these are selected by the balance tie-break.

EXAMPLE 5. *In Example B, the prices at the end of fairness are* $(10.66, 0, 0, 7.66)$. *The balance stage changes the prices to prefer* $(3.83, 5.33, 3.83, 5.33)$. *These prices are just as accurate and just as fair, but are better in terms of the smoothing of price across items.*

Balance is justified when, all else being equal, items are more likely to be similar in value to each other other than different.[10]

As an LP, we formulate the *balance* tie-breaking method as follows:

$$\min_{p,Y} \quad Y \qquad \text{[BAL]}$$

$$\text{s.t.} \quad v_i(\lambda) - p(\lambda) \leq v_i(\lambda_i^t) - p(\lambda_i^t) + \delta_i^*, \forall i, \forall \lambda \in \mathbb{F}_i \quad (15)$$

$$\pi_i^* = v_i(\lambda_i^t) - p(\lambda_i^t), \quad \forall i \in W \quad (16)$$

$$Y \leq p(k), \quad \forall k \in G \quad (17)$$

$$Y \geq 0$$

$$p(k) \geq 0, \quad \forall k \in G$$

where λ^* is the provisional allocation, δ^* is passed from [ACC], and π^* is passed from [FAIR].

Constraint generation is used to check that a solution (p^*, Y^*) is feasible in the original program, just as for [FAIR]. Finally, balance is also defined lexicographically, to sequentially minimize the maximal price, as defined in the appendix.

Pricing: Putting This Together

Finally, the balance phase terminates with prices $p^* = (p_1^*, \ldots, p_m^*)$, that define prices for the next round of the exchange. These prices are reported back to proxy agents.

Constraint generation can be costly, and is optimized in our system by retaining trades in $\mathbb{F}_i$ that define constraints across rounds of the exchange and also across stages, from *accuracy*, to *fairness*, to *balance*. Within a round, each call to column generation can add a new trade to $\mathbb{F}_i$, which augments the current set. Typically, we should expect that the set of useful trades will not change very much across rounds, or across price generation stages.

[10]The use of *balance* was initially advocated by Kwasnica et al. [18]. Dunford et al. [13] prefer an alternative method to determine item prices, which is to smooth prices across rounds (for instance minimizing sum square deviation in price). Smoothing seems less important given that ours is a proxied system and prices are used indirectly.

7. ACTIVITY RULES

Activity rules ensure both consistency and progress throughout rounds. Consistency in our proxied exchange requires that bidders tighten bounds as the exchange progresses. Activity rules ensure that bidders are active during early rounds, so as to promote useful elicitation throughout the exchange.

The ICE activity rule that we propose uses prices, and has the nice property that it is *agnostic* to the structure selected for a bid tree. A bidder must reveal the same information to meet activity irrespective of how that information is represented.

We adopt a simple *revealed-preference* activity rule, as proposed for the clock-proxy auction by Ausubel et al. [1]. Ausubel et al. argue that the revealed-preference rule is preferred to simpler quantity-based rules, for instance in removing incentives for parking on some large set of low price goods to retain activity before jumping in with a large volume just as an auction is about to close.

In a proxied auction, the revealed-preference (RP) rule requires that *the proxy has enough information in each round to determine a best-response trade for the bidder, given current prices.* To satisfy RP, the bidder must refine the upper and lower values in her bid tree until this best-response trade is well defined.[11]

Failure to meet activity must have some consequence. We propose, as a default action, to set the upper bounds in valuations down to the maximal value of the current prices and the lower-bound. This is entirely analogous to when a bidder in an ascending clock auction stops bidding: *she cannot enter bidding again in the future.* Here, the bidder can be active within the exchange, but cannot state values higher than current prices.

Let $v \in T$ denote the set of valuations that are defined by some consistent refinement to bid tree T.

DEFINITION 8. *Given prices p, then bid tree T satisfies revealed-preference if and only if there exists some feasible trade λ_l^* for which $v(\lambda_l^*) - p(\lambda_l^*) \geq v(\lambda') - p(\lambda')$ for all $\lambda' \neq \lambda_l^*$ and all $v \in T$.*

We need a constructive method to check this condition given tree T, as well as to provide *feedback* to guide a bidder in meeting the RP rule.

7.1 Testing for Revealed Preference

Consider prices p, and tree T that defines lower and upper valuations $\underline{v}$ and $\overline{v}$ on trades.

First, we construct a candidate lower-bound trade, λ_l^*,

[11]Note that we impose this revealed-preference rule even though the best-response is not used within the exchange to update the provisional allocation or prices. As described in Section 6, these updates are defined in terms of the complete proxy information.

which is a feasible trade that solves

$$\max_{\lambda} \quad \underline{v}_i(\lambda) - p(\lambda) \qquad \text{[RP1]}$$

$$\text{s.t.} \quad \lambda \in \mathit{Feas} \qquad (18)$$

where *Feas* is the set of all feasible trades for bidder i. The solution, π_l^*, is the maximal payoff the bidder can achieve across all feasible trades, given its pessimistic valuation. Problem [RP1] is a different instantiation of the exact same problem as [R-WD], used for constraint generation in pricing. Thus, it can be solved as a simple variation on the winner determination problem.

Second, we take the optimal payoff π_l^* from [RP1] and break ties to ensure that we have the trade that maximizes *value uncertainty*, defined as $\overline{v}_i(\lambda) - \underline{v}_i(\lambda)$ across all trades with payoff π_l^* at lower valuations:

$$\lambda_l^* = \arg\max_{\lambda} \quad \overline{v}_i(\lambda) - \underline{v}_i(\lambda) \qquad \text{[RP2]}$$

$$\text{s.t.} \quad \lambda \in \mathit{Feas} \qquad (19)$$

$$\underline{v}_i(\lambda) - p(\lambda) \geq \pi_l^* \qquad (20)$$

We adopt solution λ_l^* as our candidate for the trade that is the best-response across all $v \in T$ given current prices.

Third, we construct $\tilde{v}_i$ by setting the values on nodes equal to the upper-bound values on nodes in T except for nodes that are satisfied in λ^*. The value on these nodes in $\tilde{v}_i$ is set to the lower-bound values on the nodes in T. Given this valuation $\tilde{v}_i$, then we solve

$$\max_{\lambda} \quad \tilde{v}_i(\lambda) - p(\lambda) \qquad \text{[RP3]}$$

$$\text{s.t.} \quad \lambda \in \mathit{Feas} \qquad (21)$$

Let π_u^* denote the payoff from this optimal trade given adjusted values $\tilde{v}$, and λ_u^* denote the associated trade. We call this trade the *witness*.

Finally, the RP rule is satisfied if and only if $\pi_l^* \geq \pi_u^*$. If the rule is not satisfied, then we provide a *hint* to the bidder.

7.2 Activity Rule: Feedback

Let $sat(\lambda_l^*) \subseteq N$ and $sat(\lambda_u^*) \subseteq N$ denote the set of nodes that are satisfied given trades λ_l^* and λ_u^*.

LEMMA 2. *Given that RP is not satisfied, then any changes in values in the tree that satisfy RP must increase the lower bound on at least one node in $sat(\lambda_l^*) \setminus sat(\lambda_u^*)$ and decrease the upper bound on at least one node in $sat(\lambda_u^*) \setminus sat(\lambda_l^*)$.*

PROOF. Observe that changing values on nodes that are not satisfied by either trade cannot prevent the witness proving the candidate is not a best-response for some valuation. Changing the values on these nodes would have no effect. Then, changing the bounds on nodes that are satisfied in both trades also has no effect on revealed preference. Finally, the only nodes that can be usefully revised are those in $sat(\lambda_l^*)$ and $sat(\lambda_i^*)$. □

Thus, by providing the nodes $sat(\lambda_l^*) \setminus sat(\lambda_u^*)$ as candidates for increases in lower bound values, and nodes $sat(\lambda_u^*) \setminus sat(\lambda_l^*)$ as candidates for decreases in upper bound values, we can provide a hint to the bidder. One or more of these suggested trades *must* be performed to meet the RP rule.

This is an elegant feedback mechanism because it is adaptive. Once a bidder makes some changes on some subset of these nodes, then the bidder can query her proxy and ask for more feedback. The nodes reported back might change, indicating that the bidder should now refine values on a different part of three.

7.3 Correctness of Revealed Preference Activity Rule

PROPOSITION 1. *Bid tree T with valuations $\underline{v}$ and $\overline{v}$ satisfies revealed-preference given prices p iff a lower-bound trade λ_l^* (breaking ties to prefer uncertainty) satisfies*

$$\underline{v}(\lambda_l^*) - p(\lambda_l^*) \geq \overline{v}|_{\hat{N}}(\lambda') - p(\lambda') \qquad (22)$$

where $\hat{N}$ is the set of nodes that are satisfied in both λ_l^* and λ', and $\overline{v}|_{\hat{N}}$ means that the upper-bounds on nodes in set $\hat{N}$ are set to their lower-bound values.

PROOF. First, if every trade satisfies RP then it is one of the λ_l^* trades. Suppose otherwise, that some $\lambda \neq \lambda_l^*$ is the best-response across all valuations. But, then if values are $\underline{v}$ then the trade λ_i^* is better than the candidate trade λ. Now, from across the λ_i^* we need to show that either there was no trade that satisfied RP or we picked one of the trades for which RP is satisfied.

Case a) Unique λ_l^* trade. Now, the maximal difference in the tree value between λ_l^* and any other trade, consistent with bounds, is exactly $\overline{v}|_{\hat{N}}(\lambda') - \underline{v}(\lambda_l^*)$ for the set $\hat{N}$ of nodes activated in both. This is exactly the condition that we check.

Case b) If there are multiple λ_l^* but all have certain value then this reduces to case a), and exactly check the required condition which λ_i^* we choose.

Case c) If there are multiple λ_l^* and only one of them has an uncertain value, then none of the λ_l^* with certain value can satisfy RP because the λ_l^* trade with uncertain value would be a counterexample. So, the trade with maximal uncertainty should be our candidate trade.

Case d) Now, suppose there are multiple λ_l^*, each with uncertain value. This time, we argue that the only case in which RP can be satisfied is when one of these trades activates a superset of the uncertain nodes of all other trades in the set. Without this, whichever λ_l^* we choose one of the other trades will be a counterexample to RP because its own uncertain node. So, breaking ties to select λ_i^* with maximal uncertainty will always select such a trade when one exists, and find a trade for which RP will fail when RP must fail. □

Notice that the algorithm can simply go ahead and set the upper-bounds equal to lower-bounds on all nodes that are satisfied in the tree for candidate λ_l^*. This is fine, any nodes that are shared will be set correctly and other nodes are not affected.

EXAMPLE 6. *For a simple example, suppose a buyer with A XOR B and a value of* 5 *on the bid(A) node and a range [5,10] on the bid(B) node. Suppose prices are 3 for A and B. If we select $\underline{\lambda}^* = (+1, 0)$ and buy A then trade $(0, +1)$ will dominate for some valuations. However, if we select $\underline{\lambda}^* = (0, +1)$ then this will dominate the other trade for all valuations.*

EXAMPLE 7. *To understand why nodes that share some nodes with uncertain value require care, consider an example with a buyer that has A XOR B and bounds* $[5, 10]$ *on the* root *node and a value of \$1 on A. Given the same prices*

on A and B, then there is enough information to choose A as the best-response and satisfy RP. However, if we chose A as our candidate and then asked for the maximal utility on B then this could be $10 - p$ (for price p) while A could be as much as $6 - p$. To address this problem we resolve all shared uncertainty to a constant by fixing the uncertainty on nodes in the candidate trade λ_i^* to their lower bounds before looking for a witness and testing RP.

8. TERMINATION CONDITIONS

In each round, before setting α and defining the winner determination and price adjustments, we consider the following two values:

Pessimistic at Pessimistic (PP) Compute the efficient trade to solve $\underline{\lambda} = \max_\lambda \sum_i \underline{v}_i(\lambda)$. Set PP=$\underline{v}(\underline{\lambda})$.

Optimistic at Pessimistic (OP) Compute the efficient trade to solve $\overline{\lambda} = \max_\lambda \sum_i \overline{v}_i(\lambda)$. Set OP = $\underline{v}(\overline{\lambda})$.

In early rounds it is likely that PP= 0 because there are no efficient trades at pessimistic values. In some round, PP will be positive when trade first initiates at the pessimistic values. From this round forward, we have PP>OP, because the trade in PP is explicitly defined to solve OP. However, OP might be negative while PP is positive because the optimistic trade is not profitable at pessimistic values. For PP$\neq$ 0, define

$$\beta(PP, OP) = 1 + \frac{PP - OP}{PP}$$

Thus, $\beta(PP, OP)$ is always > 1 and starts off large and trends to 1 as the optimistic allocation converges towards the pessimistic allocation. In each round, we use $0 \leq \alpha \leq 1$ to define a "blended" valuation, $v_i^\alpha = \alpha \underline{v}_i + (1-\alpha)\overline{v}_i$ to use for WD and pricing. We define α as follows:

$$\alpha = \begin{cases} 0 & \text{, when PP is 0} \\ 1/\beta & \text{, otherwise} \end{cases}$$

The effect is to endogenously define a schedule for moving from optimistic to pessimistic values during the exchange, based on how close the trades are to one another.

The termination condition is defined as *move to last-and-final round when $\alpha > CUTOFF$*, where *CUTOFF* is some fraction (e.g. 0.95) that defines completion. The last-and-final round gives bidders one last chance to bid. Finally, in closing the exchange, the Threshold outcome is implemented on the final pessimistic values.

9. SYSTEMS INFRASTRUCTURE ASPECTS

We developed ICE in the Eclipse development environment (www.eclipse.org). The Eclipse environment allowed for the development by a large (15+) coding team, with smaller functionally-oriented groups working on single components. ICE is coded in Java, with XML-based configuration files and an XML-based bid-spec language. The system is designed to support threading and distributed processing, including a load-balancing architecture to allow mutliple MIPs to be solved at once.

We have a fully coded and running version of the exchange. The entire design is very modular, which allows any piece to be easily changed out and replaced. This could be useful if we were to experiment with alternate closing rules, or pricing rules, for example.

For testing purposes we can program agents with a model, that is the agent's customized view of the world and a strategy component. The proxy acts as a "trusted toolbox" for the agent; it provides exchange knowledge to the agent in an unmalleable way to the agent.

Our design allows agents to simulate other agents to help them refine their strategy in a live exchange. For instance, a Southwest Airlines agent could model JetBlue, Delta etc. and ask the exchange to run a simulation to understand the effect of its strategy.

In coding, we noticed a fair amount of overlap in our MIPs. For instance, WD, pricing constraint generation, activity rule, and closing rule all use the same basic methods to generate formulations. We have small "MIPinators" that are the workers used in generating blocks of formalisms, and are re-used by different model formalizers. At a high-level, we have the following components:

- Exchange: lightweight coordination (100 lines of Java) with very top-level calls into the Formalizers.
- Formalizers/Engines:
  - WinnerDeterminationEngine
  - PriceEngine (has 3 stages: Accuracy, Balance, Fairness)
  - ActivityRuleEngine
  - AlphaClosingRuleEngine

The formalizers simply call the appropriate MIPinators, and are each about 50 lines of Java code.

The MIPinators generate an internal MIP representation, which is then mapped to actual MIP variables and constraints. These variables and constraints are expressed in a solver-independent way, via a client that sends the abstract MIP representations over the network to the server (and optionally the load balancer). It is the server which finally converts this abstract MIP description into CPLEX API calls, solves the MIP, and then converts CPLEX's results back into an abstract representation to be handed back up the call chain.

10. CONCLUSION

In this work we design and prototype a scalable and highly-expressive *iterative* combinatorial exchange. The design includes many interesting features, including: a new bid-tree language for exchanges, a new method to construct approximate linear prices from expressive languages, and a proxied method with optimistic and pessimistic valuations. The exchange is fully implemented, running, and in a validation phase.

The next steps for our work are to allow bidders to refine the structure of the bid tree in addition to values on the tree. We are interested to study the elicitation properties of the exchange and put together a test suite of exchange problem instances. In addition, we are beginning to engage in collaborations to apply the design to airline takeoff and landing slot scheduling and to resource allocation in wide-area network distributed computational systems.

11. REFERENCES

[1] L. Ausubel, P. Cramton, and P. Milgrom. The clock-proxy auction: A practical combinatorial auction design. In Cramton et al. [10], chapter 5.

[2] M. Babaioff and W. Walsh. Incentive-compatible, budget-balanced, yet highly efficient auctions for supply chain formation. *Decision Support Systems*, 2004.

[3] M. Ball, G. Donohue, and K. Hoffman. Auctions for the safe, efficient, and equitable allocation of airspace system resources. In S. Cramton, Shoham, editor, *Combinatorial Auctions*. 2004. Forthcoming.

[4] D. Bertsimas and J. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 1997.

[5] S. Bikhchandani and J. M. Ostroy. The package assignment model. *Journal of Economic Theory*, 107(2):377–406, 2002.

[6] C. Boutilier. A pomdp formulation of preference elicitation problems. In *Proc. 18th National Conference on Artificial Intelligence (AAAI-02)*, 2002.

[7] C. Boutilier. Solving concisely expressed combinatorial auction problems. In *Proc. 18th National Conference on Artificial Intelligence (AAAI-02)*, 2002.

[8] C. Boutilier and H. Hoos. Bidding languages for combinatorial auctions. In *Proc. 17th International Joint Conference on Artificial Intelligence (IJCAI-01)*, 2001.

[9] W. Conen and T. Sandholm. Preference elicitation in combinatorial auctions. In *Proc. 3rd ACM Conf. on Electronic Commerce (EC-01)*, pages 256–259. ACM Press, New York, 2001.

[10] P. Cramton, Y. Shoham, and R. Steinberg, editors. MIT Press, 2004.

[11] S. de Vries, J. Schummer, and R. V. Vohra. On ascending Vickrey auctions for heterogeneous objects. Technical report, MEDS, Kellogg School, Northwestern University, 2003.

[12] S. de Vries and R. V. Vohra. Combinatorial auctions: A survey. *Informs Journal on Computing*, 15(3):284–309, 2003.

[13] M. Dunford, K. Hoffman, D. Menon, R. Sultana, and T. Wilson. Testing linear pricing algorithms for use in ascending combinatorial auctions. Technical report, SEOR, George Mason University, 2003.

[14] Y. Fu, J. Chase, B. Chun, S. Schwab, and A. Vahdat. Sharp: an architecture for secure resource peering. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 133–148. ACM Press, 2003.

[15] B. Hudson and T. Sandholm. Effectiveness of query types and policies for preference elicitation in combinatorial auctions. In *Proc. 3rd Int. Joint. Conf. on Autonomous Agents and Multi Agent Systems*, pages 386–393, 2004.

[16] V. Krishna. *Auction Theory*. Academic Press, 2002.

[17] D. Krych. Calculation and analysis of nash equilibria of vickrey-based payment rules for combinatorial exchanges, April 2003.

[18] A. M. Kwasnica, J. O. Ledyard, D. Porter, and C. DeMartini. A new and improved design for multi-object iterative auctions. *Management Science*, 2004. To appear.

[19] E. Kwerel and J. Williams. A proposal for a rapid transition to market allocation of spectrum. Technical report, FCC Office of Plans and Policy, Nov 2002.

[20] S. M. Lahaie and D. C. Parkes. Applying learning algorithms to preference elicitation. In *Proc. ACM Conf. on Electronic Commerce*, pages 180–188, 2004.

[21] R. P. McAfee. A dominant strategy double auction. *J. of Economic Theory*, 56:434–450, 1992.

[22] N. Nisan. Bidding and allocation in combinatorial auctions. In *Proc. 2nd ACM Conf. on Electronic Commerce (EC-00)*, pages 1–12, 2000.

[23] D. C. Parkes. An iterative generalized Vickrey auction: Strategy-proofness without complete revelation. In *Proc. AAAI Spring Symposium on Game Theoretic and Decision Theoretic Agents*, pages 78–87. AAAI Press, March 2001.

[24] D. C. Parkes. Auction design with costly preference elicitation. *Annals of Mathematics and AI*, 2004. To appear.

[25] D. C. Parkes, J. R. Kalagnanam, and M. Eso. Achieving budget-balance with Vickrey-based payment schemes in exchanges. In *Proc. 17th International Joint Conference on Artificial Intelligence (IJCAI-01)*, pages 1161–1168, 2001.

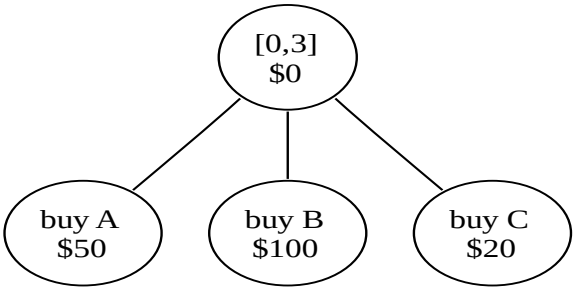
[26] D. C. Parkes and L. H. Ungar. Iterative combinatorial auctions: Theory and practice. In *Proc. 17th National Conference on Artificial Intelligence (AAAI-00)*, pages 74–81, 2000.

[27] D. C. Parkes and L. H. Ungar. Preventing strategic manipulation in iterative auctions: Proxy agents and price-adjustment. In *Proc. 17th National Conference on Artificial Intelligence (AAAI-00)*, pages 82–89, 2000.

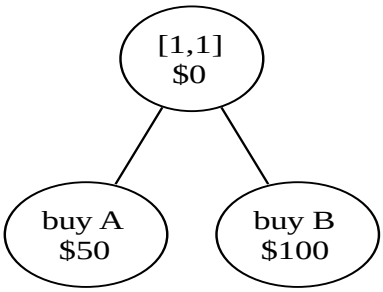
[28] S. J. Rassenti, V. L. Smith, and R. L. Bulfin. A combinatorial mechanism for airport time slot allocation. *Bell Journal of Economics*, 13:402–417, 1982.

[29] T. Sandholm, S. Suri, A. Gilpin, and D. Levine. Winner determination in combinatorial auction generalizations. In *In Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 69–76, 2002.

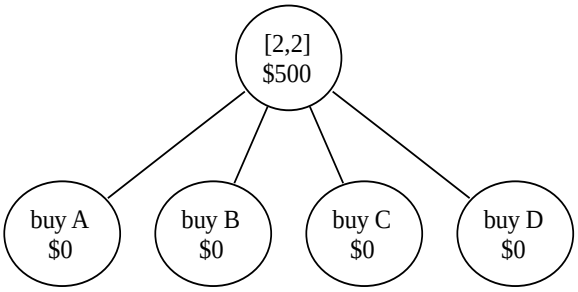
[30] P. R. Wurman and M. P. Wellman. A*k*BA: A progressive, anonymous-price combinatorial auction. In *Second ACM Conference on Electronic Commerce*, pages 21–29, 2000.



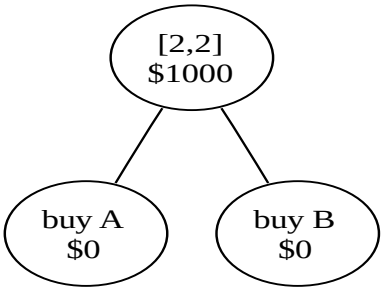
(a) Buy any number of items, at different prices for each.



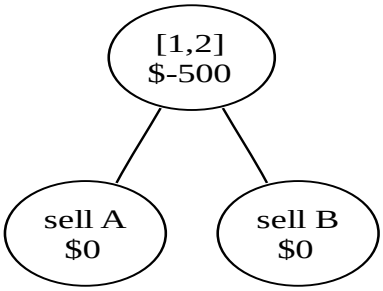
(b) Buy any one item, at different prices for each.



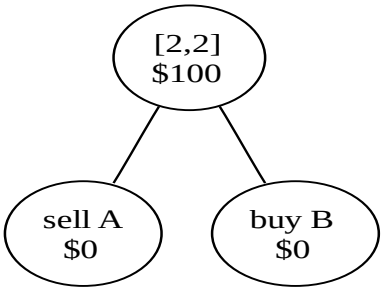
(c) Buy any two items, pay the same amount for any two.



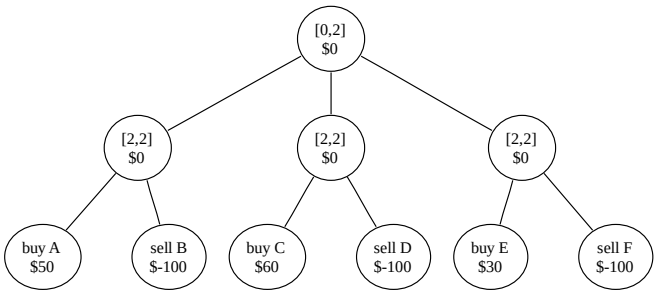
(d) Buy both items for up to $1000, but not one without the other.



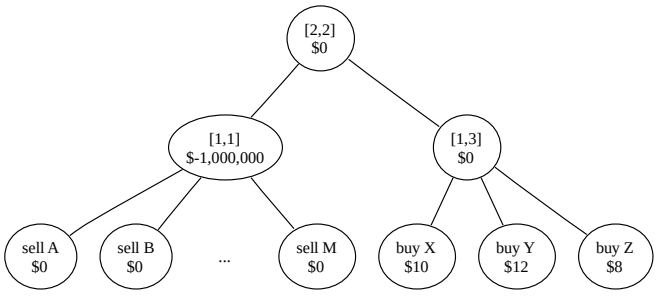
(e) Sell one, or both items, for $500.



(f) Swap A for B, for $100.



(g) Execute at most two of three possible swaps.



(h) Sell at most one high value item ($A - M$) and buy between 1 and 3 new items ($X - Z$), or do not trade.

Figure 1: Appendix A1. Bidding Language Examples.

APPENDIX

A. LEXICOGRAPHICAL TIE BREAKING

First, we formulate a program to lexicographically minimize the maximal error across all bidders. As in RAD [18] this is accomplished through a sequence of LPs, ACC^1,...,ACC^n (worst-case length n). Let I^t be the set of bidders for which the accuracy is already determined, and let δ_i^* be the accuracy to those bidders. $I^1 = \emptyset$ for the first iteration. In each iteration t, we solve:

$$\min_{p,\delta} \quad \delta \qquad [\text{ACC}^t]$$

$$\text{s.t.} \quad v_i(\lambda) - p(\lambda) \leq v_i(\lambda_i^*) - p(\lambda_i^*) + \delta, \forall i \in I \setminus I^t, \forall \lambda \in \mathbb{F}_i \quad (23)$$

$$v_i(\lambda) - p(\lambda) \leq v_i(\lambda_i^*) - p(\lambda_i^*) + \delta_i^*, \forall i \in I^t, \forall \lambda \in \mathbb{F}_i \quad (24)$$

$$\delta \geq 0$$
$$p(k) \geq 0, \quad \forall k \in G$$

In solving this [ACC^t] we must use constraint generation to ensure the solution is feasible to the original problem will all trades included as explicit constraints.

At the end of iteration t, if current solution $\delta^* = 0$ then we can stop. Let $\delta_i^* = 0$ for all remaining i that were in set $I \setminus I^t$ in the current round. Otherwise, we add bidders for which constraint (23) is binding (using dual price information) to set I^t and denote the new set as I^{t+1} for the next iteration. Finally, we set $\delta_i^* = \delta^*$ for the bidders added to set I^t. Now, repeat with [ACC^{t+1}] if we still have $I \setminus I^{t+1} \neq \emptyset$.

For [FAIR], we break ties by lexicographically minimizing the maximal payoff across all bidders. Let J^t be the set of bidders for which the surplus is already determined and let π_i^* be the surplus to such a bidder. Initially we have $J^1 = \emptyset$. In each iteration t, we solve:

$$\min_{p,\pi} \quad \pi \qquad [\text{FAIR}^t]$$

$$\text{s.t.} \quad v_i(\lambda) - p(\lambda) \leq v_i(\lambda_i^*) - p(\lambda_i^*) + \delta_i^*, \forall i, \forall \lambda \in \mathbb{F}_i \quad (25)$$

$$\pi \geq v_i(\lambda_i^*) - p(\lambda_i^*), \quad \forall i \in W \setminus J^t \quad (26)$$

$$\pi_i^* \geq v_i(\lambda_i^*) - p(\lambda_i^*), \quad \forall i \in W \cap J^t \quad (27)$$

$$\pi \geq 0$$
$$p(k) \geq 0, \quad \forall k \in G$$

where W is the set of winners (agents that trade) in the current provisional allocation.

In solving [FAIR^t], we must use constraint generation to check that a solution (p^*, π^*) is feasible in the original program.

At the end of iteration t if the current solution $\pi^* = 0$ then we can stop with $\pi_i^* = 0$ for all bidders that were considered in $W \setminus J^t$ in this final iteration. Otherwise, when $\pi^* > 0$ add the bidders to J^t for which constraint (26) is tight (detected via dual prices) and denote this new set J^{t+1}. Also, set π_i^* for these bidders equal to the current π^*. Continue and solve the next LP in the sequence [FAIR^{t+1}] while $W \setminus J^{t+1} \neq \emptyset$.

For [BAL], we break ties by lexicographically minimizing the maximal price across all goods. Let K^t be the set of items for which the price is already determined and let p_k^* be the price for such an item. Initially we have $K^1 = \emptyset$. In each iteration, t, we solve:

$$\min_{p,Y} \quad Y \qquad [\text{BAL}^t]$$

$$\text{s.t.} \quad v_i(\lambda) - p(\lambda) \leq v_i(\lambda_i^*) - p(\lambda_i^*) + \delta_i^*, \forall i, \forall \lambda \in \mathbb{F}_i \quad (28)$$

$$\pi_i^* \geq v_i(\lambda_i^*) - p(\lambda_i^*), \quad \forall i \in W \quad (29)$$

$$Y \geq p(k), \quad \forall k \in G \setminus K^t \quad (30)$$

$$p_k^* \geq p(k), \quad \forall k \in K^t \quad (31)$$

$$Y \geq 0$$
$$p(k) \geq 0, \quad \forall k \in G$$

In solving [BAL^t] we must use constraint generation to check that a solution (p^*, Y^*) is feasible in the original problem.

At the end of iteration t we add the bidders for which constraint (30) is tight to the set K^t, this set becomes K^{t+1}. Continue and solve the next LP in the sequence, [BAL^{t+1}], while $G \setminus K^{t+1} \neq \emptyset$.

B. EXPERIMENTAL SCALING

We present two examples to demonstrate the computational scalability of the winner determination component of the exchange. We have constructed a general purpose highly-parameterized generator. For the purpose of these scaling experiments we fixed down all of the parameters except tree size (to change the complexity of the valuation models of agents) and the number of agents. The experiments that are presented were completed on a 4 GHz P4 with 512 MB RAM.

For the first experiment in Figure 2 we generate random bid trees centered at a mean of depth 3 with branching factor 3 (i.e. around 40 nodes in each tree). We then scale the number of agents in the system from 2 to 60, and average of 30 instances each time. Problems with 60 agents solve in around 40 seconds. The run time is roughly scaling linearly with the size of the instance, although with more variance at larger sizes. Some of the time that is measured is due to serialization at the server, to send the results back from the solver.

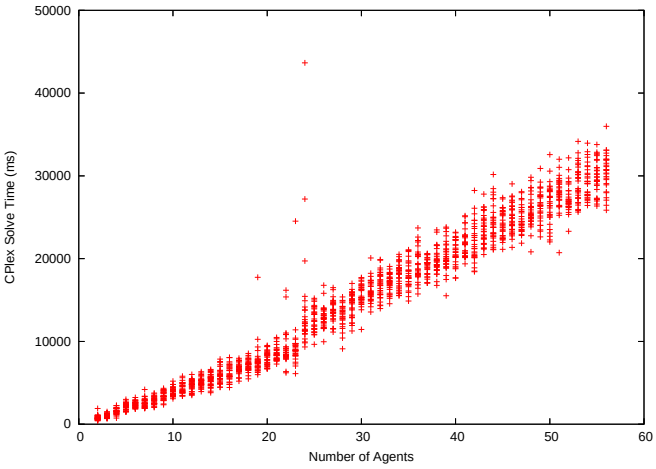


Figure 2: WD run time vs. Number agents, on trees with depth 3 and branching factor 3 (around 40 nodes per tree).

The second experiment, shown in Figure 3 we hold the number of agents constant at 10 and gradually increase both branching factor and depth (solving 30 instances each time). When generating larger bid trees the simulator generates additional items automatically to fill out the trees. Again, this plot shows roughly linear scaling. There are big gaps in the data because when the tree size branching factor increases by 1 there is a large change in tree size (measured in terms of number of internal nodes). We can solve trees with 2500+ nodes and 10 agents in around 140 secs.

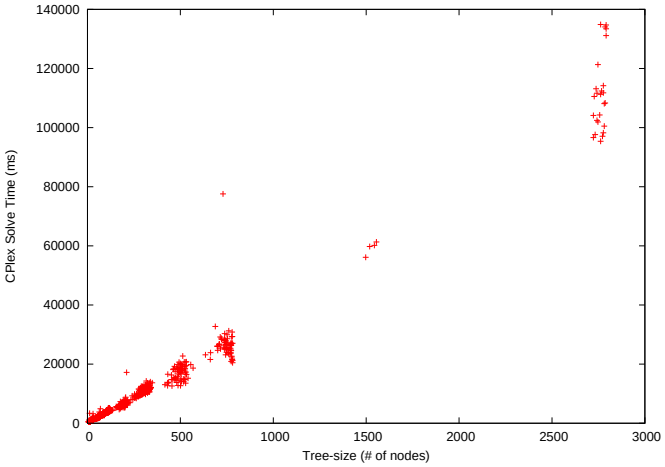


Figure 3: WD run time vs. Size of agent trees, for a fixed number of 10 agents.