Exchange Market for Complex Commodities: Search for Optimal Matches

Eugene Fink
e.fink@cs.cmu.edu

Jianli Gong
gongjianli@hotmail.com

Josh Johnson
jojohnso@csee.usf.edu

Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213

Abstract

The Internet has led to the development of on-line markets, and computer scientists have designed various auction algorithms, as well as automated exchanges for standardized commodities; however, they have done little work on exchanges for complex nonstandard goods.

We present an exchange system for trading complex goods, such as used cars or nonstandard financial securities. The system allows traders to represent their buy and sell orders by multiple attributes; for example, a car buyer can specify a model, options, color, and other desirable features. Traders can also provide complex price constraints, along with preferences among acceptable trades; for instance, a car buyer can specify dependency of an acceptable price on the model, year of production, and mileage.

We describe the representation and indexing of orders, and give algorithms for fast identification of matches between buy and sell orders. The system identifies the most preferable matches, which maximize trader satisfaction; it allows control over the trade-off between speed and optimality of matching. It supports markets with up to 300,000 orders, and processes hundreds of new orders per second.

Keywords: E-commerce, exchange markets, indexing and retrieval, best-first search.

1 Introduction

The growth of the Internet has led to the development of electronic markets, and economists expect that it will play an increasingly vital role in both wholesale and retail transactions [Feldman, 2000]; the Internet marketplaces include bulletin boards, auctions, and exchanges [Klein, 1997; Turban, 1997; Wrigley, 1997; Bakos, 2001].

Electronic bulletin boards help buyers and sellers to find each other; however, they often require customers to invest time into searching among multiple ads, and many buyers prefer on-line auctions, such as eBay (www.ebay.com). Auctions have their own problems, including significant computational costs, lack of liquidity, and asymmetry between buyers and sellers. Exchange-based markets support fast-paced trading and ensure symmetry between buyers and sellers; however, they require rigid standardization of tradable items. For example, the

New York Stock Exchange allows trading of about 3,100 securities, and a buyer or seller must indicate a specific item, such as IBM stock.

For most goods, the description of a desirable trade is more complex; for instance, a car buyer needs to specify a model, options, color, and other desirable features. An exchange for nonstandard goods should satisfy the following requirements:

- Allow complex constraints in specifications of buy and sell orders.
- Support fast-paced trading for markets with millions of orders.
- Include optimization techniques that maximize traders' satisfaction.

We have developed an automated exchange for complex goods, which allows traders to represent buy and sell orders by multiple attributes. In particular, we have defined the related trading semantics [Hu, 2002], created indexing structures for fast identification of matches between buy and sell orders [Johnson, 2001; Fink *et al.*, 2004], and analyzed the scalability of the developed system [Johnson, 2001]. We now give algorithms for fast identification of most preferable matches, which maximize the satisfaction of traders.

2 Previous work

Economists and computer scientists have studied a variety of trading models. The related computer science research has been focused on effective auction systems, optimal matching in various auctions, and general-purpose systems for auctions and exchanges. It has led to successful Internet auctions, such as eBay (www.ebay.com) and Yahoo Auctions (auctions.yahoo.com), as well as on-line exchanges, such as Island (www.island.com) and NexTrade (www.nextrade.com). Recently, researchers have developed several combinatorial auctions, which allow buying and selling sets of commodities rather than individual items.

Combinatorial auctions. A combinatorial auction allows bidding on a set of items; for example, a car buyer can bid on a red Mustang and white Camaro for a total price of $35,000. An advanced auction may allow disjunctions; for instance, a buyer may specify that she wants either a red Mustang and white Camaro or, alternatively, two silver Corvettes. On the other hand, standard combinatorial auctions do not allow incompletely specified items, such as a Mustang of any color.

Rothkopf *et al.* [1998] gave a detailed analysis of combinatorial auctions and described semantics of bids that allowed fast matching. Nisan discussed alternative bid semantics, formalized the problem of searching for optimal and near-optimal matches, and proposed a linear-programming solution [Nisan, 2000; Lavi and Nisan, 2000]. Zurel and Nisan [2001] developed a system for finding near-optimal matches based on a combination of approximate linear programming with optimization heuristics. It quickly cleared an auction with 1,000 items and 10,000 bids, and its average approximation error was less than 1%.

Sandholm [1999] developed several algorithms for one-seller combinatorial auctions, and showed that they scaled to a market with about 1,000 bids. Sandholm and his colleagues later improved the original algorithms and implemented a system that processed several thousand bids [Sandholm, 2000a; Sandholm and Suri, 2000; Sandholm *et al.*, 2001a]. They developed a mechanism for determining a trader's preferences and converting them into a compact bid representation [Conen and Sandholm, 2001]. They also described several special cases of bid processing that allowed polynomial solutions, proved the NP-completeness of more general cases, and tested various heuristics for NP-complete cases [Sandholm *et al.*, 2001b].

Sakurai *et al.* [2000] developed an algorithm for finding near-optimal matches in combinatorial auctions based on a synergy of iterative-deepening A* with limited-discrepancy search; it processed auctions with up to 5,000 bids, and its approximation error was under 5%. Hoos and Boutilier [2000] applied stochastic local search to finding near-optimal matches; it cleared auctions with up to 500 items and 10,000 bids. Akcoglu *et al.* [2000] represented an auction as a graph; its nodes were bids, and its edges were conflicts between bids. This representation led to a linear-time approximation algorithm for clearing the auction.

Fujishima *et al.* [1999a; 1999b] proposed an approach for enhancing standard auction rules, analyzed trade-offs between optimality and running time, and presented two related algorithms. The first algorithm ensured optimal matching and scaled to about 1,000 bids, whereas the second found near-optimal matches for a market with 10,000 bids.

Leyton-Brown *et al.* [2000] investigated combinatorial auctions that allowed bidders to specify a number of items; for instance, a buyer could bid on ten identical cars. They described a branch-and-bound search algorithm for finding optimal matches, which quickly processed markets with fifteen item types and 2,500 bids. Lehmann *et al.* [1999] studied heuristic algorithms for combinatorial auctions and identified cases that allowed truthful bidding, which meant that auction participants did not benefit from providing incorrect information about their intended maximal bids. Gonen and Lehmann [2000, 2001] studied branch-and-bound heuristics for bid processing and integrated them with linear programming. Mu'alem and Nisan [2002] described conditions for ensuring truthful bidding, and proposed approximation algorithms for clearing the auctions that satisfied these conditions.

Yokoo *et al.* [2001a, 2001b] considered the problem of false-name bids, that is, manipulation of prices by creating fictitious auction participants and submitting bids without intention to buy; they proposed auction rules that discouraged such bids. Suzuki and Yokoo [2002] studied another security problem; they investigated techniques for clearing an auction without revealing bids to the auctioneer. They described a distributed dynamic-programming algorithm that found matches without revealing the bids to auction participants or any central auctioneer; however, its complexity was exponential in the number of items.

Andersson *et al.* [2000] compared the main techniques for combinatorial auctions and proposed an integer-programming representation that allowed richer bid semantics. Wurman *et al.* [2001] analyzed a variety of previously developed auctions and identified the main components of an automated auction, including bid semantics, clearing mechanisms, rules for placing and canceling bids, and policies for hiding information from other bidders. Researchers have also applied auction algorithms to nonfinancial settings, such as scheduling problems [Wellman *et al.*, 2001], management of resources in wide-area networks [Chen *et al.*, 2001], and co-ordination of services by different companies [Preist *et al.*, 2001].

The reader may find a detailed survey of combinatorial auctions in the review article by de Vries and Vohra [2003]. Although the developed systems can efficiently process several thousand bids, their running time is super-linear in the number of bids, and they do not scale to larger markets.

Advanced semantics. Several researchers have studied techniques for specifying dependency of item price on the number and quality of items. They have also investigated techniques for processing "flexible" bids, specified by hard and soft constraints.

Che [1993] analyzed auctions that allowed negotiating not only price but also quality of a commodity. A bid in these auctions was a function that specified a desired trade-off between price and quality. Cripps and Ireland [1994] considered a similar setting and suggested several strategies for bidding on price and quality.

Sandholm and Suri [2001b] described a mechanism for imposing nonprice constraints in combinatorial auctions, such as budget constraints and limit on the number of winners. They also studied auctions that allowed bulk discounts [Sandholm and Suri, 2001a]; that is, they enabled a bidder to specify dependency of item price on order size. Lehmann *et al.* [2001] also considered dependency of price on order size, showed that the problem of finding the best matches was NP-hard, and developed a greedy approximation algorithm.

Jones extended the semantics of combinatorial auctions and allowed buyers to use complex constraints [Jones, 2000]; for instance, a car buyer could bid on a vehicle that was less than three-years old, or on the fastest available vehicle. They suggested semantics for compact description of complex bids; however, they did not allow complex constraints in sell orders. They implemented an algorithm that found near-optimal matches, which scaled to 1,000 bids.

Bichler discussed a market that would allow negotiations on any attributes of a commodity [Bichler *et al.*, 1999; Bichler, 2000a]; however, he did not propose any computational solution. Boutilier and Hoos [2001] developed a propositional language for bids in combinatorial auctions, which allowed a compact representation of most bids. Conen and Sandholm [2002] described a system that elicited the preferences of an auction participant and helped to specify appropriate bids.

This initial work leaves many open problems, which include the use of complex constraints with general preference functions, symmetric treatment of buy and sell orders, and design of efficient matching algorithms for advanced semantics.

Exchanges. Economists have extensively studied traditional stock exchanges; for example, see the historical review by Bernstein [1993] and the textbook by Hull [1999]. They have focused on exchange dynamics rather than on efficient algorithms [Cason and Friedman, 1996; Cason and Friedman, 1999; Bapna *et al.*, 2000]. Several computer scientists have also studied trading dynamics and proposed algorithms for finding the market equilibrium [Reiter and Simon, 1992; Cheng and Wellman, 1998; Andersson and Ygge, 1998].

Auction researchers have traditionally viewed exchanges as a variety of auction markets, called continuous double auctions. Wurman *et al.* [1998a] proposed a theory of exchange markets and implemented a general-purpose system for auctions and exchanges. Sandholm and Suri [2000] developed an exchange for combinatorial orders, but it could not support markets with more than 1,000 orders. Kalagnanam *et al.* [2000] investigated techniques for placing orders with complex constraints and identifying matches between them, but the resulting system did not scale beyond a few thousand orders.

The related open problems include development of scalable systems for large combinatorial markets, as well as support for orders with complex constraints.

General-purpose systems. Computer scientists have developed several systems for auctions and exchanges, which vary from specialized markets to general-purpose tools for building new markets. The reader may find a survey of most systems in the review articles by Guttman *et al.* [1998a, 1998b] and Maes *et al.* [1999].

Kumar and Feldman [1998] built an Internet-based system that supported several standard auctions, including open-cry, single-round sealed-bid, and multiple-round auctions. Chavez and his colleagues designed an on-line agent-based auction; they built intelligent agents that negotiated on behalf of buyers and sellers [Chavez and Maes, 1996; Chavez *et al.*, 1997]. Vetter and Pitsch [1999] constructed a more flexible agent-based system that supported several types of auctions. Preist [1999a; 1999b] developed a similar distributed system for exchange markets. Bichler designed an electronic brokerage service that helped buyers and sellers to find each other and negotiate through auction mechanisms [Bichler *et al.*, 1998; Bichler and Segev, 1999].

Benyoucef *et al.* [2001] considered the problem of simultaneous negotiations for interdependent goods in multiple markets, and applied workflow management to model the negotiation process. Their system helped a bidder purchase a combinatorial package of goods in noncombinatorial markets. Boyan *et al.* [2001] also built a system for simultaneous bidding in multiple auctions; they applied beam search with simple heuristics to the problem of buying complementary goods in different auctions. Babaioff and Nisan [2001] studied the integration of multiple auctions across a supply chain, and proposed a mechanism for sharing information among such auctions.

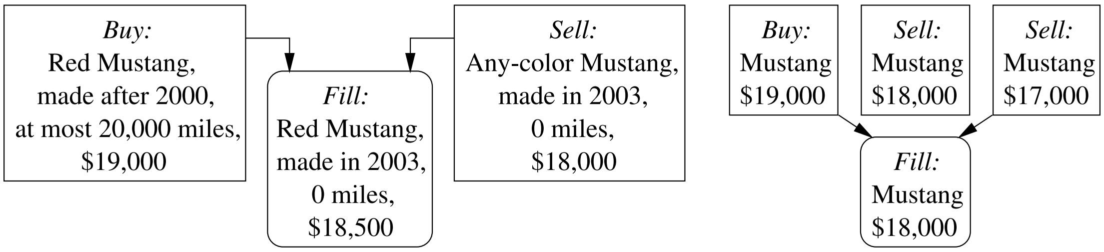
Wurman and Wellman built a general-purpose system, called the Michigan Internet AuctionBot, that supported a variety of auctions [Wellman, 1993; Wellman and Wurman, 1998; Wurman *et al.*, 1998b; Wurman and Wellman, 1999a]; however, they restricted the auction participants to simple fully specified bids. Their system included scheduler and auctioneer procedures, related databases, and advanced interfaces. Hu and his colleagues created agents for bidding in the Michigan Internet AuctionBot; they used regression and learning techniques to predict the behavior of other bidders [Hu *et al.*, 1999; Hu *et al.*, 2000; Hu and Wellman, 2001]. Wurman [2001] considered the problem of building general-purpose agents that simultaneously bid in multiple auctions.

Parkes built a system for combinatorial auctions, but it worked only for markets with up to one hundred bidders [Parkes, 1999; Parkes and Ungar, 2000a]. Sandholm [2000a; 2000b] created a more powerful system, configurable for a variety of markets, and showed its ability to process several thousand bids.

All these systems have the same limitation as commercial on-line exchanges; specifically, they require fully specified bids and do not support the use of complex constraints.

3 General exchange model

We describe a general model of trading complex commodities, which allows hard and soft constraints in the order specification.



(a) Matching orders and the resulting trade. (b) Choosing the best-price match.

Figure 1: Examples of trades in a used-car market.

Example. We begin with an example of an exchange for trading new and used cars. To simplify this example, we assume that a trader can describe a car by four attributes: model, color, year, and mileage. A prospective buyer can place a *buy order,* which includes a description of the desired vehicle and a maximal acceptable price; for instance, she may indicate that she wants a red Mustang, made after 2000, with at most 20,000 miles, and she is willing to pay $19,000. Similarly, a seller can place a *sell order;* for instance, a dealer may offer a brand-new Mustang of any color for $18,000.

An exchange system must search for matches between buy and sell orders, and generate corresponding *fills,* that is, transactions that satisfy both buyers and sellers. In the previous example, it must determine that a brand-new red Mustang for $18,500 satisfies both buyer and dealer (Figure 1a). If the system finds several matches for an order, it should choose the match with the best price; for instance, the buy order in Figure 1(b) should trade with the cheaper of the two sell orders.

Market attributes. A specific market includes a certain set of items that can be bought and sold, defined by a list of attributes. As a simplified example, we describe a car by four attributes: model, color, year, and mileage. An attribute may be a set of explicitly listed values, such as car model; an interval of integers, such as year; or an interval of real values, such as mileage. We assume that the current year is 2003, and that the oldest available car was made in 1901, which means that the years of production range from 1901 to 2003.

Cartesian products. When a trader makes a purchase or sale, she has to specify a set of acceptable values for each attribute. She specifies some set I_1 of values for the first attribute, some set I_2 of values for the second attribute, and so on. The resulting set I of acceptable items is the Cartesian product $I_1 \times I_2 \times ... \times I_n$. For example, suppose that a car buyer is looking for a Mustang or Camaro, the acceptable colors are red and white, the car should be made after 2000, and it should have at most 20,000 miles; then, the item set is $I = \{\texttt{Mustang}, \texttt{Camaro}\} \times \{\texttt{red}, \texttt{white}\} \times [2001..2003] \times [0..20{,}000]$. A trader can use specific values or ranges for each attribute; for instance, she can specify a year as 2003 or as a range from 2001 to 2003. A trader can also specify a list of several values or ranges; for example, she can specify colors as $\{\texttt{red}, \texttt{white}\}$, and years as $\{[1901..1950], [2001..2003]\}$.

Unions and filters. A trader can define an item set I as the union of several Cartesian products. For example, if she wants to buy either a used red Mustang or a new red Camaro, she can specify the set $I = \{\texttt{Mustang}\} \times \{\texttt{red}\} \times [2001..2003] \times [0..20{,}000] \cup \{\texttt{Camaro}\} \times \{\texttt{red}\} \times \{2003\} \times [0..200]$. The trader can also indicate that she wants to avoid certain items by providing a *filter function,* which is a Boolean function on the set I that gives FALSE for undesirable items. A filter is encoded by a C++ procedure that inputs an item description and returns TRUE or FALSE.

Price functions. A trader should specify a limit on the acceptable price; for instance, a buyer may be willing to pay \$18,500 for a Mustang, but only \$17,500 for a Camaro. Furthermore, she may offer an extra \$500 if a car is red, and subtract \$1 for every ten miles on its odometer. Formally, a price limit is a real-valued function defined on the set I; for each item $i \in I$, it gives a certain limit $\mathit{Price}(i)$. If a price function is a constant, it is specified by a numeric value; else, it is encoded by a C++ procedure that inputs an item and outputs the corresponding limit. For a buyer, $\mathit{Price}(i)$ is the maximal acceptable price; for a seller, it is the minimal acceptable price.

Order sizes. If a trader wants to buy or sell several identical items, she can include their number in the order specification. We assume that an order size is a natural number, thus enforcing discretization of continuous commodities. The trader can specify not only an overall order size but also a minimal acceptable size. For instance, suppose that a Toyota wholesale agent is selling one hundred cars, and she works only with dealerships that are buying at least twenty vehicles. Then, she may specify that the overall size of her order is one hundred, and the minimal size is twenty. In addition, a trader can indicate that a transaction size must be divisible by a certain number, called a *size step;* for example, the wholesale agent may specify that she is selling cars in blocks of ten. To summarize, an order includes six elements:

- Item set, $I = I1_1 \times \ldots \times I1_n \cup \ldots \cup Ik_1 \times \ldots \times Ik_n$
- Filter function, $\mathit{Filter}\colon I \to \{\text{TRUE}, \text{FALSE}\}$
- Price function, $\mathit{Price}\colon I \to \mathbf{R}$
- Overall order size, Max
- Minimal acceptable size, Min
- Size step, Step

Fills. When a buy order matches a sell order, the corresponding parties can complete a trade; we use the term *fill* to refer to the traded items and their price (Figure 1). We define a fill by a specific item i, its price p, and the number of purchased items, denoted *size*. If $(I_b, \mathit{Price}_b, \mathit{Max}_b, \mathit{Min}_b, \mathit{Step}_b)$ is a buy order, and $(I_s, \mathit{Price}_s, \mathit{Max}_s, \mathit{Min}_s, \mathit{Step}_s)$ is a matching sell order, then a fill (i, p, size) must satisfy the following conditions:

- $i \in I_b \cap I_s$.
- $\mathit{Price}_s(i) \le p \le \mathit{Price}_b(i)$.
- $\max(\mathit{Min}_b, \mathit{Min}_s) \le \mathit{size} \le \min(\mathit{Max}_b, \mathit{Max}_s)$.
- *size* is divisible by Step_b and Step_s.

FILL-PRICE$(Price_b, Price_s, i)$
The algorithm inputs the price functions of a buy and sell order, and an item i that matches both orders.

If $Price_b(i) \geq Price_s(i)$, then return $\frac{Price_b(i)+Price_s(i)}{2}$; else, return NONE (no acceptable price)

FILL-SIZE$(Max_b, Min_b, Step_b, Max_s, Min_s, Step_s)$
The algorithm inputs the size specification of a buy order, Max_b, Min_b, and $Step_b$,
and the size specification of a matching sell order, Max_s, Min_s, and $Step_s$.

Let $step$ be the least common multiple of $Step_b$ and $Step_s$
$size := \lfloor \frac{\min(Max_b, Max_s)}{step} \rfloor \cdot step$
If $size \geq \max(Min_b, Min_s)$, then return $size$; else, return NONE (no acceptable size)

Figure 2: Computing the price (FILL-PRICE) and size (FILL-SIZE) of a fill for two matching orders.

If the buyer's price limit is larger than the seller's limit, we split the difference between the buyer and seller. Furthermore, we assume that the buyer and seller are interested in trading at the maximal size, or as close to the maximal size as possible. In Figure 2, we give procedures that determine the price and size of a fill.

Quality functions. Buyers and sellers may have preferences among acceptable trades, which depend on a specific item i and its price p; for instance, a buyer may prefer a Mustang for \$18,000 to a Camaro for \$17,000. We represent preferences by a real-valued function $Qual(i, p)$, encoded by a C++ procedure, that assigns a numeric quality to each pair of an item and price. Larger values correspond to better transactions; that is, if $Qual(i_1, p_1) > Qual(i_2, p_2)$, then trading i_1 at price p_1 is better than trading i_2 at p_2. Each trader can use her own quality functions and specify different functions for different orders. Note that buyers look for low prices, whereas sellers prefer to get as much money as possible, which means that quality functions must be monotonic on price:

- *Buy monotonicity:* If $Qual_b$ is a quality function for a buy order, and $p_1 \leq p_2$, then, for every item i, we have $Qual_b(i, p_1) \geq Qual_b(i, p_2)$.
- *Sell monotonicity:* If $Qual_s$ is a quality function for a sell order, and $p_1 \leq p_2$, then, for every item i, we have $Qual_s(i, p_1) \leq Qual_s(i, p_2)$.

We do not require a trader to specify a quality function for each order; by default, the quality is the difference between the price limit and actual price, divided by the price limit:

- *Default for buy orders:* $Qual_b(i, p) = \frac{Price(i) - p}{Price(i)}$.
- *Default for sell orders:* $Qual_s(i, p) = \frac{p - Price(i)}{Price(i)}$.

Monotonic attributes. The value of a commodity may monotonically depend on some of its attributes; for example, the quality of a car decreases with an increase in mileage. When an attribute has this property, we say that it is *monotonically decreasing.* To formalize this concept, suppose that a market has n attributes, and we consider the mth attribute.

We denote attribute values of a given item by $i_1, ..., i_m, ..., i_n$, and a transaction price by p. The mth attribute is monotonically decreasing if all price and quality functions satisfy the following constraints:

- *Price monotonicity:* If *Price* is a price function for a buy or sell order, and $i_m \leq i'_m$, then, for every two items $(i_1, ..., i_{m-1}, i_m, i_{m+1}, ..., i_n)$ and $(i_1, ..., i_{m-1}, i'_m, i_{m+1}, ..., i_n)$, we have $Price(i_1, ..., i_m, ..., i_n) \geq Price(i_1, ..., i'_m, ..., i_n)$.
- *Buy monotonicity:* If $Qual_b$ is a quality function for a buy order, and $i_m \leq i'_m$, then, for every two items $(i_1, ..., i_{m-1}, i_m, i_{m+1}, ..., i_n)$ and $(i_1, ..., i_{m-1}, i'_m, i_{m+1}, ..., i_n)$, and every price p, we have $Qual_b(i_1, ..., i_m, ..., i_n, p) \geq Qual_b(i_1, ..., i'_m, ..., i_n, p)$.
- *Sell monotonicity:* If $Qual_s$ is a quality function for a sell order, and $i_m \leq i'_m$, then, for every two items $(i_1, ..., i_{m-1}, i_m, i_{m+1}, ..., i_n)$ and $(i_1, ..., i_{m-1}, i'_m, i_{m+1}, ..., i_n)$, and every price p, we have $Qual_s(i_1, ..., i_m, ..., i_n, p) \leq Qual_s(i_1, ..., i'_m, ..., i_n, p)$.

Similarly, if the quality of commodities grows with an increase in an attribute value, we say that the attribute is *monotonically increasing;* for example, the quality of a car increases with the year of production.

4 Indexing structure

The system includes a central structure for indexing of orders with fully specified items. If we can put an order into this structure, we call it an *index order.* If an order includes a set of items, rather than a fully specified item, the system adds it to an unordered list of *nonindex orders*. The indexing structure allows fast retrieval of index orders that match a given order; however, the system does not identify matches between two nonindex orders.

Main loop. In Figure 3, we show the system's main loop, which alternates between processing new orders and identifying matches for old nonindex orders. When the system receives a new order, it immediately searches for matching index orders. If there are no matches, and the new order is an index order, then the system adds it to the indexing structure. Similarly, if the system fills only part of a new index order, it stores the remaining part in the indexing structure. If it gets a nonindex order and does not find a complete fill, it adds the unfilled part to the list of nonindex orders.

After processing all new orders, the system tries to fill old nonindex orders. For each nonindex order, it identifies matching index orders. For example, suppose that the market includes an order to buy any red Mustang, and that a dealer places an order to sell a red Mustang, made in 2003, with zero miles. If the market has no matching index orders, the system adds this new order to the indexing structure. After processing all new orders, it tries to fill the nonindex orders, and determines that the dealer's order is a match for the old order to buy any red Mustang.

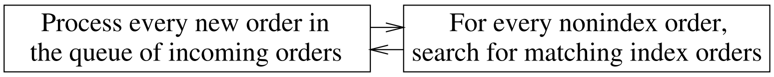


Figure 3: Main loop of the system.

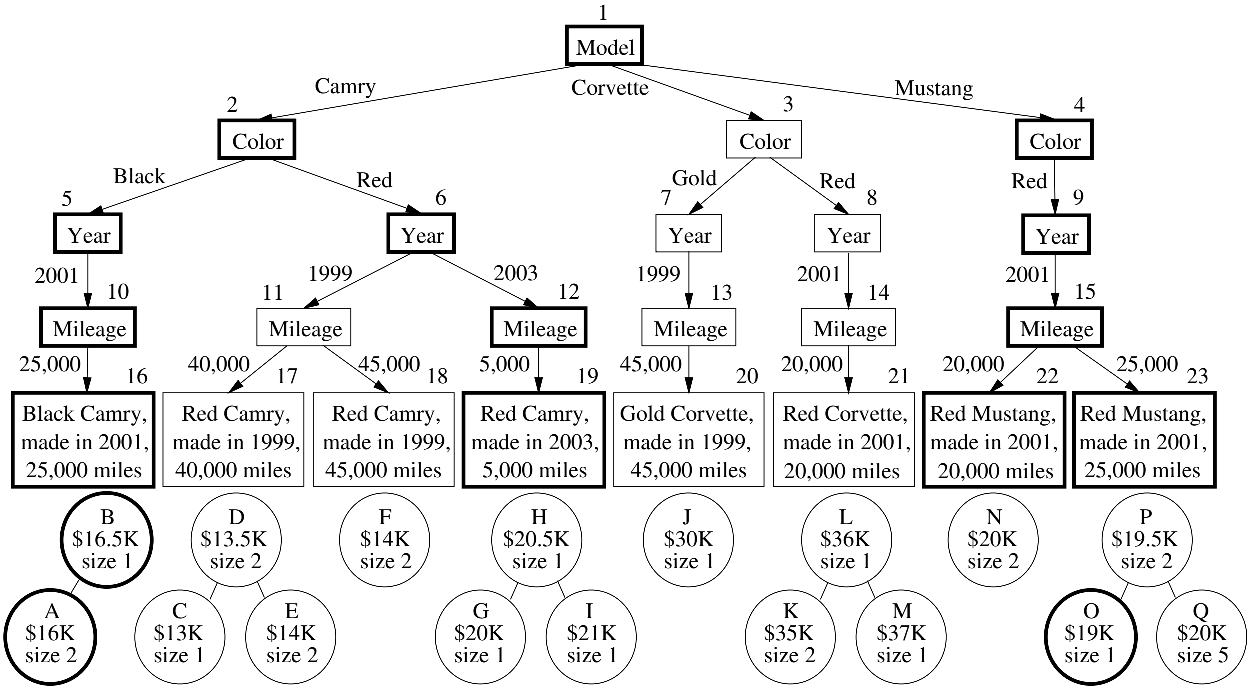


Figure 4: Indexing tree with seventeen sell orders. We illustrate the retrieval of matches for an order to buy four Camries or Mustangs made after 2000. We show the matching nodes by thick boxes, and the retrieved orders by thick circles.

Indexing trees. The indexing structure consists of two identical trees: one is for buy orders, and the other is for sell orders. In Figure 4, we show a tree for sell orders; its depth equals the number of market attributes, and each level corresponds to one of the attributes. The root node encodes the first attribute, and its children represent different values of this attribute. The nodes at depth 1 divide the orders by the second attribute, and each node at depth 2 corresponds to specific values of the first two attributes. In general, a node at depth $(i-1)$ divides orders by the values of the ith attribute, and each node at depth i corresponds to all orders with specific values of the first i attributes. If some items are not currently on market, the tree does not include the corresponding nodes.

Every nonleaf node includes a red-black tree that indexes its children by values of the corresponding attribute, which supports fast addition and deletion of a child, retrieval of a child with a given value, and identification of all children with values in a given range. Every leaf node includes orders with identical items, sorted by price from the best to the worst; that is, the system sorts buy orders from the highest to the lowest price limit, and sell orders from the lowest to the highest price. We use a red-black tree to maintain this sorting, which allows fast insertion and deletion of orders.

Summary data. The nodes of an indexing tree include summary data that help to retrieve matching orders. Every node contains the following data about the orders in the corresponding subtree:

- The minimal and maximal price of orders in the subtree.
- The minimal and maximal value for each monotonic attribute.
- The time of the latest addition of a new order.

For example, consider node 6 in Figure 4; the subtree rooted at this node includes nine orders. If the newest of them was placed at 2pm, the summary data in node 2 are as follows:

- Prices: $13,000..21,000
- Years: 1999..2003
- Mileages: 5,000..45,000
- Latest addition: 2pm

The system also keeps track of the "age" of each order, and uses it to avoid repetitive search for matches among the same index orders. Every order has two time stamps; the first is the time of placing the order, and the second is the time of the last search for matches.

Additions and deletions. When a trader places an index order, the system adds it to the corresponding leaf, and then updates the summary values of the ancestor nodes (Figure 5). If the leaf is not in the tree, the system adds the appropriate new branch.

When an index order is filled, the system removes it from the corresponding leaf, and then updates the summary values of the ancestor nodes (Figure 5). If the leaf does not include other orders, the system deletes it from the tree. If the deleted node is the only leaf in some subtree, the system removes this subtree; for example, the deletion of order J in Figure 4 leads to the removal of nodes 7, 13, and 20.

5 Search for matches

We describe two algorithms that identify matches for a given order; the first algorithm is based on depth-first search in an indexing tree, and the second is best-first search. In Figure 6, we present the notation for the order and node structures used by the algorithms. We give the depth-first algorithm in Figures 7 and 8, and the best-first algorithm in Figures 9–11.

5.1 Depth-first search

The depth-first algorithm consists of two steps; it finds the leaves of an indexing tree that match a given order (Figure 7), and selects the best matching orders in these leaves (Figure 8).

Matching leaves. The algorithm in Figure 7 retrieves the matching leaves for a given item set, represented by a union of Cartesian products and a filter function.

The PRODUCT-LEAVES subroutine finds the matching leaves for one Cartesian product using depth-first search in the indexing tree. It identifies all children of the root that match the first element of the Cartesian product, and then recursively processes the respective subtrees. For example, suppose that a buyer is looking for a Camry or Mustang made after

ADD-UPDATE(*leaf*)
The algorithm inputs the leaf that contains a newly added order.

Set *new-min* to the lowest price among *leaf*'s orders
$node := leaf$
While $node \neq$ NONE and $Min\text{-}Price[node] > new\text{-}min$:
  $Min\text{-}Price[node] := new\text{-}min$
  $node := Parent[node]$

DEL-UPDATE(*leaf*)
The algorithm inputs the leaf that contained a deleted order.

$old\text{-}min = Min\text{-}Price[leaf]$
Set $Min\text{-}Price[leaf]$ to the lowest price among *leaf*'s orders
$node := leaf$
While $Min\text{-}Price[node] > old\text{-}min$ and $Parent[node] \neq$ NONE and $Min\text{-}Price[Parent[node]] = old\text{-}min$:
  $node := Parent[node]$
  $Min\text{-}Price[node] := +\infty$
  For every *child* of *node*:
    If $Min\text{-}Price[node] > Min\text{-}Price[child]$, then $Min\text{-}Price[node] := Min\text{-}Price[child]$

Figure 5: Updating the minimal price after addition of a new order (ADD-UPDATE) and deletion of an order (DEL-UPDATE); the update of the other summary data is similar.

2000, with any color and mileage, and the tree of sell orders is as shown in Figure 4. The subroutine determines that nodes 2 and 4 match the model, and then processes the two respective subtrees. It identifies three matching nodes for the second attribute, three nodes for the third attribute, and finally four matching leaves; we show these nodes by thick boxes.

If the system already tried to find matches for a given order during the previous execution of the main loop, it skips the subtrees that have not been modified since the previous search. If the order includes a union of several Cartesian products, the system calls the PRODUCT-LEAVES subroutine for each product. If the order includes a filter function, the system uses it to prune inappropriate leaves.

If an order matches a large number of leaves, the retrieval may take considerable time. To prevent this problem, we can impose a limit on the number of retrieved leaves; for instance, if we allow at most three leaves, and a buyer places an order for any Camry, then the system retrieves the three leftmost leaves in Figure 4. We use this limit to control the trade-off between speed and quality of matches; a small limit ensures the efficiency but reduces the chances of finding the best match.

Best matches. After the system identifies matching leaves, it selects the best matching orders in these leaves, according to the quality function of the given order. In Figure 8, we give an algorithm that identifies the highest-quality matches and completes the respective trades. It arranges the leaves in a priority queue by the quality of the best unprocessed match in a leaf. At each step, the algorithm processes the best available match; it terminates after it fills the given order or runs out of matches.

Elements of the order structure:

Price[*order*]	price function
Qual[*order*]	quality function
Filter[*order*]	filter function
Max[*order*]	overall order size
Min[*order*]	minimal acceptable size
Step[*order*]	size step
Place-Time[*order*]	time of placing the order
Search-Time[*order*]	time of the last search for matches

Elements of the indexing-tree node structure:

Min-Price[*node*]	minimal price of orders in the node's subtree
Max-Price[*node*]	maximal price of orders in the node's subtree
Depth[*node*]	depth of the node in the indexing tree
Product-Num[*node*]	number of the matching Cartesian product in a given item set
Quality[*node*]	for a nonleaf node, the quality estimate; for a leaf, the quality of the best-price unprocessed order

Additional elements of the leaf-node structure:

Item[*node*]	item in the leaf's orders
Current-Order[*node*]	best-price unprocessed order in the leaf

Figure 6: Notation for the main elements of the structures that represent orders and nodes of an indexing tree. Note that the leaf-node structure includes the five elements of the node structure and two additional elements. We use this notation in the pseudocode in Figures 7–11.

MATCHING-LEAVES(*order*, *root*)

The algorithm inputs an order and the root of an indexing tree.
We denote the order's item set by $I1_1 \times ... \times I1_n \cup ... \cup Ik_1 \times ... \times Ik_n$.

Initialize an empty set of matching leaves, denoted *leaves*
For l from 1 to k, call PRODUCT-LEAVES($Il_1 \times ... \times Il_n$, *Filter*[*order*], *Search-Time*[*order*], *root*, *leaves*)
Return *leaves*

PRODUCT-LEAVES($Il_1 \times ... \times Il_n$, *Filter*, *Search-Time*, *node*, *leaves*)

The subroutine inputs a Cartesian product $Il_1 \times ... \times Il_n$, a filter function, the previous-search time, a node of the indexing tree, and a set of leaves. It finds the matching leaves in the node's subtree, and adds them to the set of leaves.

If *Search-Time* is larger than *node*'s time of the last order addition, then terminate
If *node* is a leaf and *Filter*(*Item*[*node*]) = TRUE, then add *node* to *leaves*
If *node* is not a leaf:
  Identify all children of *node* that match $I_{Depth[node]+1}$
  For each matching *child*, call PRODUCT-LEAVES($Il_1 \times ... \times Il_n$, *Filter*, *Search-Time*, *child*, *leaves*)

Figure 7: Retrieval of matching leaves. The algorithm identifies the leaves of an indexing tree that match the item set of a given order. The PRODUCT-LEAVES subroutine uses depth-first search to retrieve the matching leaves for one Cartesian product.

LEAF-MATCHES(*order*, *leaves*)
The algorithm inputs an order and matching leaves of an indexing tree.

Initialize an empty priority queue of matching leaves, denoted *queue*,
    which prioritizes the leaves by the quality of the best-price unprocessed order
For each *leaf* in *leaves*:
    Set *Current-Order*[*leaf*] to the first order among *leaf*'s orders, sorted by price
    Call LEAF-PRIORITY(*order*, *leaf*, *queue*)
While $Max[order] \geq Min[order]$ and *queue* is nonempty:
    Set *leaf* to the highest-priority leaf in *queue*, and remove it from *queue*
    $match := Current\text{-}Order[leaf]$
    Set *Current-Order*[*leaf*] to the next order among *leaf*'s orders, sorted by price
    Call TRADE(*order*, *match*)
    Call LEAF-PRIORITY(*order*, *leaf*, *queue*)
If $Max[order] < Min[order]$, then remove *order* from the market
Else, set *Search-Time*[*order*] to the current time

LEAF-PRIORITY(*order*, *leaf*, *queue*)
The subroutine inputs the given order, a matching leaf, and the priority queue of leaves. If the order's price matches the price of the leaf's best-price unprocessed order, then the leaf is added to the queue.

$match := Current\text{-}Order[leaf]$
If $match =$ NONE, then terminate (no unprocessed orders in *leaf*)
If *order* is a buy order, then $p :=$ FILL-PRICE$(Price[order], Price[match], Item[leaf])$
Else, $p :=$ FILL-PRICE$(Price[match], Price[order], Item[leaf])$
If $p =$ NONE, then terminate (no orders with acceptable price)
$Quality[leaf] := Qual[order](Item[leaf], p)$
Add *leaf* to *queue*, prioritized by *Quality*

TRADE(*order*, *match*)
The subroutine inputs the given order and the highest-quality order with matching item and price.
If the sizes of these two orders match, the subroutine completes the trade between them.

If $Search\text{-}Time[order] > Place\text{-}Time[match]$, then terminate
$size :=$ FILL-SIZE$(Max[order], Min[order], Step[order], Max[match], Min[match], Step[match])$
If $size =$ NONE, then terminate
Complete the trade between *order* and *match*
$Max[order] := Max[order] - size$
$Max[match] := Max[match] - size$
If $Max[match] < Min[match]$, then remove *match* from the market

---

Figure 8: Retrieval of matching orders. The algorithm finds the highest-quality matches for a given order and completes the corresponding trades. The LEAF-PRIORITY subroutine adds a given leaf to the priority queue, arranged by the quality of a leaf's best-price unprocessed match. The TRADE subroutine completes the trade between the given order and the best available match. The algorithm also uses the FILL-PRICE and FILL-SIZE subroutines (Figure 2).

For example, consider the tree in Figure 4, and suppose that a buyer places an order for four Camries or Mustangs made after 2000. We suppose further that she uses the default quality measure, which depends only on price. The system first retrieves order A with price \$16,000 and size 2, then order B with price \$16,500, and finally order O with price \$19,000; we show these orders by thick circles.

5.2 Best-first search

If some attributes are monotonic, we can use best-first search to find optimal matches, which is usually faster than depth-first search. The best-first algorithm uses a node's summary data to estimate the quality of matches in the node's subtree; at each step, it processes the node with the highest quality estimate.

Quality estimates. We can compute a quality estimate for a node only if all branching in the node's subtree is on monotonic attributes; a node with this property is called *monotonic*. For example, node 6 in Figure 4 is monotonic; the branching in its subtree is on year and mileage, which are monotonic attributes. On the other hand, node 2 is not monotonic because its subtree includes branching on color.

In Figure 9, we give a procedure that inputs a monotonic node and constructs the best possible item that may be present in the node's subtree, based on the summary data. To estimate the node's quality, the system computes the quality of this item traded at the best possible price from the summary data. For example, consider node 6 in Figure 4; all orders in its subtree include red Camries, and the summary data show that the best year is 2003, the best mileage is 5,000, and the best price is \$13,000. Thus, the system computes the quality estimate as $Qual(\texttt{Camry}, \texttt{red}, 2003, 5{,}000, \$13{,}000)$.

Search steps. The best-first algorithm consists of two steps, similar to the steps of the depth-first algorithm. First, it finds all smallest-depth monotonic nodes that match a given order (Figure 10); for example, if a buyer is looking for a Camry or Mustang made after 2000, and the tree of sell orders is as shown in Figure 4, then the algorithm retrieves nodes 5, 6, and 9. Second, it finds the best matching orders in the subtrees of the selected nodes (Figure 11). It arranges the nodes into a priority queue by their quality estimates; at each step, it processes the highest-quality node. If this node is a leaf, the algorithm identifies the best-price matching order in the leaf and completes the respective trade. If the node is not a leaf, the algorithm identifies its children that match the given order, and adds them to the priority queue. The algorithm terminates when it fills the given order or runs out of matches.

6 Performance

We describe experiments with artificial market data and with two real-world markets, on a 400-MHz Pentium computer with 1-Gigabyte memory. A more detailed report of the experimental results is available in Gong's [2002] masters thesis.

BEST-ITEM(*node*)
The algorithm inputs a monotonic node of an indexing tree.

For m from 1 to $Depth[node]$:
  Set i_m to the mth-attribute value on the path from the root to *node*
For m from $Depth[node] + 1$ to n:
  Set i_m to the best value of the mth attribute in *node*'s summary data
Return $(i_1, ..., i_n)$

Figure 9: Construction of the best possible item. The algorithm inputs a monotonic node and generates the best item that may be present in the subtree rooted at the node.

MATCHING-NODES(*order*, *root*)
The algorithm inputs an order and the root of an indexing tree.
We denote the order's item set by $I1_1 \times ... \times I1_n \cup ... \cup Ik_1 \times ... \times Ik_n$.

Initialize an empty set of matching monotonic nodes, denoted *nodes*
For l from 1 to k, call PRODUCT-NODES($Il_1 \times ... \times Il_n$, *Search-Time*[*order*], *root*, *nodes*)
Return *nodes*

PRODUCT-NODES($Il_1 \times ... \times Il_n$, *Search-Time*, *node*, *nodes*)
The subroutine inputs a Cartesian product $Il_1 \times ... \times Il_n$, the previous-search time, a node of the indexing tree, and a set of monotonic nodes. It finds the matching monotonic nodes in the subtree rooted at the given node, and adds them to the set of monotonic nodes.

If *Search-Time* is larger than *node*'s time of the last order addition, then terminate
If *node* is monotonic:
  $Product\text{-}Num[node] := l$
  Add *node* to *nodes*
If *node* is not monotonic:
  Identify all children of *node* that match $Il_{Depth[node]+1}$
  For each matching *child*, call PRODUCT-NODES($Il_1 \times ... \times Il_n$, *Search-Time*, *child*, *nodes*)

Figure 10: Retrieval of matching monotonic nodes. The algorithm identifies the smallest-depth monotonic nodes that match the item set of a given order. The PRODUCT-NODES subroutine uses depth-first search to retrieve the matching monotonic nodes for one Cartesian product.

NODE-MATCHES(*order*, *nodes*)
The algorithm inputs an order and matching monotonic nodes of an indexing tree.

Initialize an empty priority queue of matching nodes, denoted *queue*,
    which prioritizes the nodes by their quality estimates
For each *node* in *nodes*, call NODE-PRIORITY(*order*, *node*, *queue*)
While $Max[order] \geq Min[order]$ and *queue* is nonempty:
    Set *node* to the highest-priority node in *queue*, and remove it from *queue*
    If *node* is a leaf:
        $match := Current\text{-}Order[node]$
        Set *Current-Order*[*node*] to the next order among *node*'s orders, sorted by price
        Call TRADE(*order*, *match*)
        Call LEAF-PRIORITY(*order*, *node*, *queue*)
    If *node* is not a leaf:
        $l := Product\text{-}Num[node]$
        Identify all children of *node* that match $Il_{Depth[node]+1}$
        For each matching *child*:
            If *child* is a leaf and $Filter(Item[child]) = $ TRUE:
                Set *Current-Order*[*child*] to the first order among *child*'s orders, sorted by price
                Call LEAF-PRIORITY(*order*, *child*, *queue*)
            If *child* is not a leaf:
                $Product\text{-}Num[child] := l$
                Call NODE-PRIORITY(*order*, *child*, *queue*)
If $Max[order] < Min[order]$, then remove *order* from the market
Else, set *Search-Time*[*order*] to the current time

NODE-PRIORITY(*order*, *node*, *queue*)
The subroutine inputs the given order, a matching monotonic node, and the priority queue of nodes.
If the order may have matches in the node's subtree, then the node is added to the priority queue.

$i :=$ BEST-ITEM(*node*)
If *order* is a buy order, then $p :=$ FILL-PRICE($Price[order], Min\text{-}Price[node], i$)
Else, $p :=$ FILL-PRICE($Max\text{-}Price[node], Price[order], i$)
If $p =$ NONE, then terminate
$Quality[node] := Qual[order](i, p)$
Add *node* to *queue*, prioritized by *Quality*

Figure 11: Retrieval of matching orders. The algorithm finds the best matches for a given order and completes the corresponding trades. The NODE-PRIORITY subroutine adds a nonleaf node to the priority queue, arranged by quality estimates. The algorithm also uses four other subroutines: FILL-PRICE (Figure 2), LEAF-PRIORITY (Figure 8), TRADE (Figure 8), and BEST-ITEM (Figure 9).

6.1 Artificial markets

We have implemented an experimental setup that allows control over the number of orders, number of market attributes, number of values per attribute, and average number of matches per order. We have tested the best-first search and two versions of the depth-first search. The first version of the depth-first algorithm identifies all matching leaves, whereas the second retrieves at most ten leaves; note that the second version may not find optimal matches.

We have varied the number of orders from four to 2^{18}, that is, 262,144; we have randomly generated these orders, which include an equal number of buy and sell orders. We have considered artificial markets with one, three, and ten attributes, and we have experimented with 2, 16, and 1,024 values per attribute. Finally, we have defined the *matching density* as the mean percentage of sell orders that match a given buy order; in other words, it is the probability that a randomly selected buy order matches a randomly chosen sell order. We have experimented with four matching-density values: 0.001, 0.01, 0.1, and 1.

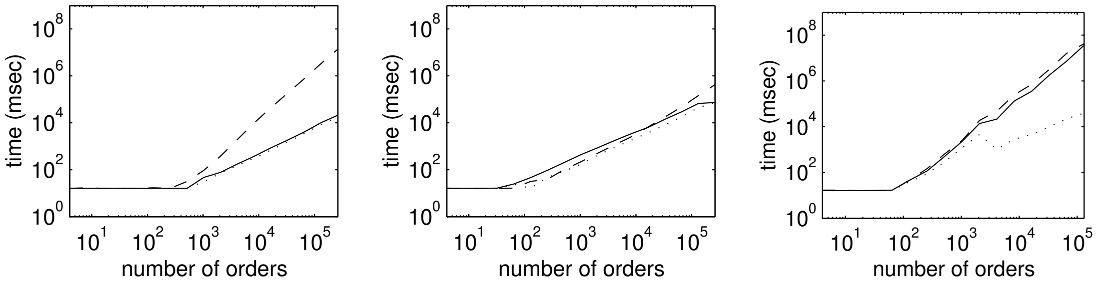
For each setting of control variables, we have measured the main-loop time and throughput. The *main-loop time* is the time of one pass through the system's main loop (Figure 3), which includes processing new orders and matching old orders. The *throughput* is the maximal acceptable rate of placing new orders; if the system gets more orders per second, the number of unprocessed orders keeps growing, and the system eventually has to reject some of them. We give the dependency of these measurements on the control variables in Figures 12 and 13; the scales of all graphs are *logarithmic.*

In Figures 12(a) and 13(a), we show how the performance changes with the number of orders. The main-loop time is approximately linear in the number of orders. The throughput in small markets grows with the number of orders; it reaches a maximum at about two hundred orders, and decreases with further increase in the number of orders.

In Figures 12(b) and 13(b), we give the dependency of the performance on the number of attributes. The main-loop time is super-linear in the number of attributes, whereas the throughput is in inverse proportion to the same super-linear function.

In Figures 12(c) and 13(c), we show how the system's behavior changes with the matching density. We have not found any monotonic dependency; the increase of the matching density sometimes leads to faster matching and sometimes slows down the system.

The best-first search is much faster than the depth-first search that identifies all matching leaves; the saving factor for large markets is between 1 and 750, and its mean value is 122. The speed of the best-first search is usually close to that of the depth-first search with a limit on the number of matching leaves. A notable exception is the performance in ten-attribute markets with a large number of values per attribute. For these markets, the best-first search is slower than the limited depth-first search by a factor of ten to hundred.

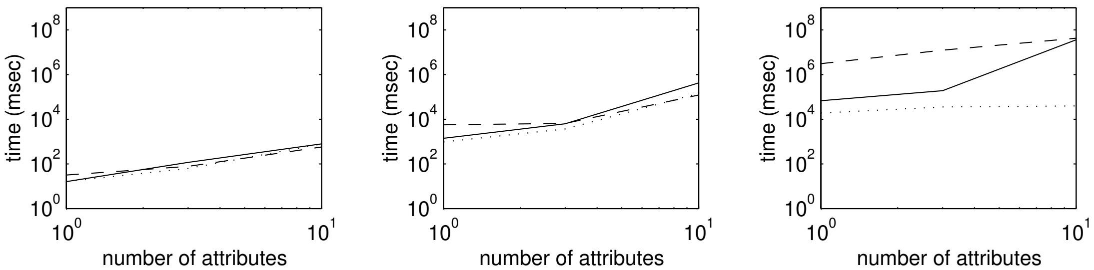


Tests with one attribute, two values per attribute, and matching density 0.001.

Tests with three attributes, sixteen values per attribute, and matching density 0.01.

Tests with ten attributes, 1,024 values per attribute, and matching density 1.

(a) Dependency of the main-loop time on the number of orders.

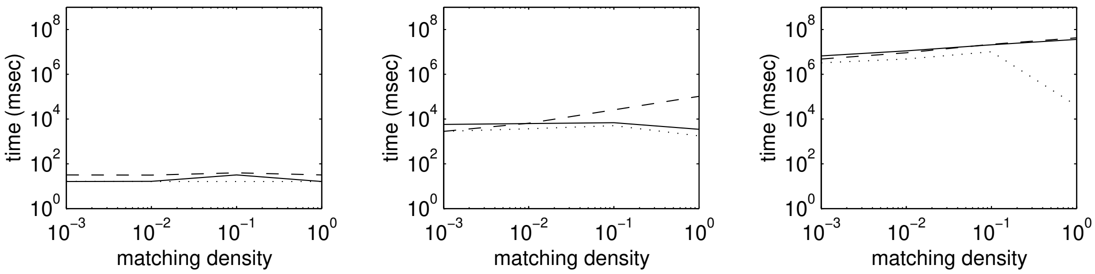


Tests with 512 orders, two values per attribute, and matching density 0.001.

Tests with 16,384 orders, sixteen values per attribute, and matching density 0.01.

Tests with 131,072 orders, 1,024 values per attribute, and matching density 1.

(b) Dependency of the main-loop time on the number of attributes.



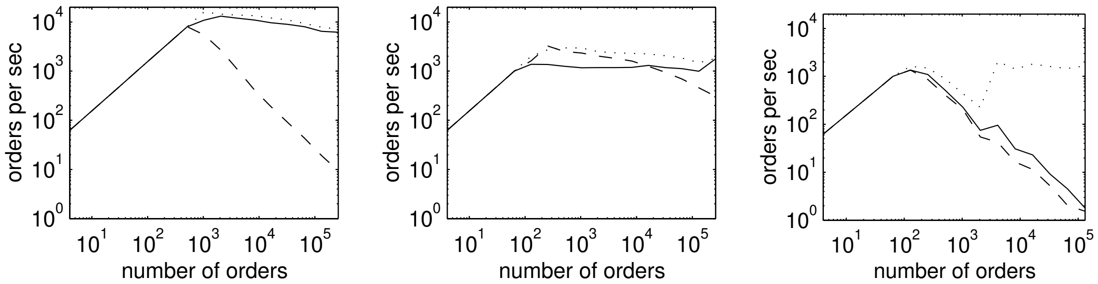
Tests with 512 orders, one attribute, and two values per attribute.

Tests with 16,384 orders, three attributes, and sixteen values per attribute.

Tests with 131,072 orders, ten attributes, and 1,024 values per attribute.

(c) Dependency of the main-loop time on the matching density.

Figure 12: Main-loop time in the artificial markets. We show the performance of the best-first search (solid lines), depth-first search that identifies all matching leaves (dashed lines), and depth-first search with a limit on the number of matching leaves (dotted lines).

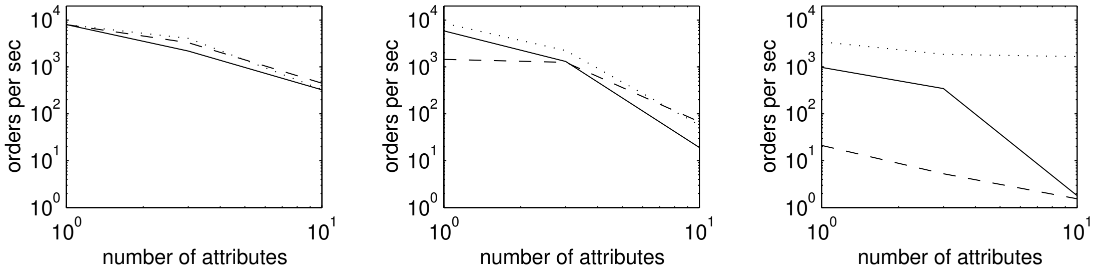


Tests with one attribute, two values per attribute, and matching density 0.001.

Tests with three attributes, sixteen values per attribute, and matching density 0.01.

Tests with ten attributes, 1,024 values per attribute, and matching density 1.

(a) Dependency of the throughput on the number of orders.

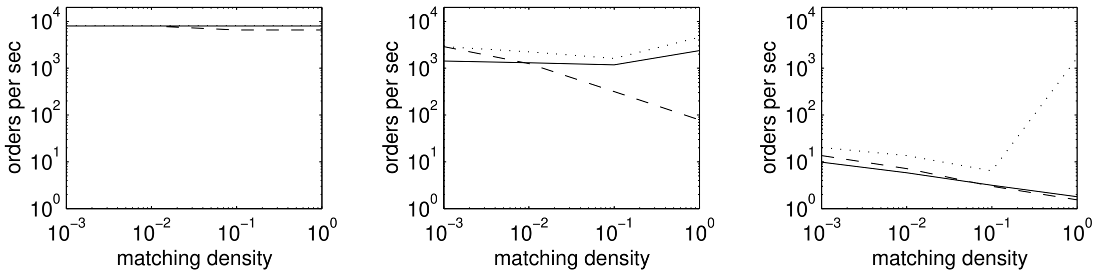


Tests with 512 orders, two values per attribute, and matching density 0.001.

Tests with 16,384 orders, sixteen values per attribute, and matching density 0.01.

Tests with 131,072 orders, 1,024 values per attribute, and matching density 1.

(b) Dependency of the throughput on the number of attributes.



Tests with 512 orders, one attribute, and two values per attribute.

Tests with 16,384 orders, three attributes, and sixteen values per attribute.

Tests with 131,072 orders, ten attributes, and 1,024 values per attribute.

(c) Dependency of the throughput on the matching density.

Figure 13: Throughput in the artificial markets; the legend is the same as in Figure 12.

6.2 Real markets

We have applied the system to an extended used-car market and to a commercial-paper market; the results are similar to that of the artificial tests.

Used cars. We have considered a used-car market that includes all models offered by AutoNation (www.autonation.com), described by eight attributes: transmission (2 values), number of doors (3 values), interior color (7 values), exterior color (52 values), model (257 values), year (103 values), option package (1,024 values), and mileage (500,000 values).

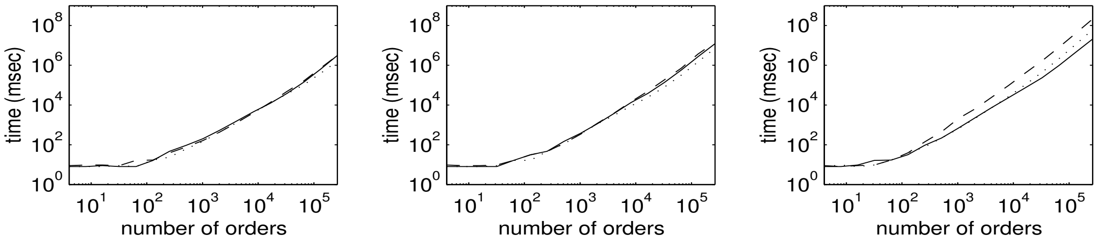
We have controlled the number of orders and matching density, and we show the results in Figures 14 and 15. The system supports markets with 300,000 orders, and processes 40 to 4,000 new orders per second. The best-first search is more efficient than the depth-first search that identifies all matching leaves; the saving factor in large markets varies from 1.0 to 8.4, with mean at 3.5. For markets with low matching density, the speed of the best-first search is close to that of the depth-first search with limit on the number of matching leaves. On the other hand, for large markets with high matching density, the best-first search is about hundred times slower than the limited depth-first search.

Commercial paper. When a large company needs a short-term loan, it may issue *commercial paper,* which is a fixed-interest "promissory note" similar to a bond. The company sells commercial paper to investors, and later returns their money with interest; the payment day is called the *maturity date.* The main difference from bonds is duration of the loan; commercial paper is issued for a short term, from one week to nine months. The appropriate interest depends on the current rate of the US Treasury bonds, company's reputation, and paper's time until maturity. After investors buy a commercial paper, they can resell it on a secondary market before the maturity date. The resale price depends on the changes in the bond rate and company's reputation.

We have described commercial paper by two attributes: company (5,000 values) and maturity date (2,550 values). We plot the dependency of the system's performance on the control variables in Figures 16 and 17. The best-first search processes 100 to 10,000 new orders per second; it outperforms the depth-first search that identifies all matching leaves by a factor of 2.3 to 8.8, with mean at 4.5. On the other hand, it is slower than the depth-first search with limit on the number of matching leaves; thus, the search for optimal matches takes more time than the suboptimal matching. This speed difference is especially significant in markets with high matching density; in particular, if the matching density is 1, the best-first search is hundred times slower than the limited depth-first search.

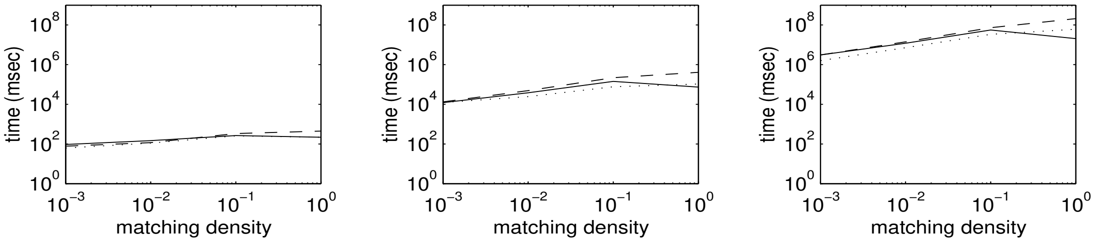
7 Concluding remarks

The reported work is a step toward the development of exchange markets for complex nonstandard goods. We have represented complex goods by multiple attributes, and allowed price and quality functions in the description of orders. We have developed two algorithms for identifying matches between buy and sell orders, which support markets with 300,000



Tests with matching density 0.001. Tests with matching density 0.01. Tests with matching density 1.

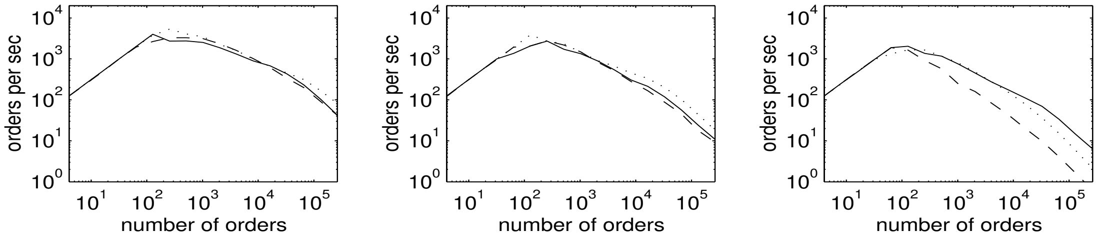
(a) Dependency of the main-loop time on the number of orders.



Tests with 512 orders. Tests with 16,384 orders. Tests with 262,144 orders.

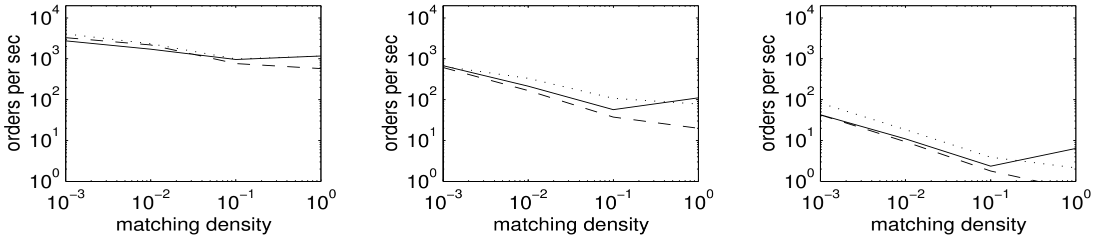
(b) Dependency of the main-loop time on the matching density.

Figure 14: Main-loop time in the used-car market. We give the results for the best-first search (solid lines), depth-first search that identifies all matching leaves (dashed lines), and depth-first search with a limit on the number of matching leaves (dotted lines).



Tests with matching density 0.001. Tests with matching density 0.01. Tests with matching density 1.

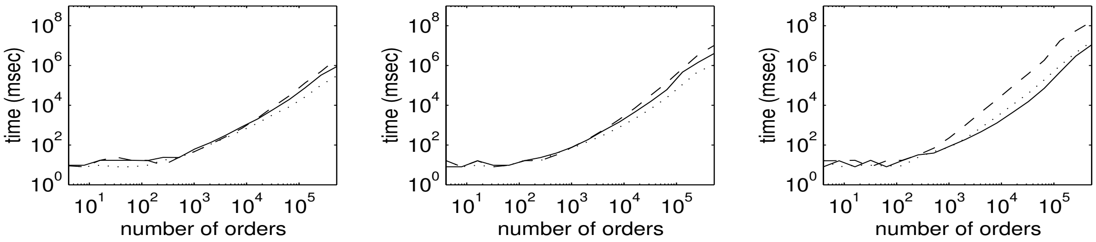
(a) Dependency of the throughput on the number of orders.



Tests with 512 orders. Tests with 16,384 orders. Tests with 262,144 orders.

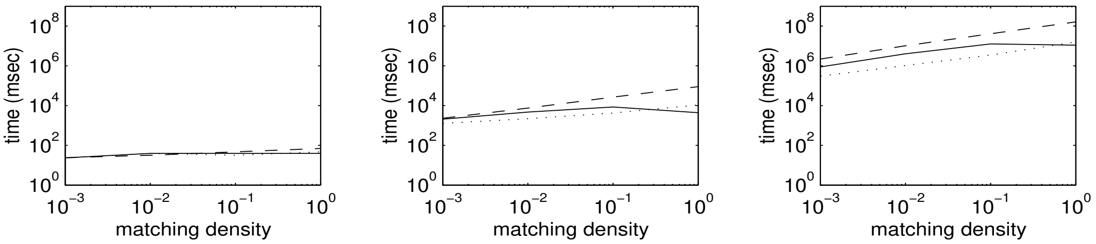
(b) Dependency of the throughput on the matching density.

Figure 15: Throughput in the used-car market; the legend is the same as in Figure 14.



Tests with matching density 0.001. | Tests with matching density 0.01. | Tests with matching density 1.

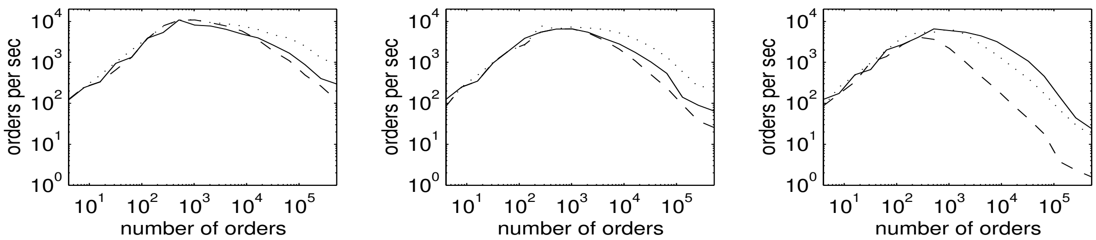
(a) Dependency of the main-loop time on the number of orders.



Tests with 512 orders. | Tests with 16,384 orders. | Tests with 262,144 orders.

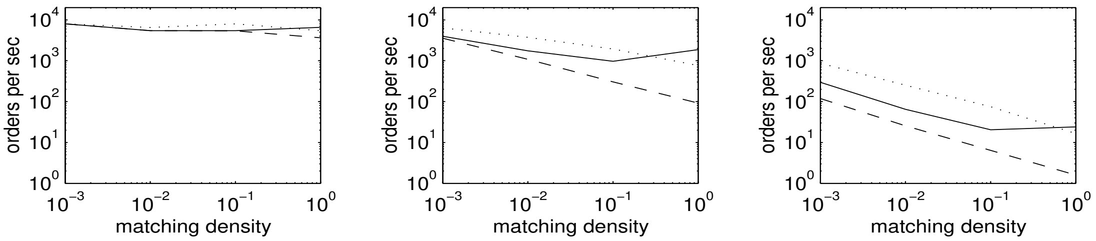
(b) Dependency of the main-loop time on the matching density.

Figure 16: Main-loop time in the commercial-paper market; the legend is the same as in Figure 14.



Tests with matching density 0.001. | Tests with matching density 0.01. | Tests with matching density 1.

(a) Dependency of the throughput on the number of orders.



Tests with 512 orders. | Tests with 16,384 orders. | Tests with 262,144 orders.

(b) Dependency of the throughput on the matching density.

Figure 17: Throughput in the commercial-paper market; the legend is the same as in Figure 14.

orders on a 400-MHz computer with 1-Gigabyte memory. The algorithms keep all orders in the main memory, and their scalability is limited by the available memory.

The first algorithm is depth-first search in an indexing tree, whereas the second is best-first search that utilizes monotonic dependency between attributes and price. When we use the depth-first algorithm, we can control the trade-off between its speed and optimality, by limiting the number of retrieved leaves of the indexing tree. If we allow suboptimal matches, the depth-first search is usually faster than the best-first search. On the other hand, the best-first search is more efficient for optimal matches.

Acknowledgments

We are grateful to Hong Tang, Jenny Hu, and John Hershberger for their help in preparing this article, and to Savvas Nikiforou for his help with software and hardware installations. We thank Lawrence Hall, Dmitry Goldgof, Sudeep Sarkar, and Srinivas Katkoori for their comments and suggestions. We also thank Ganesh Mani, Dwight Dietrich, Steve Fischetti, Michael Foster, and Alex Gurevich for their feedback and help in understanding real-world exchanges. This work has been partially sponsored by the DYNAMiX Technologies Corporation and by the National Science Foundation grant No. EIA-0130768.

References

[Akcoglu *et al.*, 2002] Karhan Akcoglu, James Aspnes, Bhaskar DasGupta, and Ming-Yang Kao. Opportunity cost algorithms for combinatorial auctions. In Erricos John Kontoghiorghes, Berç Rustem, and Stavros Siokos, editors, *Applied Optimization: Computational Methods in Decision-Making, Economics and Finance*. Kluwer Academic Publishers, Boston, MA, 2002.

[Andersson and Ygge, 1998] Arne Andersson and Fredrik Ygge. Managing large-scale computational markets. In *Proceedings of the Thirty-First Hawaii International Conference on System Sciences*, volume VII, pages 4–13, 1998.

[Andersson *et al.*, 2000] Arne Andersson, Mattias Tenhunen, and Fredrik Ygge. Integer programming for combinatorial auction winner determination. In *Proceedings of the Fourth International Conference on Multi-Agent Systems*, pages 39–46, 2000.

[Babaioff and Nisan, 2001] Moshe Babaioff and Noam Nisan. Concurrent auctions across the supply chain. In *Proceedings of the Third ACM Conference on Electronic Commerce*, pages 1–10, 2001.

[Bakos, 2001] Yannis Bakos. The emerging landscape for retail e-commerce. *Journal of Economic Perspectives*, 15(1):69–80, 2001.

[Bapna *et al.*, 2000] Ravi Bapna, Paulo Goes, and Alok Gupta. A theoretical and empirical investigation of multi-item on-line auctions. *Information Technology and Management*, 1(1):1–23, 2000.

[Benyoucef *et al.*, 2001] Morad Benyoucef, Sarita Bassil, and Rudolf K. Keller. Workflow modeling of combined negotiations in e-commerce. In *Proceedings of the Fourth International Conference on Electronic Commerce Research*, pages 348–359, 2001.

[Bernstein, 1993] Peter L. Bernstein. *Capital Ideas: The Improbable Origins of Modern Wall Street*. The Free Press, New York, NY, 1993.

[Bichler and Segev, 1999] Martin Bichler and Arie Segev. A brokerage framework for Internet commerce. *Distributed and Parallel Databases*, 7(2):133–148, 1999.

[Bichler *et al.*, 1998] Martin Bichler, Arie Segev, and Carrie Beam. An electronic broker for business-to-business electronic commerce on the Internet. *International Journal of Cooperative Information Systems*, 7(4):315–329, 1998.

[Bichler *et al.*, 1999] Martin Bichler, Marion Kaukal, and Arie Segev. Multi-attribute auctions for electronic procurement. In *Proceedings of the First* IBM IAC *Workshop on Internet-Based Negotiation Technologies*, 1999.

[Bichler, 2000a] Martin Bichler. An experimental analysis of multi-attribute auctions. *Decision Support Systems*, 29(3):249–268, 2000.

[Bichler, 2000b] Martin Bichler. A roadmap to auction-based negotiation protocols for electronic commerce. In *Proceedings of the Thirty-Third Hawaii International Conference on System Sciences*, 2000.

[Boutilier and Hoos, 2001] Craig Boutilier and Holger H. Hoos. Bidding languages for combinatorial auctions. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pages 1211–1217, 2001.

[Boyan *et al.*, 2001] Justin Boyan, Amy Greenwald, R. Mike Kirby, and Jon Reiter. Bidding algorithms for simultaneous auctions. In *Proceedings of the Third* ACM *Conference on Electronic Commerce*, pages 115–124, 2001.

[Cason and Friedman, 1996] Timothy N. Cason and Daniel Friedman. Price formation in double auction markets. *Journal of Economic Dynamics and Control*, 20:1307–1337, 1996.

[Cason and Friedman, 1999] Timothy N. Cason and Daniel Friedman. Price formation and exchange in thin markets: A laboratory comparison of institutions. In Peter Howitt, Elisabetta de Antoni, and Axel Leijonhufvud, editors, *Money, Markets, and Method: Essays in Honour of Robert W. Clower*, pages 155–179. Edward Elgar, Northampton, MA, 1999.

[Chavez and Maes, 1996] Anthony Chavez and Pattie Maes. Kasbah: An agent marketplace for buying and selling goods. In *Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology*, pages 75–90, 1996.

[Chavez *et al.*, 1997] Anthony Chavez, Daniel Dreilinger, Robert Guttman, and Pattie Maes. A real-life experiment in creating an agent marketplace. In *Proceedings of the Second International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology*, pages 159–178, 1997.

[Che, 1993] Yeon-Koo Che. Design competition through multidimensional auctions. RAND *Journal of Economics*, 24(4):668–680, 1993.

[Chen *et al.*, 2001] Chunming Chen, Muthucumaru Maheswaran, and Michel Toulouse. On bid selection heuristics for real-time auctioning for wide-area network resource management. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, 2001.

[Cheng and Wellman, 1998] John Cheng and Michael P. Wellman. The WALRAS algorithm: A convergent distributed implementation of general equilibrium outcomes. *Computational Economics*, 12(1):1–24, 1998.

[Conen and Sandholm, 2001] Wolfram Conen and Tuomas W. Sandholm. Minimal preference elicitation in combinatorial auctions. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, Workshop on Economic Agents, Models, and Mechanisms*, pages 71–80, 2001.

[Conen and Sandholm, 2002] Wolfram Conen and Tuomas W. Sandholm. Partial-revelation VCG mechanism for combinatorial auctions. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, pages 367–372, 2002.

[Cripps and Ireland, 1994] Martin Cripps and Norman Ireland. The design of auctions and tenders with quality thresholds: The symmetric case. *Economic Journal*, 104(423):316–326, 1994.

[de Vries and Vohra, 2003] Sven de Vries and Rakesh V. Vohra. Combinatorial auctions: A survey. INFORMS *Journal of Computing*, 15(3):284–309, 2003.

[Feldman, 2000] Stuart I. Feldman. Electronic marketplaces. IEEE *Internet Computing*, 4(4):93–95, 2000.

[Fink *et al.*, 2004] Eugene Fink, Joshua Marc Johnson, and Jenny Hu. Exchange Market for Complex Goods: Theory and Experiments. *Netnomics*, 6(1):21–42, 2004.

[Fujishima *et al.*, 1999a] Yuzo Fujishima, Kevin Leyton-Brown, and Yoav Shoham. Speeding up ascending-bid auctions. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, volume 1, pages 554–563, 1999.

[Fujishima *et al.*, 1999b] Yuzo Fujishima, Kevin Leyton-Brown, and Yoav Shoham. Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, volume 1, pages 548–553, 1999.

[Gonen and Lehmann, 2000] Rica Gonen and Daniel J. Lehmann. Optimal solutions for multi-unit combinatorial auctions: Branch and bound heuristics. In *Proceedings of the Second* ACM *Conference on Electronic Commerce*, pages 13–20, 2000.

[Gonen and Lehmann, 2001] Rica Gonen and Daniel J. Lehmann. Linear programming helps solving large multi-unit combinatorial auctions. In *Proceedings of the Electronic Market Design Workshop*, 2001.

[Gong, 2002] Jianli Gong. Exchanges for complex commodities: Search for optimal matches. Master's thesis, Department of Computer Science and Engineering, University of South Florida, 2002.

[Guttman *et al.*, 1998a] Robert H. Guttman, Alexandros G. Moukas, and Pattie Maes. Agent-mediated electronic commerce: A survey. *Knowledge Engineering Review*, 13(3):147–161, 1998.

[Guttman *et al.*, 1998b] Robert H. Guttman, Alexandros G. Moukas, and Pattie Maes. Agents as mediators in electronic commerce. *International Journal of Electronic Markets*, 8(1):22–27, 1998.

[Holte, 2001] Robert C. Holte. Combinatorial auctions, knapsack problems, and hill-climbing search. In *Proceedings of the Canadian Conference on Artificial Intelligence*, pages 57–66, 2001.

[Hoos and Boutilier, 2000] Holger H. Hoos and Craig Boutilier. Solving combinatorial auctions using stochastic local search. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence*, pages 22–29, 2000.

[Hu and Wellman, 2001] Junling Hu and Michael P. Wellman. Learning about other agents in a dynamic multiagent system. *Journal of Cognitive Systems Research*, 1:67–79, 2001.

[Hu *et al.*, 1999] Junling Hu, Daniel Reeves, and Hock-Shan Wong. Agents participating in Internet auctions. In *Proceedings of the* AAAI *Workshop on Artificial Intelligence for Electronic Commerce*, 1999.

[Hu *et al.*, 2000] Junling Hu, Daniel Reeves, and Hock-Shan Wong. Personalized bidding agents for online auctions. In *Proceedings of the Fifth International Conference on the Practical Application of Intelligent Agents and Multi-Agents*, pages 167–184, 2000.

[Hu, 2002] Jenny Ying Hu. Exchanges for complex commodities: Representation and indexing of orders. Master's thesis, Department of Computer Science and Engineering, University of South Florida, 2002.

[Hull, 1999] John C. Hull. *Options, Futures, and Other Derivatives.* Prentice Hall, Upper Saddle River, NJ, fourth edition, 1999.

[Johnson, 2001] Joshua Marc Johnson. Exchanges for complex commodities: Theory and experiments. Master's thesis, Department of Computer Science and Engineering, University of South Florida, 2001.

[Jones, 2000] Joni L. Jones. *Incompletely Specified Combinatorial Auction: An Alternative Allocation Mechanism for Business-to-Business Negotiations.* PhD thesis, Warrington College of Business, University of Florida, 2000.

[Kalagnanam *et al.*, 2000] Jayant R. Kalagnanam, Andrew J. Davenport, and Ho S. Lee. Computational aspects of clearing continuous call double auctions with assignment constraints and indivisible demand. Technical Report RC21660(97613), IBM, 2000.

[Kelly and Steinberg, 2000] Frank Kelly and Richard Steinberg. A combinatorial auction with multiple winners for universal service. *Management Science*, 46:586–596, 2000.

[Klein, 1997] Stefan Klein. Introduction to electronic auctions. *International Journal of Electronic Markets*, 7(4):3–6, 1997.

[Kumar and Feldman, 1998] Manoj Kumar and Stuart I. Feldman. Internet auctions. In *Proceedings of the Third* USENIX *Workshop on Electronic Commerce*, pages 49–60, 1998.

[Lavi and Nisan, 2000] Ran Lavi and Noam Nisan. Competitive analysis of incentive compatible on-line auctions. In *Proceedings of the Second* ACM *Conference on Electronic Commerce*, pages 233–241, 2000.

[Lehmann *et al.*, 1999] Daniel J. Lehmann, Liadan Ita O'Callaghan, and Yoav Shoham. Truth revelation in rapid, approximately efficient combinatorial auctions. In *Proceedings of the First* ACM *Conference on Electronic Commerce*, pages 96–102, 1999.

[Lehmann *et al.*, 2001] Benny Lehmann, Daniel J. Lehmann, and Noam Nisan. Combinatorial auctions with decreasing marginal utilities. In *Proceedings of the Third* ACM *Conference on Electronic Commerce*, pages 18–28, 2001.

[Leyton-Brown *et al.*, 2000a] Kevin Leyton-Brown, Mark Pearson, and Yoav Shoham. Towards a universal test suite for combinatorial auction algorithms. In *Proceedings of the Second* ACM *Conference on Electronic Commerce*, pages 66–76, 2000.

[Leyton-Brown *et al.*, 2000b] Kevin Leyton-Brown, Yoav Shoham, and Moshe Tennenholtz. An algorithm for multi-unit combinatorial auctions. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence*, pages 56–61, 2000.

[Leyton-Brown *et al.*, 2000c] Kevin Leyton-Brown, Moshe Tennenholtz, and Yoav Shoham. Bidding clubs: Institutionalized collusion in auctions. In *Proceedings of the Second* ACM *Conference on Electronic Commerce*, pages 253–259, 2000.

[Maes *et al.*, 1999] Pattie Maes, Robert H. Guttman, and Alexandros G. Moukas. Agents that buy and sell: Transforming commerce as we know it. *Communications of the* ACM, 42(3):81–91, 1999.

[Monderer and Tennenholtz, 2000] Dov Monderer and Moshe Tennenholtz. Optimal auctions revisited. *Artificial Intelligence*, 120(1):29–42, 2000.

[Mu'alem and Nisan, 2002] Ahuva Mu'alem and Noam Nisan. Truthful approximation mechanisms for restricted combinatorial auctions. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, pages 379–384, 2002.

[Nisan, 2000] Noam Nisan. Bidding and allocation in combinatorial auctions. In *Proceedings of the Second* ACM *Conference on Electronic Commerce*, pages 1–12, 2000.

[Noussair *et al.*, 1998] Charles Noussair, Stephane Robin, and Bernard Ruffieux. The effect of transaction costs on double auction markets. *Journal of Economic Behavior and Organization*, 36:221–233, 1998.

[Parkes and Ungar, 2000a] David C. Parkes and Lyle H. Ungar. Iterative combinatorial auctions: Theory and practice. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence*, pages 74–81, 2000.

[Parkes and Ungar, 2000b] David C. Parkes and Lyle H. Ungar. Preventing strategic manipulation in iterative auctions: Proxy agents and price-adjustment. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence*, pages 82–89, 2000.

[Parkes *et al.*, 1999] David C. Parkes, Lyle H. Ungar, and Dean P. Foster. Accounting for cognitive costs in on-line auction design. In Pablo Noriega and Carles Sierra, editors, *Agent Mediated Electronic Commerce*, pages 25–40. Springer-Verlag, New York, NY, 1999.

[Parkes, 1999] David C. Parkes. *i*Bundle: An efficient ascending price bundle auction. In *Proceedings of the First* ACM *Conference on Electronic Commerce*, pages 148–157, 1999.

[Preist *et al.*, 2001] Chris Preist, Andrew Byde, Claudio Bartolini, and Giacomo Piccinelli. Towards agent-based service composition through negotiation in multiple auctions. Technical Report HPL-2001-71, Hewlett Packard, 2001.

[Preist, 1999a] Chris Preist. Commodity trading using an agent-based iterated double auction. In *Proceedings of the Third Annual Conference on Autonomous Agents*, pages 131–138, 1999.

[Preist, 1999b] Chris Preist. Economic agents for automated trading. In Alex L. G. Hayzelden and John Bigham, editors, *Software Agents for Future Communication Systems*, pages 207–220. Springer-Verlag, Berlin, Germany, 1999.

[Reiter and Simon, 1992] Stanley Reiter and Carl Simon. Decentralized dynamic processes for finding equilibrium. *Journal of Economic Theory*, 56:400–445, 1992.

[Ronen, 2001] Amir Ronen. On approximating optimal auctions. In *Proceedings of the Third* ACM *Conference on Electronic Commerce*, pages 11–17, 2001.

[Rothkopf *et al.*, 1998] Michael H. Rothkopf, Aleksandar Pekeč, and Ronald M. Harstad. Computationally manageable combinatorial auctions. *Management Science*, 44(8):1131–1147, 1998.

[Sakurai *et al.*, 2000] Yuko Sakurai, Makoto Yokoo, and Koji Kamei. An efficient approximate algorithm for winner determination in combinatorial auctions. In *Proceedings of the Second* ACM *Conference on Electronic Commerce*, pages 30–37, 2000.

[Sandholm and Ferrandon, 2000] Tuomas W. Sandholm and Vincent Ferrandon. Safe exchange planner. In *Proceedings of the International Conference on Multi-Agent Systems*, pages 255–262, 2000.

[Sandholm and Suri, 2000] Tuomas W. Sandholm and Subhash Suri. Improved algorithms for optimal winner determination in combinatorial auctions and generalizations. In *Proceesings of the Seventeenth National Conference on Artificial Intelligence*, pages 90–97, 2000.

[Sandholm and Suri, 2001a] Tuomas W. Sandholm and Subhash Suri. Market clearability. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pages 1145–1151, 2001.

[Sandholm and Suri, 2001b] Tuomas W. Sandholm and Subhash Suri. Side constraints and non-price attributes in markets. In *Proceedings of the International Joint Conference on Artificial Intelligece, Workshop on Distributed Constraint Reasoning*, 2001.

[Sandholm *et al.*, 2001a] Tuomas W. Sandholm, Subhash Suri, Andrew Gilpin, and David Levine. CABOB: A fast optimal algorithm for combinatorial auctions. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pages 1102–1108, 2001.

[Sandholm *et al.*, 2001b] Tuomas W. Sandholm, Subhash Suri, Andrew Gilpin, and David Levine. Winner determination in combinatorial auction generalizations. In *Proceedings of the International Conference on Autonomous Agents, Workshop on Agent-Based Approaches to* B2B, pages 35–41, 2001.

[Sandholm, 1999] Tuomas W. Sandholm. An algorithm for optimal winner determination in combinatorial auctions. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 542–547, 1999.

[Sandholm, 2000a] Tuomas W. Sandholm. Approach to winner determination in combinatorial auctions. *Decision Support Systems*, 28(1–2):165–176, 2000.

[Sandholm, 2000b] Tuomas W. Sandholm. eMediator: A next generation electronic commerce server. In *Proceedings of the Fourth International Conference on Autonomous Agents*, pages 73–96, 2000.

[Suzuki and Yokoo, 2002] Koutarou Suzuki and Makoto Yokoo. Secure combinatorial auctions by dynamic programming with polynomial secret sharing. In *Proceedings of the Sixth International Financial Cryptography Conference*, pages 44–56, 2002.

[Tesauro and Das, 2001] Gerald Tesauro and Rajarshi Das. High-performance bidding agents for the continuous double auction. In *Proceedings of the Third* ACM *Conference on Electronic Commerce*, pages 206–209, 2001.

[Turban, 1997] Efraim Turban. Auctions and bidding on the Internet: An assessment. *International Journal of Electronic Markets*, 7(4):7–11, 1997.

[Vetter and Pitsch, 1999] Michael Vetter and Stefan Pitsch. An agent-based market supporting multiple auction protocols. In *Proceedings of the Workshop on Agents for Electronic Commerce and Managing the Internet-Enabled Supply Chain*, 1999.

[Wellman and Wurman, 1998] Michael P. Wellman and Peter R. Wurman. Real-time issues for Internet auctions. In *Proceedings of the First* IEEE *Workshop on Dependable and Real-Time E-Commerce Systems*, 1998.

[Wellman *et al.*, 2001] Michael P. Wellman, William E. Walsh, Peter R. Wurman, and Jeffrey K. MacKie-Mason. Auction protocols for decentralized scheduling. *Games and Economic Behavior*, 35:271–303, 2001.

[Wellman, 1993] Michael P. Wellman. A market-oriented programming environment and its application to distributed multicommodity flow problems. *Journal of Artificial Intelligence Research*, 1:1–23, 1993.

[Wrigley, 1997] Clive D. Wrigley. Design criteria for electronic market servers. *International Journal of Electronic Markets*, 7(4):12–16, 1997.

[Wurman and Wellman, 1999a] Peter R. Wurman and Michael P. Wellman. Control architecture for a flexible Internet auction server. In *Proceedings of the First* IAC *Workshop on Internet Based Negotiation Technologies*, 1999.

[Wurman and Wellman, 1999b] Peter R. Wurman and Michael P. Wellman. Equilibrium prices in bundle auctions. In *Proceedings of the* AAAI *Workshop on Artificial Intelligence for Electronic Commerce*, 1999.

[Wurman and Wellman, 2000] Peter R. Wurman and Michael P. Wellman. AkBA: A progressive, anonymous-price combinatorial auction. In *Proceedings of the Second* ACM *Conference on Electronic Commerce*, pages 21–29, 2000.

[Wurman *et al.*, 1998a] Peter R. Wurman, William E. Walsh, and Michael P. Wellman. Flexible double auctions for electronic commerce: Theory and implementation. *Decision Support Systems*, 24(1):17–27, 1998.

[Wurman *et al.*, 1998b] Peter R. Wurman, Michael P. Wellman, and William E. Walsh. The Michigan Internet AuctionBot: A configurable auction server for human and software agents. In *Proceedings of the Second International Conference on Autonomous Agents*, pages 301–308, 1998.

[Wurman *et al.*, 2001] Peter R. Wurman, Michael P. Wellman, and William E. Walsh. A parametrization of the auction design space. *Games and Economic Behavior*, 35(1–2):304–338, 2001.

[Wurman, 2001] Peter R. Wurman. Toward flexible trading agents. In *Proceedings of the* AAAI *Spring Symposium on Game Theoretic and Decision Theoretic Agents*, pages 134–140, 2001.

[Ygge and Akkermans, 1997] Fredrik Ygge and Hans Akkermans. Duality in multi-commodity market computations. In *Proceedings of the Third Australian Workshop on Distributed Artificial Intelligence*, pages 65–78, 1997.

[Yokoo *et al.*, 2001a] Makoto Yokoo, Yuko Sakurai, and Shigeo Matsubara. Bundle design in robust combinatorial auction protocol against false-name bids. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pages 1095–1101, 2001.

[Yokoo *et al.*, 2001b] Makoto Yokoo, Yuko Sakurai, and Shigeo Matsubara. Robust combinatorial auction protocol against false-name bids. *Artificial Intelligence*, 130(2):167–181, 2001.

[Zurel and Nisan, 2001] Edo Zurel and Noam Nisan. An efficient approximate allocation algorothm for combinatorial auctions. In *Proceedings of the Third* ACM *Conference on Electronic Commerce*, pages 125–136, 2001.