Multi-Attribute Exchange Market: Theory and Experiments

Eugene Fink, Josh Johnson, and John Hershberger
eugene@csee.usf.edu, joshjohnson@cfl.rr.com, jhershbe@csee.usf.edu
Computer Science, University of South Florida, Tampa, Florida 33620, USA

Abstract. The Internet has opened opportunities for efficient on-line trading, and researchers have developed algorithms for various auctions, as well as exchanges for standardized commodities; however, they have done little work on exchanges for complex nonstandard goods. We propose a formal model for trading complex goods, present an exchange system that allows traders to describe purchases and sales by multiple attributes, and give the results of applying it to a used-car market and corporate-bond market.

1 Introduction

The growth of the Internet has led to the development of on-line markets, which include bulletin boards, auctions, and exchanges. Bulletin boards help buyers and sellers find each other, but they often require customers to invest significant time into reading multiple ads, and many buyers prefer on-line auctions, such as eBay (www.ebay.com). Auctions have their own problems, including high computational costs, lack of liquidity, and asymmetry between buyers and sellers. Exchange markets support fast-paced trading and ensure symmetry between buyers and sellers, but they require rigid standardization of tradable items. For example, the New York Stock Exchange allows trading of about 3,000 stocks, and a buyer or seller has to indicate a specific stock. For most goods, the description of a desirable trade is more complex. An exchange for nonstandard goods should allow the use of multiple attributes in specifications of buy and sell orders.

Economists and computer scientists have long realized the importance of auctions and exchanges, and studied a variety of trading models. The related computer science research 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.org). Recently, researchers have developed efficient systems for combinatorial auctions, which allow buying and selling sets of commodities rather than individual items [1, 2, 7–10]. Computer scientists have also studied exchange markets; in particular, Wurman, Walsh, and Wellman built a general-purpose system for auctions and exchanges [11], Sandholm and Suri developed an exchange for combinatorial orders [9], and Kalagnanam, Davenport, and Lee investigated techniques for placing orders with complex constraints [6].

A recent project at the University of South Florida has been aimed at building an automated exchange for complex goods [3–5]. We have developed a system that supports large-scale exchanges for commodities described by multiple attributes. We give a formal model of a multi-attribute exchange (Sections 2 and 3), describe the developed system (Section 4), and show how its performance depends on the market size (Section 5).

2 General Exchange Model

We begin with an example of a multi-attribute market, and then define orders and matches between them.

Example. We consider 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 a desired car and a maximal acceptable price; for instance, she may indicate that she wants a red Mustang, made after 2000, with less than 20,000 miles, and she is willing to pay $19,000. Similarly, a seller can place a *sell order;* for example, a dealer may offer a brand-new Mustang of any color for $18,000. An exchange system must generate trades that satisfy both buyers and sellers; in the previous example, it must determine that a brand-new red Mustang for $18,500 satisfies the buyer and dealer.

Orders. When a trader makes a purchase or sale, she has to specify a set of acceptable items, denoted I, which stands for *item set.* In addition, a trader should specify a limit on the acceptable price, which is a real-valued function on the set I; for each item $i \in I$, it gives a certain limit $Price(i)$. For a buyer, $Price(i)$ is the maximal acceptable price; for a seller, it is the minimal acceptable price. If a trader wants to buy or sell several identical items, she can include their number in the order specification, which is called an *order size.* She can specify not only an overall order size, but also a minimal acceptable size. For instance, suppose that a Ford wholesale agent is selling one hundred cars, and she works only with dealerships that are buying at least ten cars. Then, she may specify that the overall size of her order is one hundred, and the minimal size is ten.

Fills. An order specification includes an item set I, price function $Price$, overall order size Max, and minimal acceptable size Min. 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. We define a fill by a specific item i, its price p, and the number of purchased items, denoted *size.* If $(I_b, Price_b, Max_b, Min_b)$ is a buy order, and $(I_s, Price_s, Max_s, Min_s)$ is a matching sell order, then a fill must satisfy the following conditions:

1. $i \in I_b \cap I_s$.
2. $Price_s(i) \leq p \leq Price_b(i)$.
3. $\max(Min_b, Min_s) \leq size \leq \min(Max_b, Max_s)$.

3 Order Representation

We next describe the representation of orders in the developed exchange system.

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 the car model; an interval of integers, such as the year; or an interval of real values, such as the mileage.

Cartesian products. When a trader places an order, she has to specify some set I_1 of acceptable values for the first attribute, some set I_2 for the second attribute, and so on. The resulting set I of acceptable items is the Cartesian product $I_1 \times I_2 \times \ldots$. For example, suppose that a car buyer is looking for a Mustang or Camaro, the acceptable colors are red or 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 desired year as 2003 or as a range from 2001 to 2003. She can also specify a list of several values or ranges; for example, she can specify a set of colors as {`red`, `white`}, and a set of years as {[1900..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 $I = (\{\texttt{Mustang}\} \times \{\texttt{red}\} \times [2001..2003] \times [0..20{,}000]) \cup (\{\texttt{Camaro}\} \times \{\texttt{red}\} \times \{2003\} \times [0..200])$. Furthermore, the trader can indicate that she wants to avoid certain items; for instance, a superstitious buyer may want to avoid black cars with 13 miles on the odometer. In this case, the trader must use a *filter function* that prunes undesirable items. This filter is a Boolean function on the set I, encoded by a C++ procedure, which gives FALSE for unwanted items.

Orders. An order includes an item set, defined by a union of Cartesian products and optional filter function, along with a price function and size. If the price function is a constant, it is specified by a numeric value; else, it is a C++ procedure that inputs an item and outputs the corresponding price limit. The size specification includes two positive values: overall size and minimal acceptable size.

4 Exchange System

The system consists of a central matcher and multiple user interfaces that run on separate machines. The traders enter orders through interface machines, which send the orders to the matcher. The system supports three types of messages to the matcher: placing, modifying, and cancelling an order.

The matcher 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 matcher 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.

In Fig. 1, we show the main loop of the matcher, which alternates between processing new messages and identifying matches for old orders. When it receives a message with a new order, it immediately identifies 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

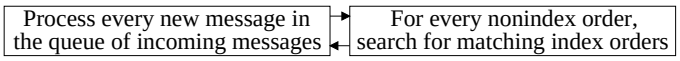

Fig. 1. Main loop of the matcher.

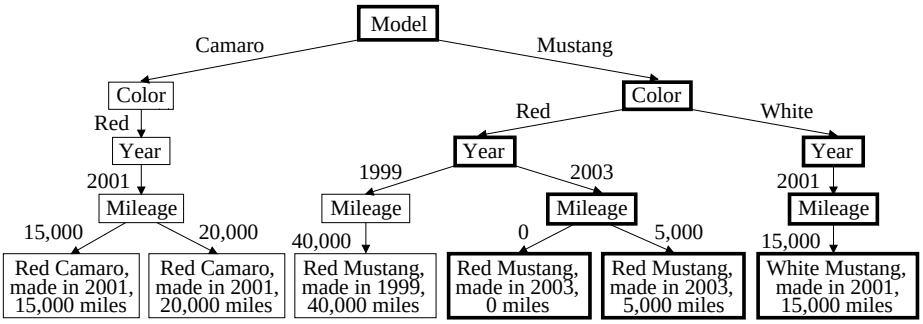

Fig. 2. Indexing tree for a used-car market. Thick boxes show the retrieval of matches for an order to buy a Mustang made after 2000, with any color and mileage.

order and does not find a complete fill, it adds the unfilled part to the list of nonindex orders.

When the system gets a cancellation message, it removes the specified order from the market. When it receives a modification message, it makes changes to the specified order. If the changes can potentially lead to new matches, it immediately searches for index orders that match the modified order. For example, if a seller reduces the price of her order, the system immediately identifies new matches. On the other hand, if the seller increases her price, the system does not search for matches.

After processing all messages, 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 a new 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 messages, 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.

The indexing structure consists of two identical trees: one is for buy orders, and the other is for sell orders. The height of an indexing tree equals the number of attributes, and each level corresponds to one of the attributes (Fig. 2). The root node encodes the first attribute, and its children represent different values of this attribute. The nodes at the second level divide the orders by the second attribute, and each node at the third level corresponds to specific values of the first two attributes. In general, a node at level i divides orders by the values of the ith attribute, and each node at level $(i + 1)$ corresponds to all orders with specific values of the first i attributes. Every leaf node includes orders with identical items, sorted by price.

To find matches for a given order, the system identifies all children of the root that match the first attribute of the order's item set, and then recursively processes the respective subtrees. For example, suppose that a buyer is looking

for a Mustang made after 2000, with any color and mileage, and the tree of sell orders is as shown in Fig. 2. The system identifies one matching node for the first attribute, two nodes for the second attribute, two nodes for the third attribute, and finally three matching leaves; we show these nodes by thick boxes. If the order includes the union of several Cartesian products, the system finds matches separately for each product. If the order includes a filter function, the system uses the filter to prune inappropriate leaves. After identifying the matching leaves, the system selects the best-price orders in these leaves.

5 Performance

We describe experiments with an extended used-car market and corporate-bond market. We have run the system on a 2-GHz Pentium computer with one-gigabyte memory. A more detailed report of the experimental results is available in Johnson's masters thesis [5].

The used-car market includes all car models available through AutoNation (www.autonation.com), described by eight attributes: transmission (2 values), number of doors (3 values), interior color (7 values), exterior color (52 values), year (103 values), model (257 values), option package (1,024 values), and mileage (500,000 values). The corporate-bond market is described by two attributes: issuing company (5,000 values) and maturity date (2,550 values).

We have varied the number of old orders in the market from one to 300,000, which is the maximum possible number for one-gigabyte memory. We have also controlled the number of incoming new orders in the beginning of the system's main loop (Fig. 1); we have experimented with 300 and 10,000 new orders. In addition, we have controlled the *matching density,* defined 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 considered five matching-density values: 0.0001, 0.001, 0.01, 0.1, and 1.

For each setting of the control variables, we have measured the main-loop time, throughput, and response time. The *main-loop time* is the time of one pass through the system's main loop (Fig. 1). The *throughput* is the maximal acceptable rate of placing new orders; if the system gets more orders per second, it has to reject some of them. Finally, the *response time* is the average time between placing an order and getting a fill.

In Figs. 3 and 4, we show how the performance changes with the number of old orders in the market; note that the scales of all graphs are *logarithmic*. The main-loop and response times are linear in the number of orders. The throughput in small markets grows with the number of orders; it reaches a maximum at about three hundred orders, and slightly decreases with further increase in the market size. The system processes 500 to 5,000 orders per second in the used-car market, and 2,000 to 20,000 orders per second in the corporate-bond market. In Figs. 5 and 6, we show that the main-loop and response times grow linearly with the matching density. On the other hand, we have not found any monotonic dependency between the matching density and the throughput.

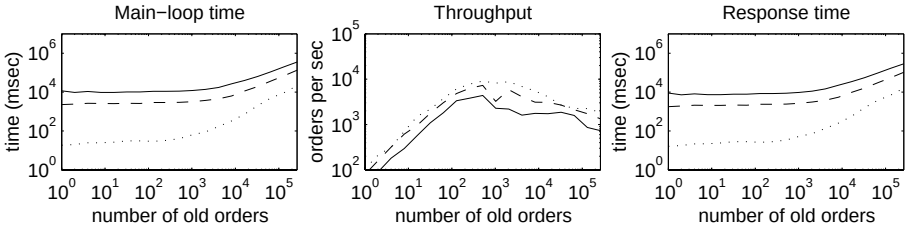

Fig. 3. Dependency of the performance on the number of old orders in the used-car market. The dotted lines show experiments with 300 new orders and matching density of 0.0001. The dashed lines are for 10,000 new orders and matching density of 0.001. The solid lines are for 10,000 new orders and matching density of 0.01.

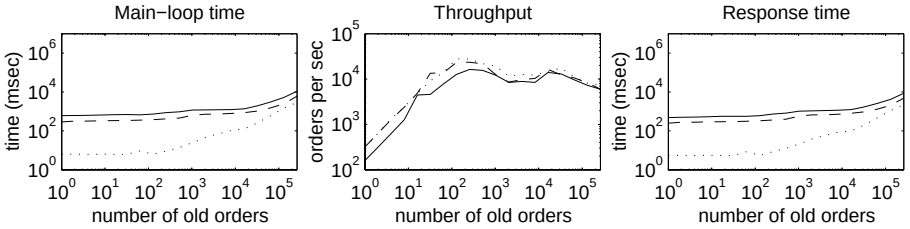

Fig. 4. Dependency of the performance on the number of old orders in the corporate-bond market. The dotted lines show experiments with 300 new orders and matching density of 0.0001. The dashed lines are for 10,000 new orders and matching density of 0.001. The solid lines are for 10,000 new orders and matching density of 0.01.

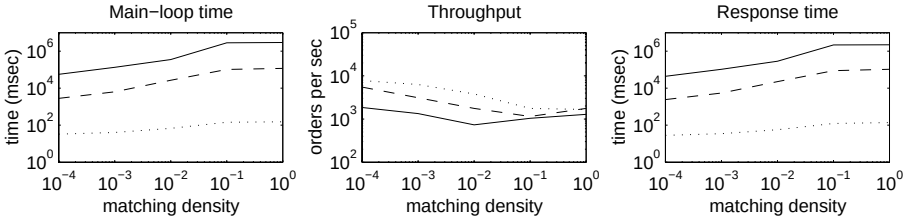

Fig. 5. Dependency of the performance on the matching density in the used-car market. The dotted lines show experiments with 300 old orders and 300 new orders. The dashed lines are for 10,000 old orders and 10,000 new orders. The solid lines are for 300,000 old orders and 10,000 new orders.

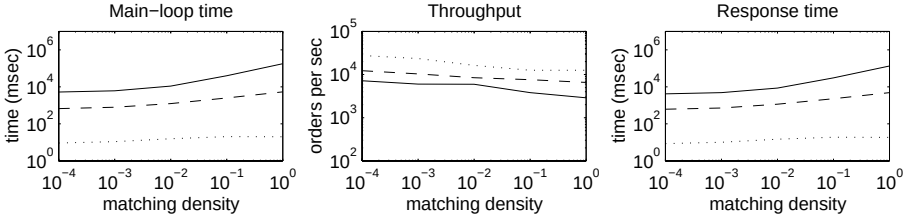

Fig. 6. Dependency of the performance on the matching density in the corporate-bond market. The dotted lines show experiments with 300 old orders and 300 new orders. The dashed lines are for 10,000 old orders and 10,000 new orders. The solid lines are for 300,000 old orders and 10,000 new orders.

6 Concluding Remarks

We have proposed a formal model for trading complex multi-attribute goods, and built an exchange system that supports markets with up to 300,000 orders on a 2-GHz computer with one-gigabyte memory. The system keeps all orders in main memory, and its scalability is limited by the available memory. We are presently working on a distributed system that includes a central matcher and multiple preprocessing modules, whose role is similar to that of stock brokers.

Acknowledgments. We are grateful to Hong Tang for her help in preparing this article, and to Savvas Nikiforou for his help with software and hardware installations. We 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

1. Rica Gonen and Daniel 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.
2. Rica Gonen and Daniel Lehmann. Linear programming helps solving large multi-unit combinatorial auctions. In *Proceedings of the Electronic Market Design Workshop*, 2001.
3. Jianli Gong. Exchanges for complex commodities: Search for optimal matches. Master's thesis, Department of Computer Science and Engineering, University of South Florida, 2002.
4. 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.
5. Joshua Marc Johnson. Exchanges for complex commodities: Theory and experiments. Master's thesis, Department of Computer Science and Engineering, University of South Florida, 2001.
6. 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.
7. Noam Nisan. Bidding and allocation in combinatorial auctions. In *Proceedings of the Second* ACM *Conference on Electronic Commerce*, pages 1–12, 2000.
8. Tuomas W. Sandholm. Approach to winner determination in combinatorial auctions. *Decision Support Systems*, 28(1–2):165–176, 2000.
9. Tuomas W. Sandholm and Subhash Suri. Improved algorithms for optimal winner determination in combinatorial auctions and generalizations. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence*, pages 90–97, 2000.
10. Tuomas W. Sandholm and Subhash Suri. Market clearability. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pages 1145–1151, 2001.
11. 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.