EXCHANGES FOR COMPLEX COMMODITIES:

THEORY AND EXPERIMENTS

by

JOSHUA JOHNSON

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science
Department of Computer Science and Engineering
College of Engineering
University of South Florida

August 2001

Major Professor: Eugene Fink, Ph.D.

Acknowledgements

I gratefully acknowledge the help of Eugene Fink, who has supervised my thesis work and guided me through all steps of research and writing. I am also grateful to Lawrence Hall and Srinivas Katkoori for their valuable comments and suggestions.

The reported work has been a joint project with PowerLoom Corporation, which has provided research funding for this project. PowerLoom employees have participated in the development and implementation of the described system. I am especially thankful to Ganesh Mani, Dwight Dietrich, Steve Fischetti, and Michael Foster for their feedback and help in understanding of real-world exchange markets.

I have received invaluable support from my wife, Katie Johnson, who has provided comfort and encouragement through my long years of undergraduate and graduate studies. My parents, Scott and Marianne Johnson, have also given me never ending support. I thank my sister, Leah Nelson, for helping me with matters of grammar; and finally, I thank Savvas Nikiforou for his helpful comments and assistance with LaTeX.

Table of Contents

Abstract iii

Chapter 1 Introduction 1
1.1 Motivation 1
1.2 Example 3
1.3 Previous Work 7
1.3.1 Combinatorial Auctions 8
1.3.2 Advanced Semantics 9
1.3.3 Exchanges 10
1.3.4 General-Purpose Systems 11
1.4 Contributions 12

Chapter 2 General Problem 13
2.1 Buyers and Sellers 13
2.2 Concept of an Order 14
2.3 Quality Function 15
2.4 Order Size 17
2.5 Fills and Order Execution 18
2.6 Market Attributes 21

Chapter 3 Matcher Engine 22
3.1 Order Representation 22
3.1.1 Sell Items 22
3.1.2 Buy Item Sets 23
3.1.3 Attribute Sets 24
3.1.4 Price and Size 25
3.2 Basic Functionality 25
3.2.1 Matches and Fills 26
3.2.2 Architecture 27
3.2.3 Order Modification 30
3.2.4 Fairness Heuristics 30
3.3 Complex Orders 31
3.3.1 Complex Sells 32
3.3.2 Complex Buys 34
3.3.3 Matching 35
3.4 Confirmations 37

Chapter 4 Performance 40
4.1 Artificial Markets 40
4.1.1 Control Variables 40
4.1.2 Measured Variables 41
4.1.3 Summary Graphs 42
4.2 Real Markets 50
4.2.1 Used Cars 50
4.2.2 Commercial Paper 61

Chapter 5 Concluding Remarks 72

References 73

Appendix A Experiments with a Small Number of Attributes 79
A.1 Processing Time 80
A.2 Matching Time 91
A.3 Response Time 102
A.4 Maximal Throughput 113

Appendix B Experiments with a Large Number of Attributes 123
B.1 Processing Time 124
B.2 Matching Time 135
B.3 Response Time 146
B.4 Maximal Throughput 157

EXCHANGES FOR COMPLEX COMMODITIES:

THEORY AND EXPERIMENTS

by

JOSHUA JOHNSON

An Abstract

of a thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science
Department of Computer Science and Engineering
College of Engineering
University of South Florida

August 2001

Major Professor: Eugene Fink, Ph.D.

The modern economy includes a variety of markets, and the Internet has opened opportunities for efficient on-line trading. Researchers have developed algorithms for various auctions, which have become a popular means for on-line sales. They have also designed algorithms for exchange-based markets, similar to the traditional stock exchange, which support fast-paced trading of rigidly standardized securities. On the other hand, there has been little work on exchanges for complex nonstandard commodities, such as used cars or collectible stamps.

We propose a formal model for trading of complex goods and services, and present an automated exchange for a limited version of this model. The exchange allows the traders to describe commodities by multiple attributes; for example, a car buyer may specify a model, options, color, and other desirable properties. Furthermore, a trader may enter constraints on the acceptable items rather than a specific item; for example, a buyer may look for any car that satisfies certain constraints, rather than for one particular vehicle.

We present an extensive empirical evaluation of the implemented exchange, using artificial data, and then give results for two real-world markets, used cars and commercial paper. The experiments show that the system supports markets with up to 260,000 orders, and generates one hundred to one thousand trades per second.

Abstract Approved: ______________________________

Major Professor: Eugene Fink, Ph.D.
Assistant Professor, Department of Computer Science

Date Approved: ______________________________

Chapter 1

Introduction

1.1 Motivation

Economists define *market* as "an arrangement which permits numerous buyers and sellers of related commodities to carry on extensive business transactions on a regular, organized basis" [Trenton, 1964]. The modern economy includes a wide variety of markets, from cars to software to office space.

The supply chain between a manufacturer and customer may include several middlemen. For instance, customers usually buy cars through dealerships, which in turn acquire cars from manufacturers; the sale of used cars may also involve dealers, who serve as middlemen in the secondary market.

These resales increase the cost of goods, since they include commissions for the middlemen. The recent growth of the Internet has opened opportunities for reducing the number of middlemen [Klein, 1997; Turban, 1997; Wrigley, 1997], and many companies have experimented with direct sales over the web. Middlemen are also using the Internet to increase the volume of their sales and reduce expenses. Furthermore, many companies specialize in the development of electronic marketplaces, which include bulletin boards, auctions, and exchanges.

Electronic bulletin boards are similar to traditional newspaper classifieds. These boards vary from newsgroup postings to on-line sale catalogs, and they help buyers and sellers find each other; however, they often require a user to invest significant effort into searching among multiple ads. For this reason, many buyers prefer on-line auctions, such as eBay (www.ebay.com).

Auctions have their own problems, which include significant computational costs, transaction delays, and asymmetry between buyers and sellers. A traditional auction requires a buyer to bid on a specific item. It helps sellers to obtain the highest price, but limits buyers' flexibility. A *reverse auction* requires a seller to bid on a customer's order; thus, it benefits buyers, and restricts the sellers' flexibility. Furthermore, auctions limit the liquidity, that is, they may cause significant transaction delays. For example, if a seller posts an item on eBay, she can sell it in three or more days, depending on the selected duration of the auction, but not sooner. Thus, auctions are not appropriate for fast sales, which are essential in many markets.

An exchange-based market does not have these problems: it ensures symmetry between buyers and sellers, and supports fast-paced trading. Examples of liquid markets include the traditional stock and commodity exchanges, such as the New York Stock Exchange and Chicago Mercantile Exchange, as well as currency and bond exchanges. For instance, a trader can buy or sell any public stock in seconds, at the best available price. Although stocks have long served as an example of an efficient market, trading in other industries has not reached this efficiency.

The main limitation of traditional exchanges is rigid standardization of tradable items. For instance, the New York Stock Exchange allows trading of about 3,100 securities, and the buyer or seller has to indicate a specific item, such as IBM stock. For most goods and services, however, the description is much more complex. For instance, a car buyer may need to specify a make, model, options, color, and other desirable features. Furthermore, she usually has a certain flexibility and may accept any car that satisfies her constraints, rather than looking for one specific vehicle. For example, she may be willing to get any red Mustang with air conditioning.

Building an exchange for such complex commodities is a major open problem.

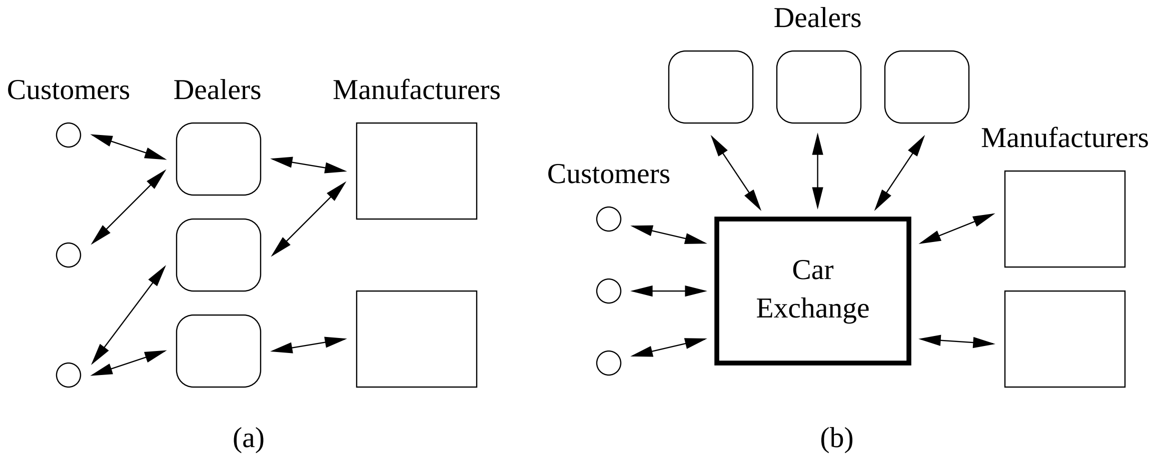


Figure 1.1: The traditional car market versus a centralized exchange. The usual market includes customers, dealers, and manufacturers (a). An alternative scheme is trading through the central exchange (b).

An effective trading system should satisfy the following requirements:

- Allow complex constraints in specifications of buy and sell orders
- Support fast-paced trading for large markets, with millions of orders
- Include optimization techniques that maximize the traders' satisfaction
- Ensure the "fairness" of the market, according to financial industry standards
- Allow a user to select preferred trades among matches for her order

1.2 Example

We give an example of a car exchange, which would allow manufacturers, dealers, and customers to trade new and used vehicles. To simplify this example, we assume that a user can describe a car by four attributes: model, color, year, and mileage; for instance, a seller may offer a red Mustang, made in 1996, with 60,000 miles.

In Figure 1.1(a), we illustrate a traditional car market, which includes manufacturers, car dealers, and customers. The dealers get cars from manufacturers and

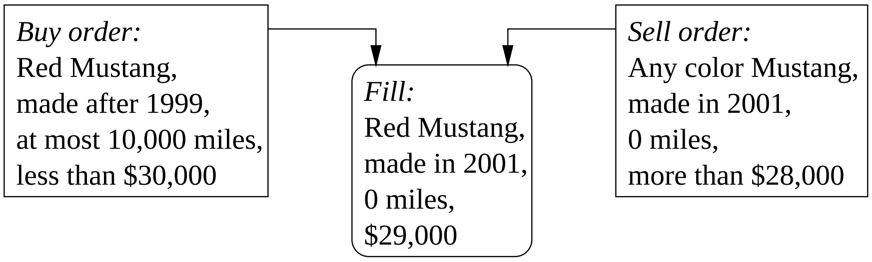


Figure 1.2: Matching orders and the resulting trade. When the system discovers a match between two orders, it generates a fill, which is a trade that satisfies both parties.

resell them to individual drivers. In Figure 1.1(b), we show an alternative trading model, with a centralized exchange, similar to a stock market. Car dealers may still participate in this exchange, by acquiring large quantities of cars from a manufacturer and reselling them to customers; however, the dealers are no longer an essential link, and manufacturers can sell directly to end users.

The exchange allows the users to place buy and sell orders, analogous to the orders in a stock market. A prospective buyer may 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 to purchase a red Mustang, made after 1999, with less than 10,000 miles, and she is willing to pay at most $30,000. Similarly, a user may place a *sell order,* which includes the same main elements. For instance, a manufacturer may offer brand-new Mustangs of any color, for $28,000.

The exchange system searches for matches between buy and sell orders, and generates corresponding *fills,* that is, transactions that satisfy both buyers and sellers. In the previous example, the system will notice that a brand-new red Mustang for $29,000 satisfies both buyer and seller, and it will generate a fill (see Figure 1.2). Thus, the customer will acquire a red Mustang from the manufacturer, for $29,000.

If the system does not find fills for some orders, it keeps them in memory and tries to match them with new, incoming orders. We illustrate it in Figure 1.3, where

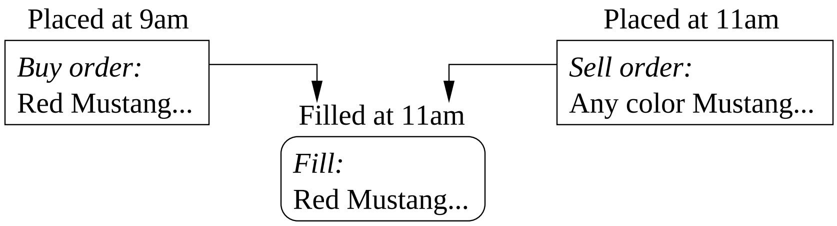


Figure 1.3: If the system cannot find an immediate match for an order, it keeps looking for matches among newly placed orders. In this example, it gets the buy order at 9am and finds a match two hours later.

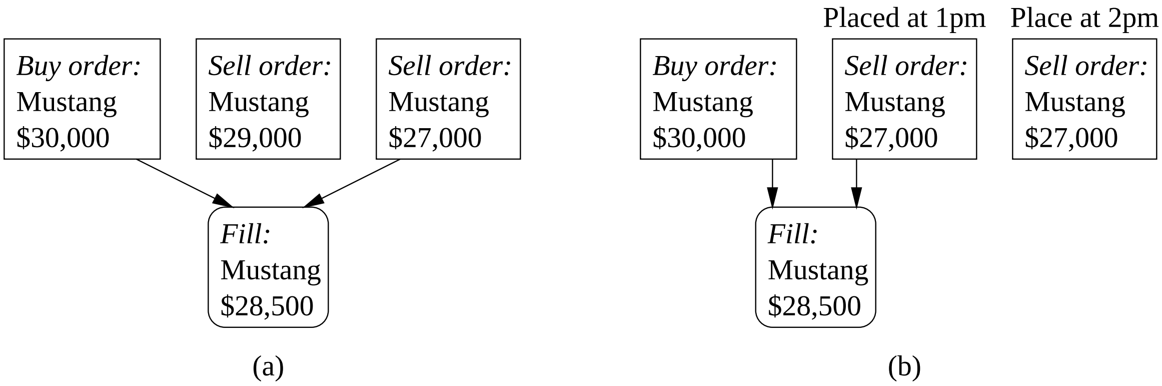


Figure 1.4: Fairness rules: When the system finds several matches for an order, it chooses the match with the best price (a); if two matches have the same price, it prefers the earlier order (b).

the buy order remains open until the system receives a matching sell order two hours later.

If the system finds several matches for an order, it chooses the match with the best price. For example, the buy order in Figure 1.4(a) will trade with the cheaper of the two sell orders. If two matching orders have the same price, the exchange gives preference to the earlier order, as shown in Figure 1.4(b). These rules correspond to the standard "fairness" requirements of the financial industry.

The system allows a user to trade several identical items, by specifying a size for an order. For example, a dealer can place an order to sell four Mustangs, which may trade with several different buy orders. The system may first match it with a two-car buy order (see Figure 1.5), and later find another match for the remaining

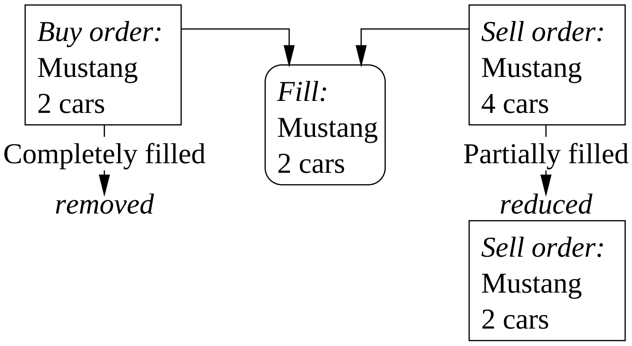


Figure 1.5: Buyers and sellers may trade several items at once, by specifying order sizes. When the system finds a match, it completely fills the smaller order and reduces the size of the larger order.

two cars.

A user can specify that she is willing to buy or sell any of several items. For example, a customer can place an order to buy either a Mustang or Corvette. As another example, a dealer can place an order to sell ten cars, which may be Mustangs, Escorts, and Rangers. In the latter example, the dealer may end up selling ten Mustangs, or ten Escorts, or three Mustangs and seven Rangers.

If a user describes a set of items, she can indicate that the price depends on an item. For example, if a customer wants a Mustang or Corvette, she may offer $30,000 for a Mustang, but only $28,000 for a Corvette. Furthermore, she may offer an extra $200 if a car is red, and subtract $1 for every ten miles on its odometer. She can also specify her preferences for selecting among potential trades; for instance, she may indicate that a red Mustang is better than a white Mustang, and that a Mustang for $29,000 is better than a Corvette for $28,000 (see Figure 1.6).

A customer may request additional information about potential trades before committing to one of them. For instance, she may view the pictures of matching cars, along with their technical descriptions, and manually select the best match.

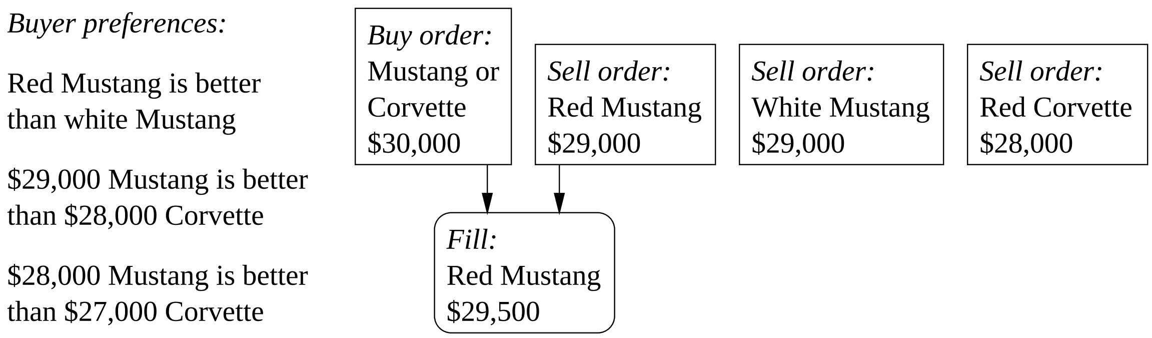


Figure 1.6: Example of preferences: A customer wants to buy a Mustang or Corvette, and specifies her preferences; the system uses them to choose among matching orders.

1.3 Previous Work

Economists and computer scientists have long realized the importance of auctions and exchanges for the modern economy, and studied a variety of trading models. The related computer science research has been focused on optimal matches in various auction scenarios, and on general-purpose systems for auctions and exchanges.

This research has led to successful systems for Internet auctions, including Yahoo Auctions (auctions.yahoo.com), eBay (www.ebay.com), and Bid.Com (www.bid.com), which support millions of traders. Some companies have also built on-line auctions for business-to-business transactions; for example, FreeMarkets (www.freemarkets.com) has deployed a reverse auction, which allows suppliers to bid for a large contract with a business customer.

Although these auctions are more effective than traditional bulletin boards, they have the typical drawbacks of auction markets, including asymmetric treatment of buyers and sellers, and significant computational requirements.

Recently, researchers have developed several efficient systems for *combinatorial auctions,* which allow buying and selling sets of commodities rather than individual items. They have considered not only auctions with completely specified commodities, but also markets that allow the user to negotiate desirable features of merchandise.

1.3.1 Combinatorial Auctions

A traditional combinatorial auction allows bidding on a set of fully specified items. For example, Katie may bid on a red Mustang, black Corvette, and silver BMW, for a total price of $80,000. In this case, she will get all three cars together or nothing; that is, the system will not generate a partial fill.

An advanced auction may allow disjunctions; for instance, Katie may specify that she wants either a red Mustang and black Corvette or, alternatively, two silver BMWs. 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 discussed restrictions that ensure computational feasibility. They described semantics of combinatorial bids that allow fast matching, but did not develop a matching algorithm. Nisan discussed alternative semantics for combinatorial bids, formalized the problem of searching for optimal and near-optimal matches, and proposed a linear-programming solution, but did not test its effectiveness [Nisan, 2000; Lavi and Nisan, 2000].

Sandholm [1999] developed several efficient algorithms for one-seller combinatorial auctions, and showed that they scaled to a market with about one thousand 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.*, 2001].

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

Andersson *et al.* [2000] compared the main techniques for combinatorial auctions, and proposed an integer-programming representation of auctions, which allowed a richer bid semantics. In particular, they removed some of the restrictions imposed by Rothkopf *et al.* [1998].

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.

1.3.2 Advanced Semantics

Several researchers studied techniques for processing "flexible" bids, specified by hard and soft constraints, similar to buy orders in Figures 1.2, 1.3, and 1.6.

Bichler discussed a market that would allow negotiation on any attributes of a commodity [Bichler *et al.*, 1999; Bichler, 2000]; for instance, a car buyer could set a fixed price and negotiate on the options and service plan. He analyzed several alternative versions of this model, and concluded that it would greatly increase the economic utility of auctions; however, he pointed out the difficulty of implementing it and did not propose any computational solution.

Jones and Koehler extended the semantics of combinatorial auctions and allowed buyers the use of complex constraints [Jones and Koehler, 2000; Jones, 2000]; for instance, a car buyer could bid on a car that was less than three-years old, or on the fastest available car. They suggested an advanced semantics for these constraints, which allowed compact description of complex bids; however, they did not allow complex constraints in sell orders. They implemented an algorithm that supported this semantics and found near-optimal matches; however, it scaled only to one thousand 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 development of efficient matching algorithms for advanced semantics.

1.3.3 Exchanges

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

Successful on-line exchanges include electronic communication networks, such as REDI (www.redibook.com), Island (www.island.com), NexTrade (www.nextrade.org), Archipelago (www.tradearca.com), and Instinet (www.instinet.com). The directors of large stock and commodity exchanges are also considering electronic means of trading. For example, the Chicago Mercantile Exchange has deployed the Globex electronic trading system, which supports trading around the clock.

Some auction researchers have investigated the related theoretical issues; they have traditionally viewed exchanges as a variety of auction markets, called *continuous double auctions.* In particular, Wurman *et al.* [1998a] proposed a theory of exchange markets and implemented a general-purpose system for auctions and exchanges, which processed traditional fully specified orders. Sandholm and Suri [2000] developed an exchange for combinatorial orders, similar to bids in combinatorial auctions; however, their system could not support markets with more than one thousand pending orders.

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

1.3.4 General-Purpose Systems

Computer scientists have developed several systems for different types of auctions and exchanges, which have varied 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].

For example, Chavez and his colleagues designed an on-line agent-based auction; specifically, they built intelligent agents that negotiated with one another, 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, which supported several types of auctions. Preist [1999a; 1999b] developed a similar distributed system that supported exchange markets. Bichler designed an electronic brokerage service, which helped buyers and sellers find each other and negotiate through auction mechanisms [Bichler *et al.*, 1998; Bichler and Segev, 1999].

Wurman and Wellman developed a general-purpose system, which could run a variety of different auctions [Wellman, 1993; Wellman and Wurman, 1998; Wurman *et al.*, 1998b]; however, they restricted the users to simple fully specified bids. Parkes built a fast system for combinatorial bids, but it worked only for small markets, with up to one hundred users [Parkes, 1999; Parkes and Ungar, 2000]. Sandholm created a more powerful server for combinatorial auctions, configurable for a variety of markets, and showed its ability to process several thousand bids [Sandholm, 2000a; Sandholm, 2000b; Sandholm and Suri, 2000].

All of these systems had the same key limitations as commercial on-line exchanges: they required fully specified bids and did not support the use of constraints.

1.4 Contributions

The review of previous work has shown that techniques for trading of complex commodities are still limited. Researchers have investigated several auction models, as well as exchanges for standardized securities, but they have not applied the exchange model to complex goods. The main open problems are (1) design of an automated exchange for complex securities, (2) analysis of related trading rules, and (3) development of a rigorous theory of complex exchanges.

The work reported here is a step toward addressing these problems. We define complex orders and propose related trading semantics. Specifically, we view a market as a set of tradable items, and an order as its subset. The proposed model is applicable to a variety of markets, from financial securities to automobiles. We have used it in developing a prototype exchange system, in collaboration with PowerLoom Corporation, the sponsor for this project.

We have performed an extensive empirical evaluation of the resulting system, using a suite of artificial markets, as well as used-car and commercial-paper markets. The experiments have shown that the system allows fast trading in large-scale markets. For instance, it supports the used-car market with 260,000 pending orders and 300 new orders per second, on a single 400-MHz workstation.

First, we analyze a general trading problem and formalize the concept of complex commodities (Chapter 2). Then, we describe the implemented functionality, and discuss its advantages and main limitations (Chapter 3). Finally, we give results of testing the system and show how its performance depends on the market size, order complexity, and other properties of the exchange (Chapter 4).

Chapter 2

General Problem

We formalize a general problem of building an automated exchange, and illustrate it with car-market examples. The purpose of an exchange is to allow purchase and sale of certain items; in other words, it helps prospective buyers and sellers find each other.

2.1 Buyers and Sellers

When a buyer looks for a certain item, she usually has some flexibility; that is, she is willing to buy any of several acceptable items. For example, suppose that Katie is looking for a sports car; then, she may be willing to buy one of several models, such as a Corvette, Mustang, or Viper. For each of these models, Katie has to determine the maximal acceptable price. Furthermore, a buyer usually has preferences among acceptable items; for instance, Katie may prefer Mustangs to other models, and she may prefer red cars to black ones. The preferences may depend on the price, features, or other factors, such as service quality or delivery date.

Similarly, when a dealer sells a vehicle, she has to decide on a minimal acceptable price. For instance, Laura may be selling a Corvette for no less than $15,000 and a Mustang for no less than $20,000. If the seller offers multiple items, she may prefer some sales to others. For example, Laura may prefer to sell the Mustang for $20,000, rather than the Corvette for $15,000. If Katie came to Laura to purchase a sports car, then Laura would try to sell the Mustang before offering the Corvette.

If a buyer's constraints match a seller's constraints, then they may *trade;* that is, the buyer may purchase an item from the seller. If a buyer finds several acceptable items, possibly provided by different sellers, she will buy the best available item, where the notion of "best" depends on her subjective preferences. Similarly, a seller may be able to choose the most attractive deal among several bids.

We use the term *buy order* to refer to a buyer's set of requirements and preferences. For example, Katie's desire to purchase a sports car can be expressed as an *order* for a sports car, and her price limits and preferences will be a part of this order. When a buyer announces her desire to trade, we say that she has *placed an order.*

Similarly, a *sell order* is a seller's set of constraints, which define the offered merchandise. For example, Laura may place an order to sell a Mustang or Corvette, and her order may also include price limits and preferences. If a buy and sell order match, they may result in a trade between the corresponding parties.

2.2 Concept of an Order

A specific market includes a certain set of items that can potentially be bought and sold; we denote it by M, which stands for *market set.* Note that this set may be very large or even infinite; in the car market, M includes all vehicles that have ever been made, as well as vehicles that *can* be made in the future. The choice of a market set limits the objects that can be traded, but it does not guarantee that all of these objects will be traded. For instance, if we restrict M to cars, then the market does not allow trading of bicycles or golf carts. On the other hand, even if M includes Star Wars land speeders, they may never be traded.

When a customer makes a purchase or sale, she needs to specify a set of acceptable items, denoted I, which stands for *item set;* it must be a subset of M, that is, $I \subseteq M$. For example, if Katie shops for a brand-new sports car, then her set I

includes all new sports vehicles.

In addition, a customer should specify a limit on the price that she is willing to pay, which may depend on specific items in I. For instance, Katie may be willing to pay \$25,000 for a red Mustang, but only \$24,000 for a black Mustang, and even less for a Corvette. Formally, a price limit is a real-valued function defined on the set I, whose values are nonnegative; for each item $i \in I$, it defines a certain limit, $Price(i)$. If a customer is buying an item, then $Price(i)$ is the maximal acceptable price. For a seller, on the other hand, it is the minimal acceptable price.

Formally, a buy or sell order must include two elements (see Figure 2.1a):

- a set of items $I \subseteq M$, and
- a price function $Price(i) \colon I \to \mathbf{R}^+$,

  where $\mathbf{R}^+$ is the set of nonnegative real-valued prices.

We say that a buy order *matches* a sell order if the buyer's constraints are consistent with the seller's constraints, thus allowing a mutually acceptable trade (see Figure 2.1b). For instance, if Katie is willing to pay \$20,000 for a red Corvette, and Laura is ready to sell a red Corvette for \$19,000, then their orders match.

Formally, let $(I_b, Price_b)$ be a buy order and $(I_s, Price_s)$ be a sell order. Then, these orders match if some item i satisfies both buyer and seller, at a mutually acceptable price:

$$\text{there exists } i \in I_b \cap I_s \text{ such that } Price_b(i) \geq Price_s(i).$$

2.3 Quality Function

Both buyers and sellers may have preferences among acceptable trades, which depend on a specific item i and its price p. For instance, Katie may prefer a red Mustang for

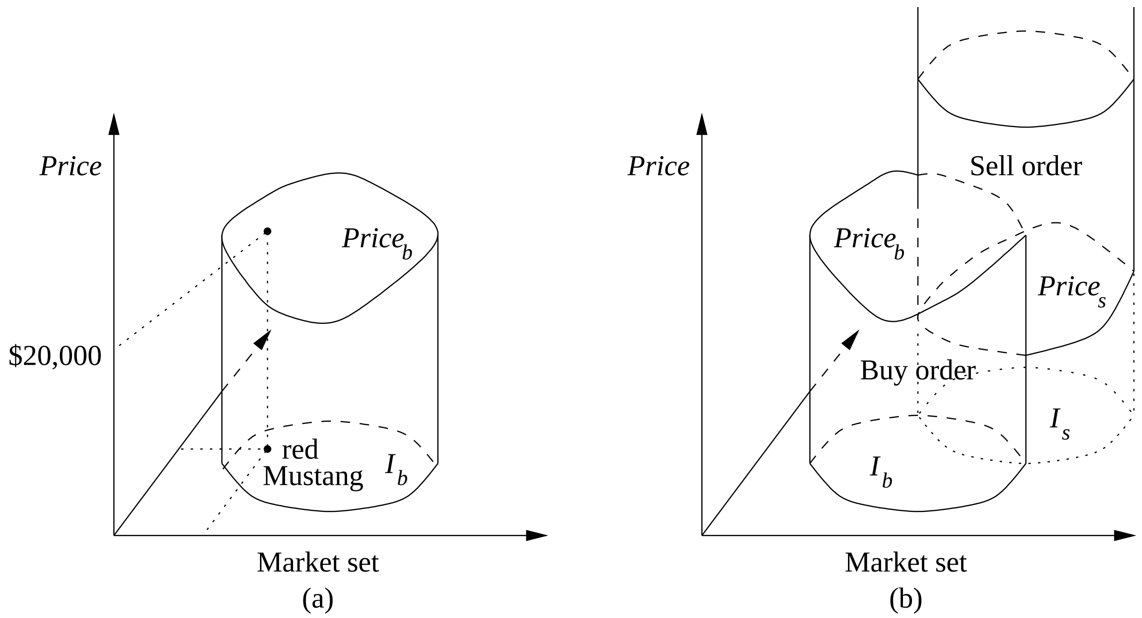


Figure 2.1: An example of a buy order (a) and a match between a buy and sell order (b). The horizontal plane represents the market set M, and the vertical axis is price $\mathbf{R}^+$. The buyer is interested in a certain set I_b of cars, with different price limits; in particular, she would buy a red Mustang for \$20,000. Her order matches the sell order shown on the right.

\$25,000 to a black Corvette for \$20,000.

We define these preferences as a real-valued function $Qual(i, p)$, which assigns a numeric quality to each pair. The larger values correspond to "better" items; that is, if $Qual(i_1, p_1) > Qual(i_2, p_2)$, then a customer would rather pay p_1 for i_1 than p_2 for i_2. For example, Katie's quality function would satisfy the following inequality:

$$Qual(\texttt{red-Mustang}, \$25{,}000) > Qual(\texttt{black-Chevrolet}, \$20{,}000).$$

Each customer may use her own quality function; furthermore, she may specify different functions for different orders. Note that we define quality as a *totally ordered* function, which is a simplification. In real life, customers sometimes reason in terms of partially ordered functions. For instance, Katie may believe that a \$25,000 Mustang is better than a \$20,000 Corvette, but she may be undecided between a \$25,000

Mustang and an \$18,000 Corvette.

Also note that buyers prefer lower prices, whereas sellers try to get as much money as possible, which means that all 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, $Qual_b(i, p_1) \geq Qual_b(i, p_2)$.
- **Sell monotonicity:** If $Qual_s$ is a quality function for a sell order, and $p_1 \leq p_2$, then, for every item i, $Qual_s(i, p_1) \leq Qual_s(i, p_2)$.

2.4 Order Size

If a user wants to buy or sell several identical items, she may include their number in the order specification; for example, Katie can place an order to buy two sports cars, and Laura can announce a sale of one thousand Corvettes.

We assume that the order size is a natural number, that is, the market participants buy and sell whole items. This assumption is somewhat restrictive, since it enforces discretization of continuous commodities, such as copper or orange juice.

The user may specify not only the overall order size, but also the minimal acceptable size. For instance, suppose that Amanda is a wholesale agent for Chevrolet, and she needs to sell one thousand cars. Furthermore, she has no time for individual sales, and works with dealerships that are buying at least ten cars at once. Then, she may specify that the overall size of her sell order is one thousand, and the minimal acceptable size is ten.

To summarize, an order may include five elements:

- Item set, I
- Price function, $Price\colon I \to \mathbf{R}^+$

- Quality function, $Qual$: $I \times \mathbf{R}^+ \rightarrow \mathbf{R}$
- Order size, Max
- Minimal acceptable size, Min

The item set, price limit, and size specification form hard constraints, which determine whether a buy and sell order match each other. To define the matching conditions, we denote the item set of a buy order by I_b, its price function by $Price_b$, and its size parameters by Max_b and Min_b. Similarly, we denote the parameters of a sell order by I_s, $Price_s$, Max_s, and Min_s. The two orders match if they satisfy the following two conditions:

- For some item $i \in I_b \cap I_s$, $Price_b(i) \geq Price_s(i)$
- $Min_b \leq Max_s$ and $Min_s \leq Max_b$

The quality function is a soft constraint, which does not affect the matching conditions; it defines a user's preference among available matches.

2.5 Fills and Order Execution

When a buy order matches a sell order, the corresponding parties may complete a *trade,* which involves delivery of an appropriate item or multiple items to a buyer, for an appropriate price. We use the term *fill* to refer to the traded items and their price.

For example, suppose that Katie has placed an order for two sports cars, and Laura is selling three red Mustangs. If the prices of these orders match, Katie may purchase two red Mustangs from Laura; in this case, we say that two red Mustangs are a fill for her order. Formally, a fill consists of three parts: a specific item i, its price p, and the number of purchased items, denoted *size.*

Suppose that $(I_b, Price_b, Max_b, Min_b)$ is a buy order, and $(I_s, Price_s, Max_s, Min_s)$ is a matching sell order. A valid fill $(i, p, size)$ for these orders must satisfy the following three conditions:

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

Note that a fill is *fully specified,* that is, it consists of a specific item, price, and quantity. Unlike an order, it cannot include a set of possible items or a range of different sizes. Furthermore, all items in a fill have the same price; for instance, a fill (`red-Mustang`, \$20,000, 2) means that Katie purchased two red Mustangs at \$20,000 each. If she had bought these cars for different prices, we would represent them as two different fills for the same order.

If both buyer and seller specify a set of items, we may have freedom to select one of several possible items: the resulting fill can contain any item $i \in I_b \cap I_s$. For example, suppose that Katie wants to buy a sports car, and Laura has placed an order to sell Mustangs, Corvettes, and Vipers. Then, the resulting fill can contain any of these models. Similarly, we may have some freedom in selecting the price and size of the fill. The heuristics for making these choices depend on a specific implementation.

- **Choice of an item:** If $I_b \cap I_s$ includes several items, the choice of an item may be random. Alternatively, we may choose an item to maximize either the buyer's preference function or the seller's preferences. A more complex heuristic may search for an item that maximizes the overall satisfaction of the buyer and seller.

- **Price choice:** The default strategy is to split the price difference between the buyer and seller, which means that $p = \frac{Price_b(i)+Price_s(i)}{2}$. Another standard

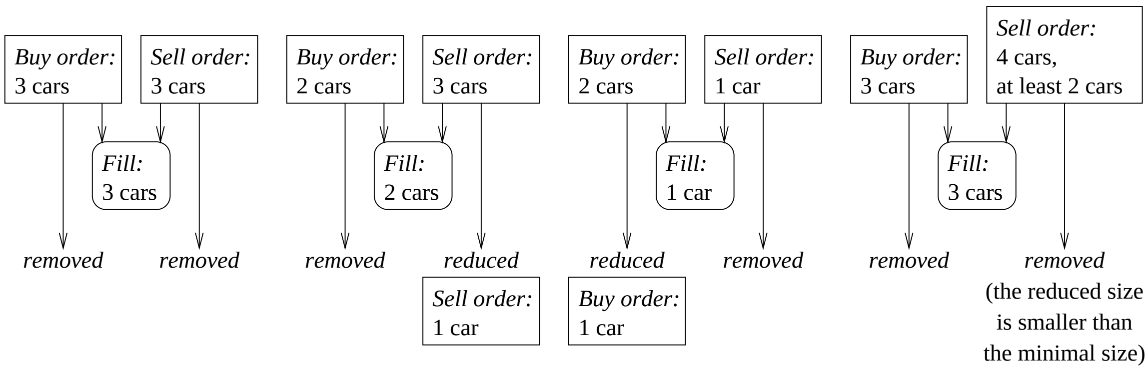


Figure 2.2: Examples of order execution.

option is to favor either the buyer or seller; that is, we may always use $p = Price_s(i)$ or, alternatively, we may always use $p = Price_b(i)$.

- **Size choice:** We assume that both buyers and sellers are interested in trading at the maximal size, or as close to the maximal size as possible. Thus, the fill has the largest possible size, that is, $size = \min(Max_b, Max_s)$. This default is the same as in stock and futures trading.

After a buyer or seller has gotten a fill, she may keep the initial order, reduce its size, or remove the order; the default option is size reduction. For example, if Laura ordered a sale of three cars and has gotten a two-car fill, then the size of her order becomes one.

If the reduced size is zero, we remove the order from the market. If the size remains positive but drops below the minimal acceptable size Min, the order is also removed. For example, if Amanda indicated that her minimal sale is ten cars, and her order size has dropped to five cars, then this order is cancelled.

The process of generating a fill and then reducing the buy and sell orders is called an *execution* of orders. We illustrate different scenarios of order execution in Figure 2.2.

2.6 Market Attributes

The set M of all possible items may be very large, which means that we cannot explicitly represent all items. For example, we probably cannot make a catalog of all feasible cars, since it would include a separate entry for each possible combination of models, colors, features, and other attributes that describe a specific vehicle.

To avoid this problem, we define a set M by a list of attributes and possible values of each attribute. As a simplified example, we may define a used car by four attributes: *Model, Color, Year,* and *Mileage.* Then, a user describes a specific car by substituting values for these attributes; for example, a seller may offer a red Mustang, made in 1998, with 30,000 miles.

Formally, every attribute is a set of values; for instance, the *Model* set may include all car models, *Color* may include all visible wavelengths, *Year* may include the integer values from 1896 to 2001, and *Mileage* may include real values from 0 to 500,000. The market set M is a *Cartesian product* of these attribute sets; in this example, $M = \mathit{Model} \times \mathit{Color} \times \mathit{Year} \times \mathit{Mileage}$. If the market includes n attributes, then each item is an n-tuple; in the car example, it is a quadruple that specifies the model, color, year, and mileage.

The Cartesian-product representation is a simplification, based on the assumption that all items in the market have the same attributes. Some markets do not satisfy this assumption; for example, if we trade chariots and Star Wars land speeders on the same market, we may need two different sets of attributes. We further limit the model by assuming that every attribute set has one of three types:

- A set of explicitly listed values, such as car models
- An interval of integer numbers, such as year
- An interval of real values, such as mileage

Chapter 3

Matcher Engine

We have built a prototype system for a special case of the automated exchange problem. We describe the semantics of orders in the implemented exchange, and then explain its functionality and overall architecture.

3.1 Order Representation

We first describe the representation of item sets and prices in the implemented system, and discuss the related limitations. The representation is less general than the formal model in Chapter 2. In particular, it limits possible item sets and does not allow the use of price and quality functions.

3.1.1 Sell Items

A sell order has to include a specific item, rather than a set of acceptable items. For example, Laura can order the sale of a red Mustang made in 1998, which has 10,000 miles; however, she cannot offer a set of various Mustangs made between 1990 and 2000. If she is selling multiple different cars, she needs to place multiple orders.

This limitation is based on the assumption that sellers usually offer specific items; however, some real-world markets do not satisfy this assumption. In particular, it creates problems for trading of services, such as package delivery or carpet cleaning. For instance, a maid service may offer to clean any carpets, rather than a specific carpet in a specific building.

To describe an item, the seller has to provide a value for each attribute; for example, Laura may define the model as `Mustang`, the color as `red`, and so on. If the market includes n attributes, then the definition of a sell item is a sequence of n values, $(i_1, i_2, \ldots, i_n)$, where i_1 is the value of the first attribute, i_2 is for the second attribute, and so on. For example, Laura would define her car as (`Mustang`, `red`, 1998, 10,000).

3.1.2 Buy Item Sets

A buyer may specify a set I of multiple items, but possible sets are limited by the representation. A buyer has to give a set of acceptable values for each attribute, which is called an *attribute set*. Thus, if the market includes n attributes, the buy-order description contains n attribute sets, and the set I is a Cartesian product of these attribute sets. For example, Katie may indicate that she wants a Mustang or Corvette, the acceptable colors are red, silver, and black, the car should be made in 1998 or later, and it should have no more than 30,000 miles.

To give a formal definition, suppose that the set of all possible values for the first attribute is M_1, the set of all values for the second attribute is M_2, and so on, which means that the market set of all possible items is $M = M_1 \times M_2 \times \ldots \times M_n$. The buyer has to specify a set I_1 of values for the first attribute, where $I_1 \subseteq M_1$, a set of values for the second attribute, $I_2 \subseteq M_2$, and so on. The resulting item set I is the Cartesian product of the specified sets:

$$I = I_1 \times I_2 \times \ldots \times I_n.$$

For instance, Katie has specified the following item set in the automobile example:

$$I = \{\texttt{Mustang}, \texttt{Corvette}\} \times \{\texttt{red}, \texttt{silver}, \texttt{black}\} \times \{1998, 1999, \ldots\} \times [0..30{,}000].$$

Note that an item set in the implemented matcher *must* be a Cartesian product of attribute sets. For example, Katie cannot describe an item set that includes red Mustangs and black Corvettes, but no black Mustangs.

3.1.3 Attribute Sets

A buyer may use specific values or ranges; for example, she may specify a desired year as 2001 or as a range from 1998 to 2001. Note that ranges work only for numeric attributes, such as year and mileage.

The specification of a market may include certain standard sets of values, such as all sports cars or all American cars, and the buyer may use them in her orders. For example, she may place an order for any American car.

Moreover, the buyer may use unions and intersections in her specification of attribute sets. For instance, suppose that Katie is interested in Mustangs, Corvettes, and European sports cars; suppose further that we have defined a standard set that includes all European cars, and another standard set that comprises all sports cars. Then, Katie can represent the desired set of models as follows:

$$\{\texttt{Mustang}, \texttt{Corvette}\} \cup (\texttt{European-cars} \cap \texttt{Sports-cars}).$$

Formally, an attribute set may be:

- A specific value, such as `Mustang` or 2001
- A range of values, such as 1998–2001
- A standard set of values, such as all European cars
- An intersection of several attribute sets
- A union of several sets

3.1.4 Price and Size

A buyer or seller must specify a numeric price p for her order, rather than a price function. The price of a sell order represents the minimal acceptable price for an offered item; for example, Laura may offer a red Mustang for \$20,000.

The price of a buy order is the maximal acceptable price for every item in the order. For instance, suppose that Katie wants to buy a Mustang or Corvette, and the price of her order is \$20,000; then, she is willing to pay that price for either car. On the other hand, if Katie's price limit depends on the model, she has to place a separate order for each car.

The size specification is the same as in the general model, as described in Section 2.4; that is, an order may include a total size *Max* and a minimal acceptable size *Min*. For example, Laura may want to sell four red Mustangs, and refuse to sell less than two at once; in this case, Laura's *Max* size is four and her *Min* size is two.

Note that, if an order includes multiple items, then the specified price is for each item. For instance, if Laura is selling four cars at \$20,000, then the price of each car is \$20,000, and the total price of all four cars is \$80,000.

Also note that, if a buyer specifies a set of items, then the order size determines the total number of items that she is willing to purchase. For example, if Katie is looking for Mustangs and Corvettes, and her order size is two, then she will buy at most two cars; she would not buy two Mustangs and two Corvettes.

3.2 Basic Functionality

We describe the basic operations of the implemented system, which include processing of new orders, search for matches, and generation of fills, as well as modification of previously placed orders.

3.2.1 Matches and Fills

The search for matching orders is based on the definition in Section 2.4; that is, two orders match if the system can choose an item, price, and size that satisfy both of them.

We now define a match for the limited representation of the implemented system. Let $(I_1, I_2, \ldots, I_n, p_b, Max_b, Min_b)$ be a buy order, where $I_1, I_2, \ldots, I_n$ are attribute sets, and the last three parameters specify price and size constraints, and let $(i_1, i_2, \ldots, i_n, p_s, Max_s, Min_s)$ be a sell order, where $i_1, i_2, \ldots, i_n$ describe a specific item. Then, these two orders match if they satisfy the following conditions:

- For every $k \in [1..n]$, we have $i_k \in I_k$
- $p_b \geq p_s$
- $Min_b \leq Max_s$ and $Min_s \leq Max_b$

When two orders match, the system may generate a fill, using the item from the sell order. Since the sell item is fully specified, it uniquely defines the fill item. The size of the resulting fill is the maximal size that matches both orders, which is $\min(Max_s, Max_b)$. The default price of the fill is $\frac{p_b+p_s}{2}$, but this default can be changed in favor of buyers or sellers.

The implemented system includes an indexing structure for sell orders, which allows a fast retrieval of all sells that match a given buy order. On the other hand, the system does not index buy orders, and it cannot efficiently retrieve all buys for a given sell. This asymmetry does not prevent efficient matching, since the implemented "one-way" search finds matches for sell orders.

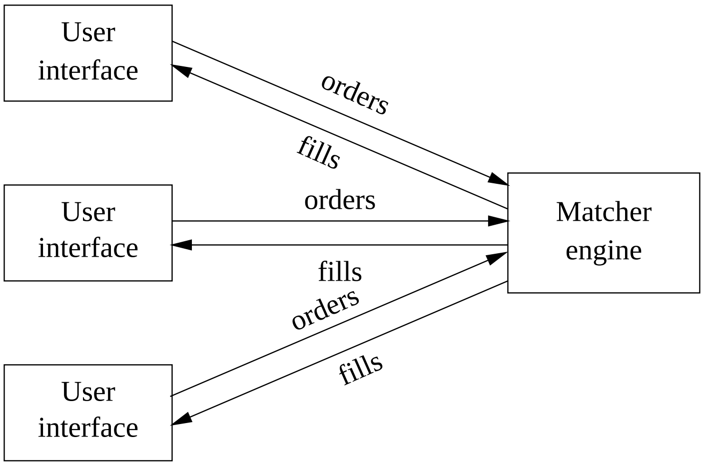


Figure 3.1: The architecture of the trading system.

3.2.2 Architecture

The system consists of a central matcher and multiple interfaces, which run on separate machines and communicate with the matcher over a network (see Figure 3.1).

The users enter their orders through the interface machines, which send the orders to the matcher engine. The central engine serves as a trading pit, similar to a trading floor of the stock exchange; it finds matches among the received orders, generates fills, and sends them to the corresponding interface machines.

The implemented architecture is based on an asynchronous messaging protocol; the matcher receives orders over the network and sends resulting fills. It supports not only "order" and "fill" messages, but also several other message types, such as cancellation and modification of orders; we will discuss some of them in Sections 3.2.3 and 3.4.

In Figure 3.2(a), we illustrate the main cycle of the matcher engine, which alternates between parsing of incoming messages and search for matches. When the matcher parses messages with new orders, it performs a preliminary search for matches and adds the orders to the indexing structure (see Figure 3.2b). We refer to an order that is currently in the system as a *pending order.*

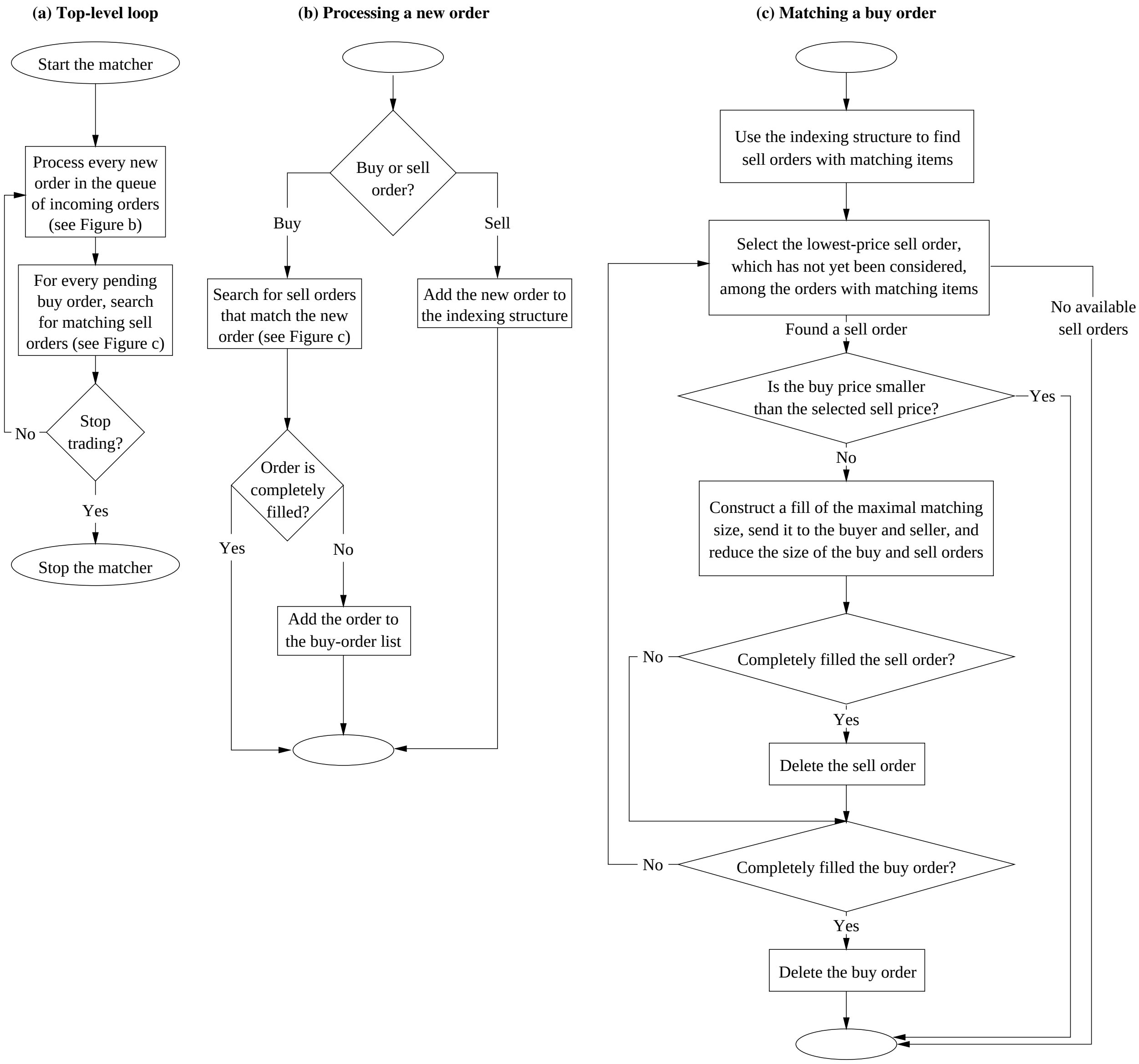


Figure 3.2: Main operations of the matcher engine: top-level loop (a), processing an incoming order (b), and search for sell orders that match a given buy order (c).

When the system receives a new buy order, it immediately searches for matches, which may result in finding a fill before processing the next message. If there are no matches, the system adds the order to the list of buy orders. Similarly, if the system fills only a portion of a large buy order, it stores the remaining part.

When the system gets a new sell order, it adds the order to the indexing structure without searching for matches, since the engine has no efficient mechanism for identifying relevant buy orders. The system will find a fill later, when processing a matching buy order.

For example, if Laura places a sell order for one Mustang, the system adds her order to the indexing structure. If Katie later places a buy order for two sports cars, the system immediately looks for matches among pending sells. It finds Laura's order, generates the corresponding fill, and informs Katie and Laura that they have exchanged a Mustang. Since Laura was selling only one car, the system removes her order from the indexing structure. On the other hand, Katie needs one more car; thus, the system reduces the size of her order and adds it to the list of buy orders.

After processing the messages, the system matches all pending buy orders, which include not only the new arrivals, but also the old unfilled orders. For each buy order, it searches for matching sell orders, as shown in Figure 3.2(c). When the system finds a match, it generates a fill and sends a notification to the buyer and seller interfaces.

The matcher keeps track of the "age" of each pending order, and uses it to avoid repetitive search for matches among the same sell orders. If it has already searched for matches for some buy order and has not found any, then the matching process will involve search only among newly arrived sell orders.

3.2.3 Order Modification

The traders can cancel or modify their old orders. For example, if Katie has ordered two sports cars and has not gotten a fill, she may decide to cancel her order. When the matcher gets a cancellation message, it immediately deletes the order from the indexing structure.

Alternatively, Katie can modify her order, for example, change her price or reduce the order size. The system treats a modification message as a cancellation of an old order and immediate placement of a new one. For sell orders, it means that the modified order replaces the old one in the indexing structure. If a user modifies a buy order, the system immediately searches for matches.

The engine includes several optimization heuristics that prevent unneeded matching upon modification. For example, if Katie reduces the price of a buy order or increases the minimal acceptable size, the system will not perform a search, since these modifications cannot lead to new matches.

A user may indicate a cancellation time for each order, and then the system will automatically remove the order upon reaching the specified time. For instance, Katie may indicate that her order should remain in the system for two days. If Katie later changes her mind, she may manually cancel her order before the specified time or, alternatively, change the cancellation time. The system allows specifying any cancellation time, with one-second precision, and maintains a priority queue of all times.

3.2.4 Fairness Heuristics

If the system identifies multiple matches between buy and sell orders, it may need to choose among them before generating fills. For example, if a buy order matches two different sell orders, the system has to select between the two, and the users usually expect a “fair”choice. We have used help from Michael Foster, a professional trader working for PowerLoom Corporation, to identify standard fairness expectations.

First, if the system has found several matches for the same trade, it should prefer the best-price match. For instance, if Katie is looking for a sports car and the matcher has found two different orders to sell sports cars, then it has to match Katie’s buy order with the cheaper sell.

Second, if several users compete for the same trade, the system should give priority to the user who offers a better price. For instance, suppose that Laura and Michelle are both selling a Corvette, and Laura’s price is better. Then, the system should fill Laura’s order before Michelle’s order. Although traders often view this requirement as different from the first one, both impose the same constraints on the matching process.

Third, if several traders offer the same price, the system should execute their orders on a first-come first-serve basis. Thus, if Laura and Michelle offer Mustangs for the same price, and Laura has made her offer before Michelle, then Laura’s sell order should get priority. Professional traders consider this “chronological” fairness almost as important as price fairness. When a seller makes a low-price offer in a volatile market, she assumes a risk and expects to be rewarded with priority over other sellers who follow her lead.

3.3 Complex Orders

The described semantics allows efficient matching, but limits the system’s flexibility. We now present an extension to the basic semantics, supported in the implemented matcher.

3.3.1 Complex Sells

We have described simple sell orders, which include specific items rather than item sets. The implemented system also enables the seller to enter a list of several alternative items. For instance, Laura may indicate that she wants to sell a car, which may be either a red Mustang or black Corvette; furthermore, she may give different price limits for different cars.

A sell order with several items is called a *complex order;* formally, it is a set of several items and their prices, along with the order's size constraints. We denote the item set as $\{(i1, p1), (i2, p2), \ldots, (ik, pk)\}$, where i's are items and p's are their prices. The user has to describe each item as an n-tuple of its attribute values, and the overall encoding of a complex order is as follows:

Item 1: $(i1_1, i1_2, \ldots, i1_n),\ p1$

Item 2: $(i2_1, i2_2, \ldots, i2_n),\ p2$

$\vdots$

Item k: $(ik_1, ik_2, \ldots, ik_n),\ pk$

Size: Max, Min

For example, Laura may place the following complex sell order:

Item 1: (`Mustang`, `red`, 2001, 0), \$20,000

Item 2: (`Corvette`, `black`, 2001, 0), \$15,000

Size: 4, 2

This order means that Laura is selling four cars, each sale must include at least two cars, and every sold car must be either a red Mustang or a black Corvette. For example, Laura may sell four Mustangs and no Corvettes; alternatively, she may sell one Mustang and three Corvettes. She wants at least \$20,000 for each Mustang and

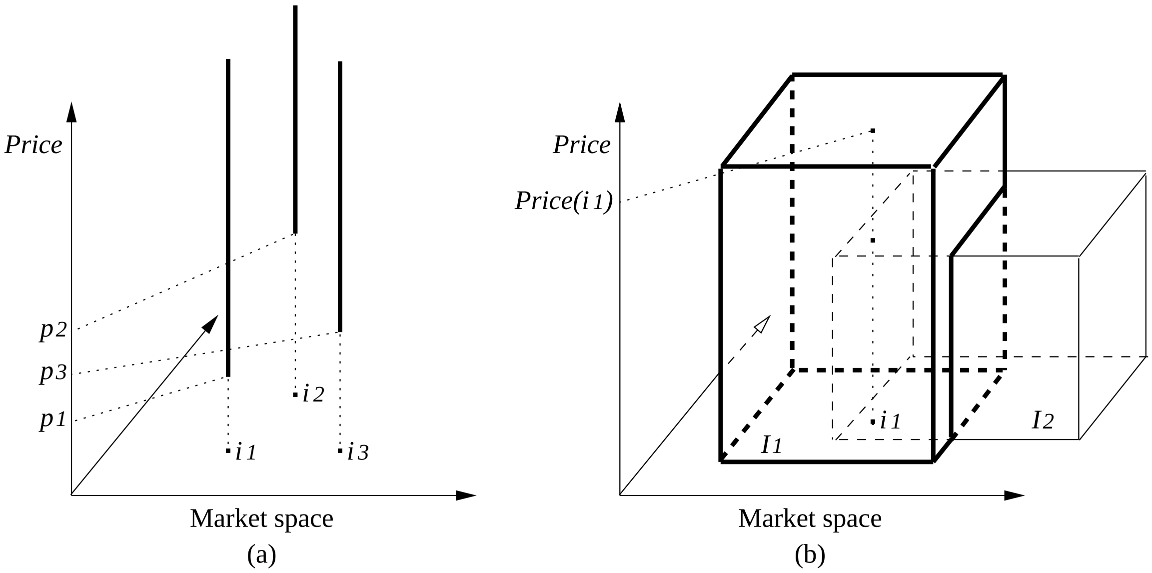


Figure 3.3: An example of a complex sell order (a) and complex buy order (b). A complex sell includes a list of items (*i1*, *i2*, and *i3*) and their prices (*p1*, *p2*, and *p3*), whereas a complex buy is a union of several Cartesian-product sets, along with a price limit for each set. Complex buy and sell orders are a special case of general orders, illustrated in Figure 2.1 (page 16).

at least \$15,000 for each Corvette; for instance, if selling one Mustang and three Corvettes, she must get at least $\$20{,}000 + \$15{,}000 \cdot 3 = \$65{,}000$.

A complex sell order is a special case of a general order, defined in Section 2.2. Recall that the general model allows any item set I and price function $Price\colon I \to \mathbf{R}^+$, as illustrated in Figure 2.1 (page 16). For a complex order, the item set consists of the specified items, and the price function is defined by the listed prices, as shown in Figure 3.3(a):

$$I = \{i1, i2, i3, \ldots, ik\}$$

$$Price(i1) = p1$$

$$\vdots$$

$$Price(ik) = pk$$

3.3.2 Complex Buys

The system includes a similar mechanism for complex buy orders. Recall that a buyer normally defines an item set as a Cartesian product (Section 3.1.2). When placing a complex order, a buyer specifies the item set as a union of several Cartesian products, $I = I1 \cup I2 \cup \ldots \cup Im$, and she may specify different prices for different Cartesian products. Formally, a complex buy order with m Cartesian products includes the following elements:

Subset 1: $I1_1 \times I1_2 \times \ldots \times I1_n,\ p1$

Subset 2: $I2_1 \times I2_2 \times \ldots \times I2_n,\ p2$

$\vdots$

Subset m: $Im_1 \times Im_2 \times \ldots \times Im_n,\ pm$

Size: Max, Min

For example, Katie may place the following buy order:

Subset 1: $\{\texttt{Mustang}, \texttt{Camry}\} \times \{\texttt{red}, \texttt{silver}\} \times \{1998, 1999, \ldots\} \times [0..30{,}000]$, \$25,000

Subset 2: $\texttt{Sports-Car} \times \{\texttt{silver}\} \times \{1999, 2000, \ldots\} \times [0..10{,}000]$, \$30,000

Size: 3, 1

In this case, she is buying three cars, where each car must belong to one of the subsets. Thus, she may buy a Mustang or Camry, as long as it is red or silver, made in 1998 or later, and has at most 30,000 miles (Subset 1). Alternatively, she will accept a sports car, if it is silver, made in 1999 or later, and has at most 10,000 miles (Subset 2).

The specified subsets do not have to be disjoint; for instance, a brand-new silver Mustang belongs to both parts of Katie's order. In this case, the system takes the largest of the specified prices; thus, it assumes that Katie is willing to pay \$30,000 for a silver Mustang.

We may describe a complex buy order as follows, in terms of the general model of Section 2.2:

$$I = I1 \cup I2 \cup \ldots \cup Im$$

$$\forall\ i1 \in I1,\ \ Price(i1) = p1$$

$$\forall\ i2 \in I2,\ \ Price(i2) = p2$$

$$\vdots$$

$$\forall\ im \in Im,\ \ Price(im) = pm$$

We illustrate this definition in Figure 3.3(b). Observe that, if some item belongs to multiple sets, the definition is ambiguous, as it assigns several prices to this item. In this case, we define the price as the maximum of subset prices; we show this situation for an item *i1* in Figure 3.3(b).

3.3.3 Matching

If buyers and sellers enter complex orders, the system has to identify matches among them and generate appropriate fills. Consider the following two complex orders:

Sell order: $(\{(i1_s, p1_s), (i2_s, p2_s), \ldots, (ik_s, pk_s)\}, Max_s, Min_s)$

Buy order: $(\{(I1_b, p1_b), (I2_b, p2_b), \ldots, (Im_b, pm_b)\}, Max_b, Min_b)$

These orders match if *some* element of the sell matches *some* element of the buy:

- There exist $j \in [1..k]$ and $l \in [1..m]$,
  such that $ij_s \in Il_b$ and $pl_b \geq pj_s$
- $Min_b \leq Max_s$ and $Min_s \leq Max_b$

If the two orders satisfy this condition, the matcher generates a fill with item ij_s, price $\frac{pj_s + pl_b}{2}$, and size $\min(Max_b, Max_s)$.

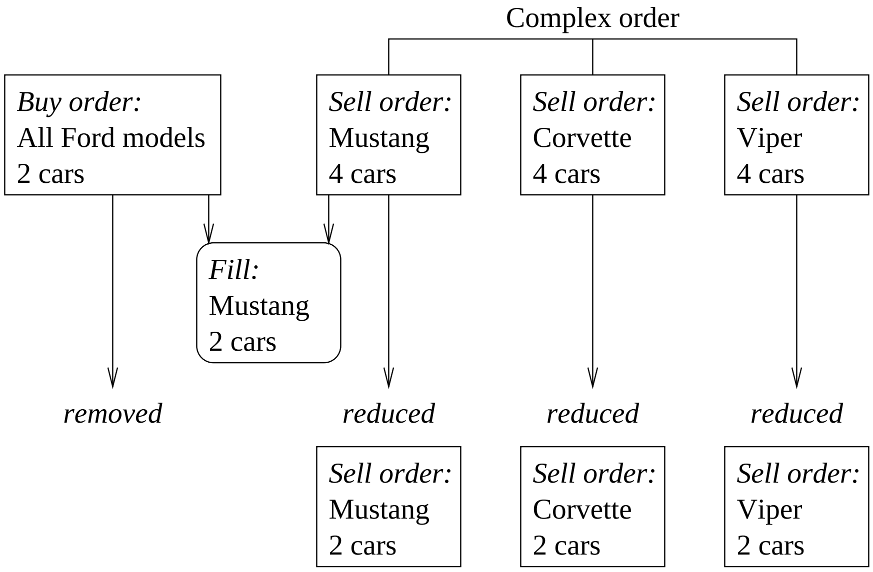


Figure 3.4: Example of a trade between a complex sell order and a simple buy. After generating a fill, the system reduces the size of all elements of the complex sell.

The system stores each element of a complex order as a separate simple order in the indexing structure. Thus, it represents a k-element sell order as k simple orders:

$$(i1_s, p1_s, Max_s, Min_s),\ (i2_s, p2_s, Max_s, Min_s), \ldots, (ik_s, pk_s, Max_s, Min_s).$$

Similarly, it represents an m-element complex buy as m buy orders:

$$(I1_b, p1_b, Max_b, Min_b),\ (I2_b, p2_b, Max_b, Min_b), \ldots, (Im_b, pm_b, Max_b, Min_b).$$

The matching process does *not* distinguish between simple orders and elements of a complex order; that is, it treats each element as a separate simple order.

When the system generates a fill for an element of a complex order, it reduces the size of all its elements; this synchronized reduction is the only difference between processing parts of a complex order and individual simple orders. In Figure 3.4, we illustrate a trade that involves a complex sell. In this example, Laura wants to sell four cars, where each car may be a Mustang, Corvette, or Viper. The system stores her order as three simple sell orders of the same size, connected by a complex-order link. Katie is looking for two cars produced by Ford, and she uses a predefined set "all Ford models" in her order. Since Mustang is a Ford, the system finds a match between the buy order and the "Mustang" element of Laura's sell. It generates the corresponding fill, and then reduces the size of all elements of the sell.

3.4 Confirmations

When placing an order, a trader may provide not only a description used in automated matching, but also additional information for human traders. For instance, Laura may post a picture of a specific car; as another example, when a company sells stocks or bonds, it has to publish a prospectus.

Since the automated matcher cannot process this information, it has to provide a mechanism that allows a trader to preview matching orders and choose among them. The implemented mechanism is based on *confirmations,* which enable a trader to browse through potential matches and find the most desirable choice.

When a user places an order, she may indicate the need for confirmation. In this case, when the system finds matches for the order, it sends their descriptions to the corresponding user interface. If the user confirms some of the matches, the engine executes the corresponding trades. For example, Katie may place an order to buy a brand-new silver Corvette, and require confirmation for her order. Then, she will be able to browse through matching Corvettes and hand-pick the best match.

If two matching orders do not need confirmation, the system executes the trade without prior notification to the user. If one of the orders needs confirmation, the system notifies the corresponding user and executes the trade upon getting a confirmation (see Figure 3.5a). If both orders require confirmation, the system notifies both sides and completes the trade only after getting both confirmations (see Figure 3.5b).

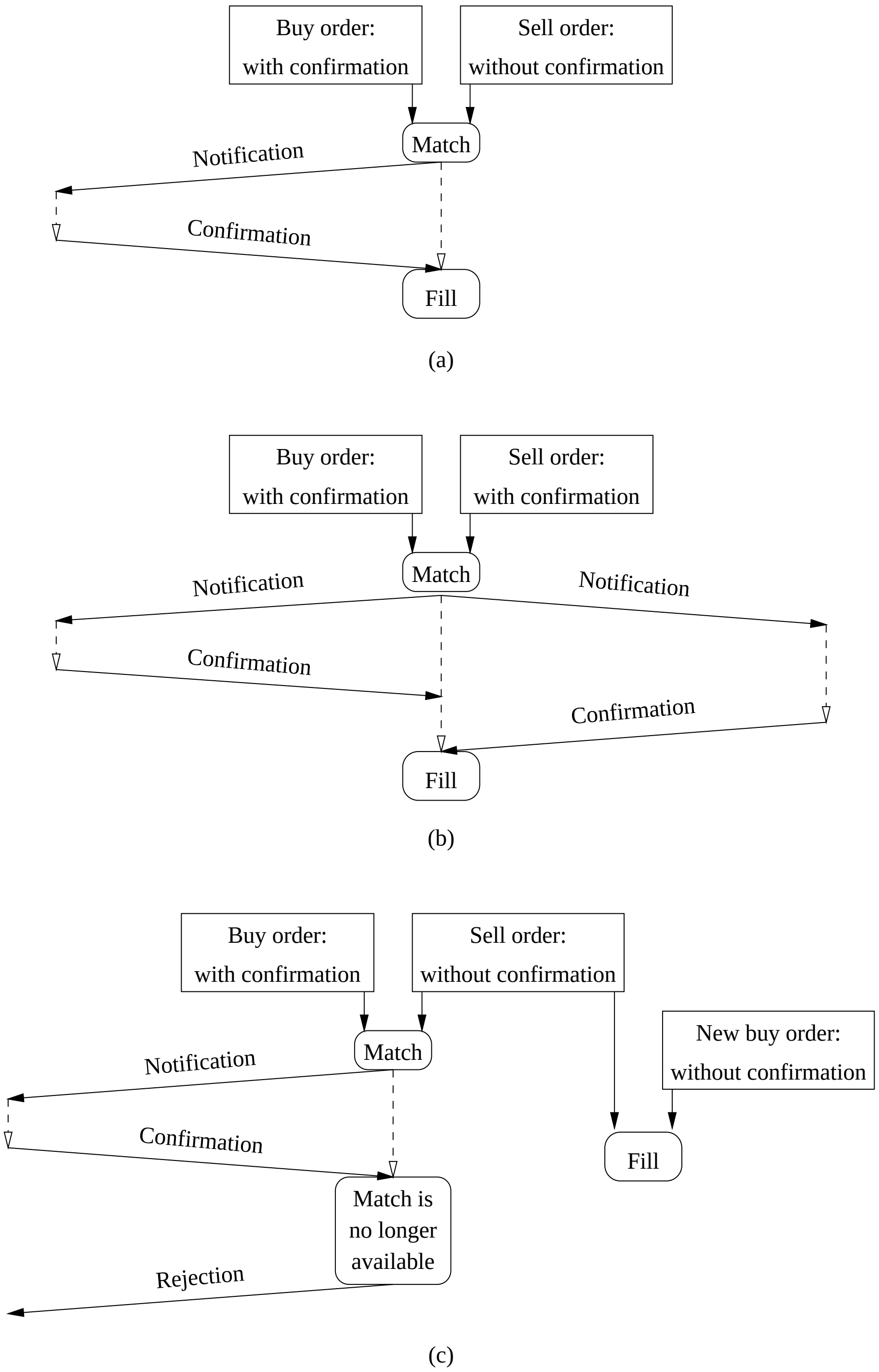


Figure 3.5: Trading with confirmations. If one of the matching orders needs confirmation, the system notifies the corresponding trader and waits for her approval (a). If both orders need confirmation, the system delays the trade until it receives approval from both parties (b). If the system finds an alternative match before getting an approval, it executes the corresponding trade and later rejects the confirmation (c).

For example, if Katie requests confirmation for her Corvette order, and Laura sells a Corvette that also needs confirmation, then the system will generate a fill only after getting confirmations from both Katie and Laura.

A trader may confirm several different matches for her order, which allows the system to execute any of them. For instance, Katie may confirm several different Corvettes, and then she will get one of them.

When the system asks users for confirmation, it does *not* remove either order from the matching process, and may find other matches for them. Thus, if a user thinks too long, she may be too late in sending a confirmation and miss a trade. In this case, the matcher notifies the user that the confirmed trade is no longer available (see Figure 3.5c). For instance, when Katie confirms a purchase of a specific Corvette, she may find out that someone else has bought it before her. As another example, if Laura is slow in sending her confirmation, she may learn that Katie has already purchased another car and cancelled her order.

The engine always tries to fill orders without confirmation before sending requests for confirmation. This heuristic improves the speed of the trading process and reduces the number of "late" confirmations.

Chapter 4

Performance

We describe experiments with artificial market data, and then show the performance for two real-world markets. The experiments have involved a single computer, without network communications, and thus the results do not account for network delays.

4.1 Artificial Markets

We give results of artificial tests with five control variables. We measure the matcher's efficiency for different numbers of market attributes and values per attribute, as well as the dependency of the performance on the number of orders.

4.1.1 Control Variables

We have implemented an experimental setup that allows control over five features of artificial markets: number of attributes in an item description, number of alternative values for an attribute, number of pending orders, rate of incoming orders, and average number of matches per order.

Attributes: The number of attributes determines the complexity of traded items. For example, the description of a public stock includes only one attribute, the stock symbol, whereas the example car market has four attributes: model, color, year, and mileage. The description of a used vehicle in a real market includes more attributes, such as transmission type, options package, and number of previous owners. We have considered artificial markets with one, three, ten, thirty, and one hundred attributes.

Values: The implemented system supports integer and real attributes, as well as explicitly listed values (see Section 2.6). We have considered only integer attributes in the artificial markets, and we have controlled the number of values per attribute. We have varied this number from two to 1,024; that is, the smallest market has two values for each attribute, and the largest market includes 1,024 values per attribute.

Pending orders: We have varied the number of pending orders from one to 2^{18}, that is, 262,144, and measured the dependency of the system's efficiency on the number of orders. We have randomly generated these orders, which include an equal number of buys and sells.

New orders: Recall that the system's top-level loop involves processing new orders and matching old orders (see Figure 3.2a). The number of new orders in the beginning of the loop is proportional to the rate of placing new orders, and we have directly controlled the number of new orders, which includes an equal number of buys and sells. We have experimented with 256, 8,192, and 262,144 new orders, which are powers of two, 2^8, 2^{13}, and 2^{18}.

Matching density: We define 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. For example, if each buy order matches 1% of sell orders, then the matching density is 0.01. We have experimented with five density values: 0.0001, 0.001, 0.01, 0.1, and 1.

4.1.2 Measured Variables

We have considered five different settings for the number of attributes, six settings for the number of values per attribute, nineteen settings for the number of pending orders, three settings for the number of new orders, and five settings for the matching density. For each combination of settings, we have run two independent experiments; for each experiment, we have measured the time of processing new orders and matching old orders, average time between placing an order and getting a response, and maximal throughput of the system.

Processing time: The first measurement is the time of processing new orders, which is the first part of the system's main loop (see Figure 3.2a). This time is proportional to the number of new orders; it also depends on several other factors, including the number of attributes and pending orders.

Matching time: The second measurement is the time of matching old orders, which is the second part of the main loop (Figure 3.2a). The total of processing and matching time is the overall length of the main loop, which determines the system's speed.

Response time: We have recorded the average time between placing an order and getting the system's response. If the system immediately finds a match for the new order, then its response is the corresponding fill; else, it responds with a confirmation message. The response time has varied from a few seconds to more than twenty minutes. This delayed response is often unacceptable in financial markets, but it would not cause problems in consumer markets, such as used cars.

Maximal throughput: Finally, we have determined the maximal acceptable frequency of placing new orders; if the system gets more orders per second, then the number of unprocessed orders in the input queue keeps growing and eventually leads to an overflow.

4.1.3 Summary Graphs

We give a detailed summary of experiments in Appendices A and B. The results in Appendix A (page 80) are for markets with one, three, and ten attributes. The graphs in Appendix B (page 124) are for more complex markets, which include thirty and one hundred attributes. We show the four measurements for each of the settings, including the minimal, maximal, and mean values of these measurements.

In Figures 4.1–4.5, we show the dependency of the system's performance on each of the control variables. We use three performance measurements: (1) the time of one pass through the system's main loop, which includes processing of new orders and matching of old orders; (2) the mean time between placing a new order and getting a response; and (3) the system's throughput.

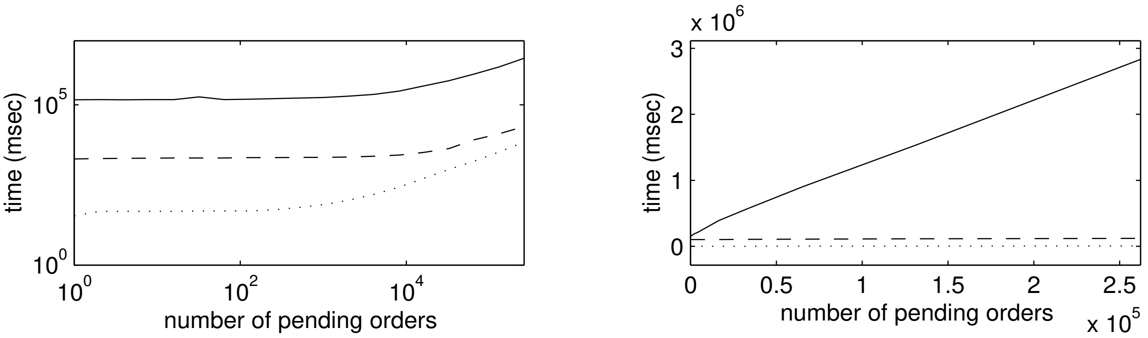
For each control variable, we consider three settings of the other four variables. For each of the three settings, we give two graphs with the dependency of the performance on the selected variable; the first graph is in logarithmic scale, and the second is in linear scale.

In Figure 4.1, we show how the performance changes with the number of pending orders. 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 an upper limit when the market grows to about two hundred orders, and slightly decreases with further increase in the number of orders.

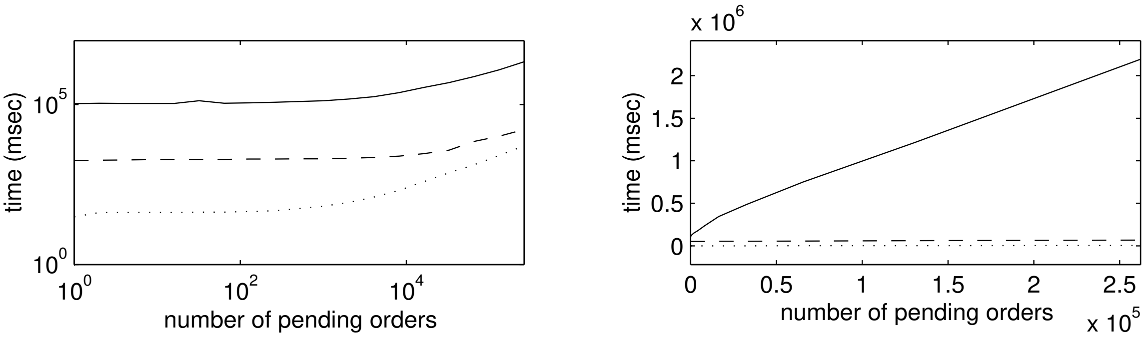
In Figure 4.2, we show that both main-loop time and response time linearly increase with the number of new orders. We do not show the dependency of throughput on the number of new orders, because the rate of incoming orders directly depends on the throughput, and we cannot treat this rate as an independent control variable.

In Figure 4.3, we give the dependency of the performance on the number of attributes. The main-loop and response times are super-linear in the number of attributes, whereas throughput is in inverse proportion to the same super-linear function.

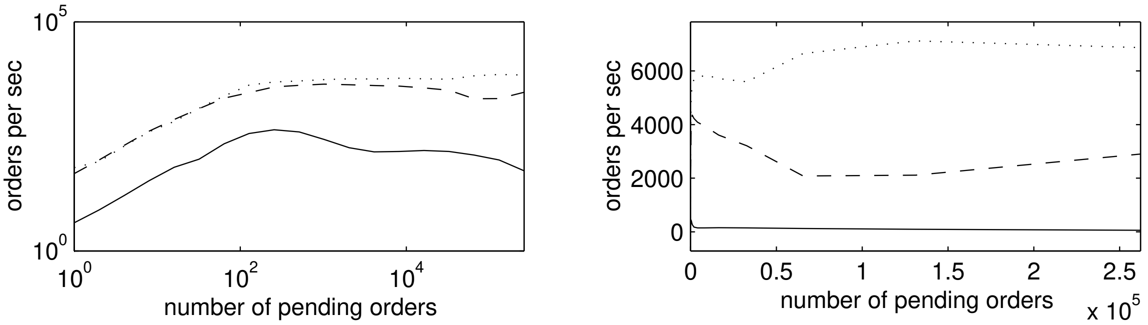
In Figure 4.4, we show that the main-loop and response times grow sub-linearly with the number of values per attribute, and throughput slightly decreases with an increase in the number of values. Finally, the graph in Figure 4.5 shows a linear relationship between the main-loop time and matching density.



(a) Time of processing new orders and matching old orders.

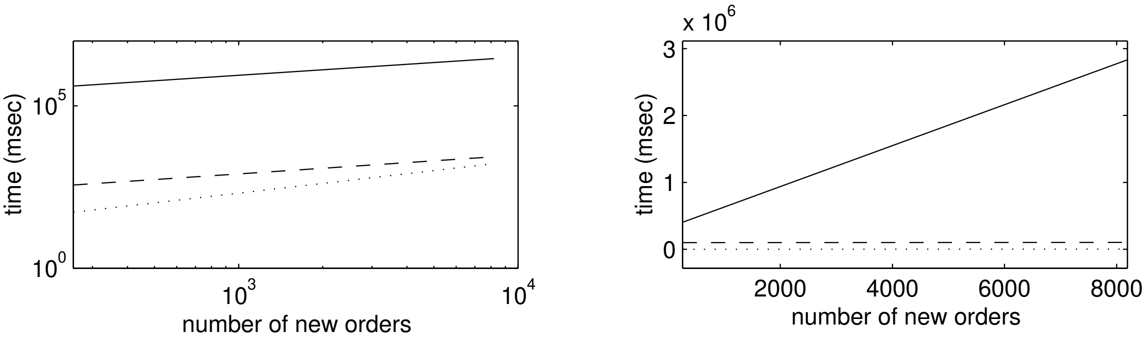


(b) Average time between order placement and response.

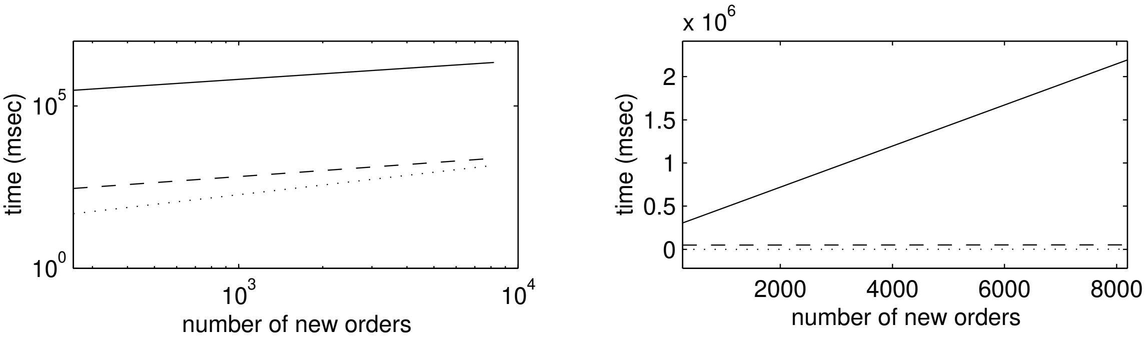


(c) Maximal number of new orders per second.

Figure 4.1: Dependency of the system's performance on the *number of pending orders.* We consider three different settings of control variables, which correspond to dotted, dashed, and solid lines. The dotted lines show performance for markets with one attribute, two values per attribute, 256 new orders in the input queue, and matching density of 0.0001. The dashed lines are for markets with three attributes, sixteen values per attribute, 8,192 new orders, and matching density of 0.001. Finally, the solid lines are for ten attributes, 1,024 values per attribute, 8,192 orders in the input queue, and matching density of 0.01. The graphs on the left are in logarithmic scale, whereas the graphs on the right are in linear scale.

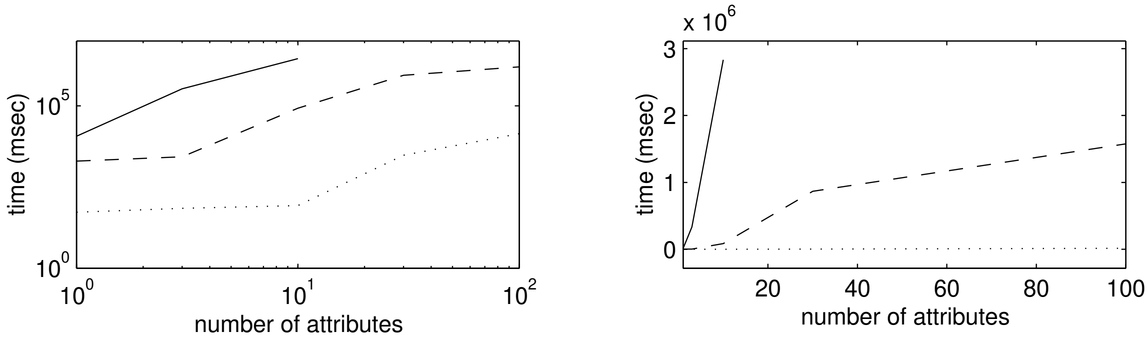


(a) Time of processing new orders and matching old orders.

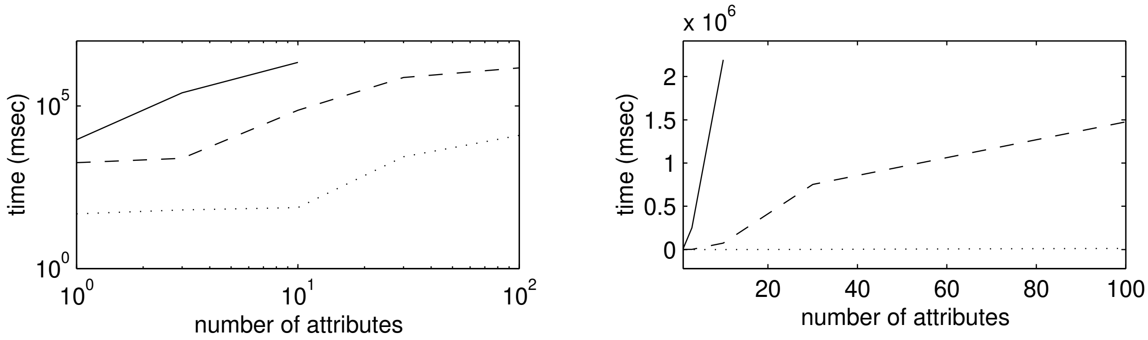


(b) Average time between order placement and response.

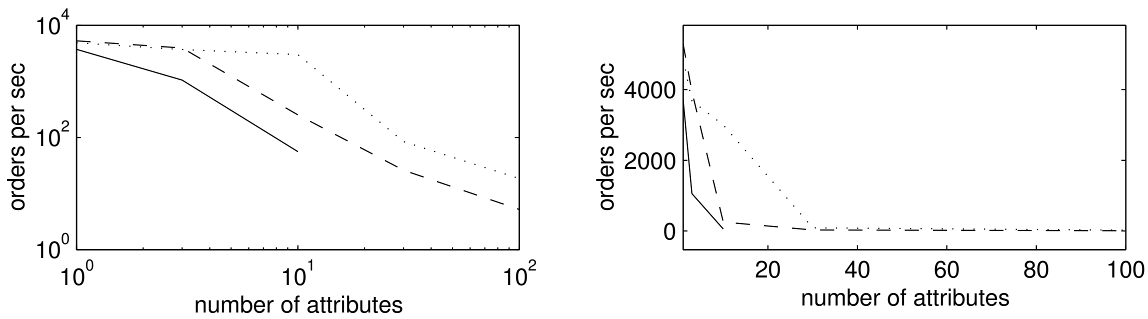
Figure 4.2: Dependency of the performance on the *number of new orders* in the input queue. Note that we do not plot this dependency for the system's throughput, because the rate of incoming orders directly depends on the throughput, and thus we cannot view the number of new orders as an independent control variable. The dotted lines give the results for one attribute, two values per attribute, 256 pending orders, and matching density of 0.0001. The dashed lines are for three attributes, sixteen values per attribute, 8,192 pending orders, and matching density of 0.001. Finally, the solid lines are for ten attributes, 1,024 values per attribute, 262,144 pending orders, and matching density of 0.01. The graphs on the left are in logarithmic scale, whereas the graphs on the right are in linear scale.



(a) Time of processing new orders and matching old orders.

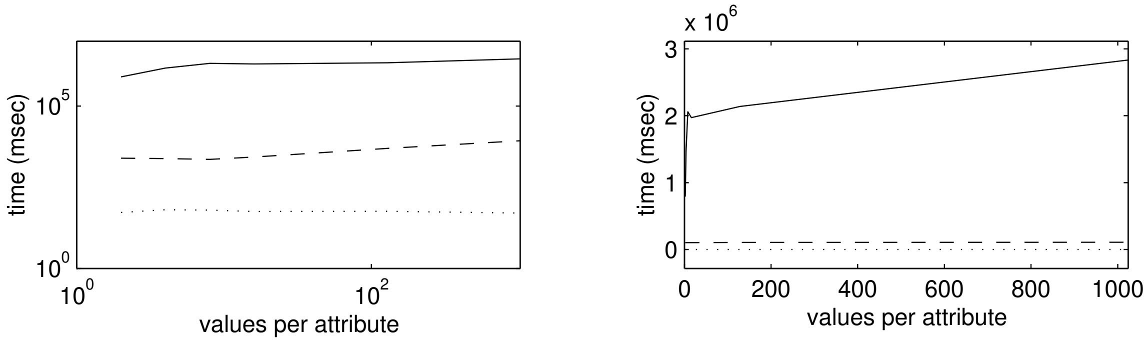


(b) Average time between order placement and response.

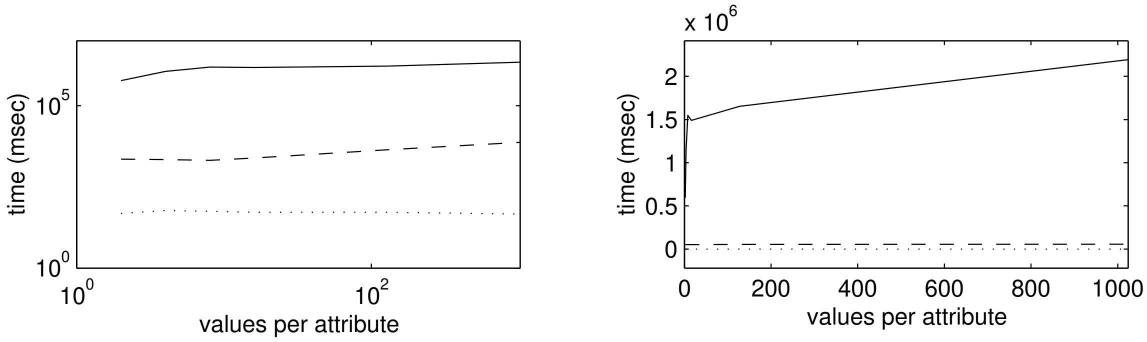


(c) Maximal number of new orders per second.

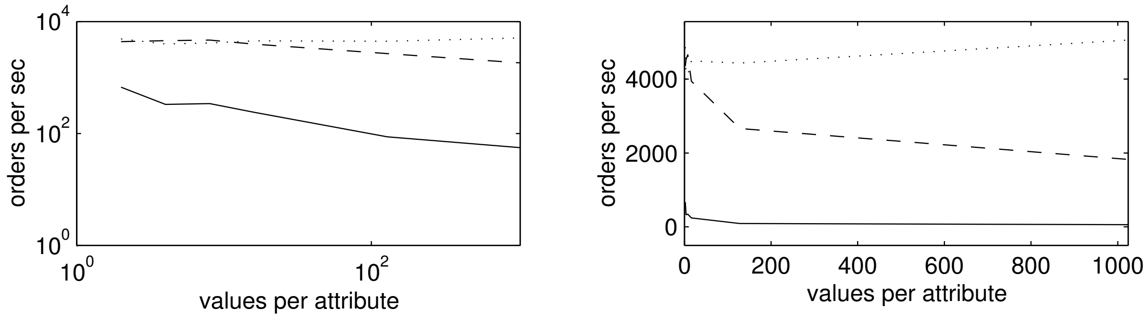
Figure 4.3: Dependency of the performance on the *number of market attributes.* The dotted lines show experiments with two values per attribute, 256 pending orders, 256 new orders, and matching density of 0.0001. The dashed lines are for sixteen values per attribute, 8,192 pending orders, 8,192 new orders, and matching density of 0.001. The solid lines are for 1,024 values per attribute, 262,144 pending orders, 8,192 new orders, and matching density of 0.01. The graphs on the left are in logarithmic scale, whereas the graphs on the right are in linear scale. Note that we do not plot the solid lines for 30 and 100 attributes, since we have not been able to run the corresponding experiments, which would require more than 1 Gbyte main memory.



(a) Time of processing new orders and matching old orders.

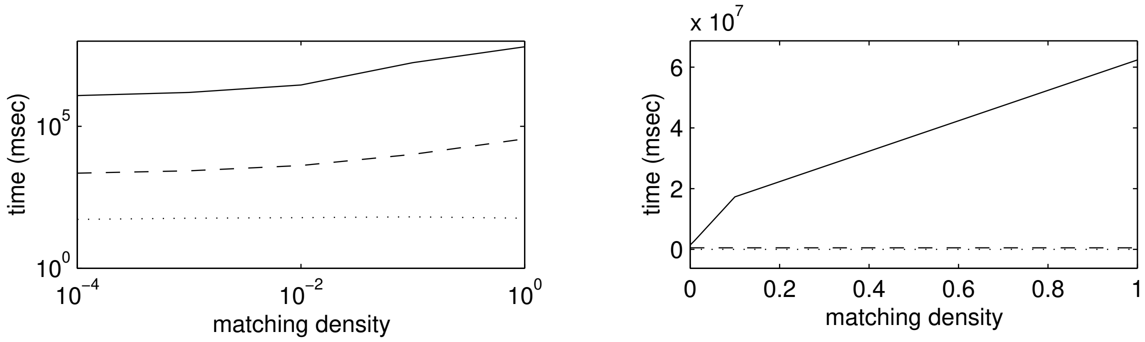


(b) Average time between order placement and response.

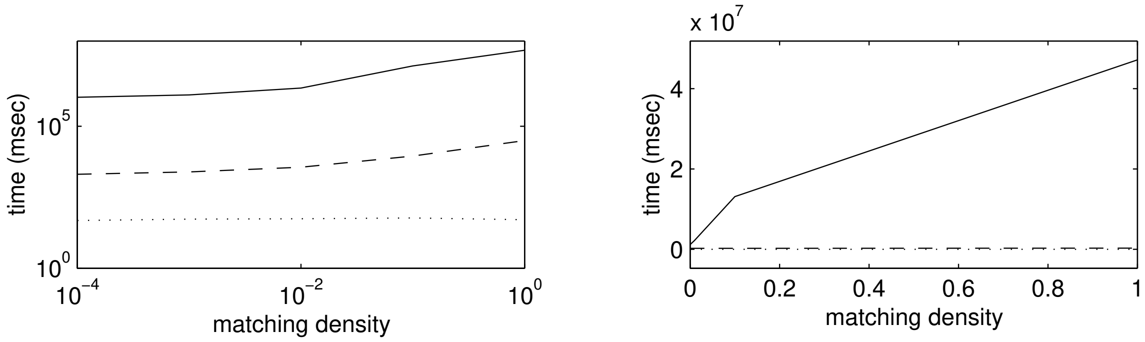


(c) Maximal number of new orders per second.

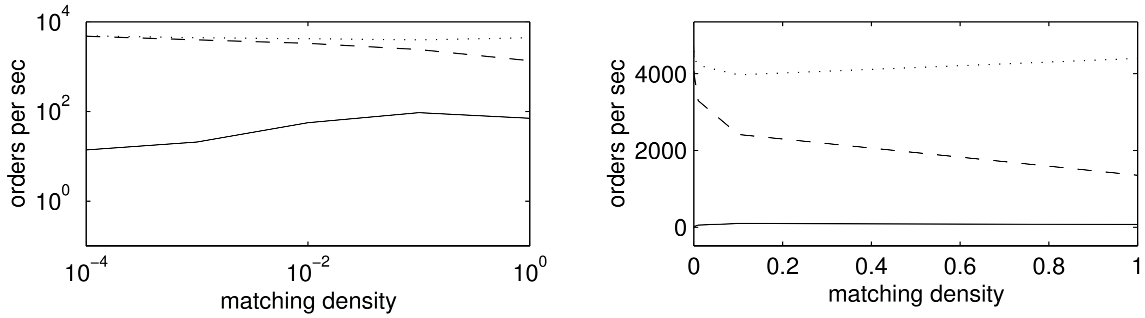
Figure 4.4: Dependency of the performance on the *number of values per attribute*. The dotted lines show experiments with one attribute, 256 pending orders, 256 new orders, and matching density of 0.0001. The dashed lines are for three attributes, 8,192 pending orders, 8,192 new orders, and matching density of 0.001. The solid lines are for ten attributes, 262,144 pending orders, 8,192 new orders, and matching density of 0.01. The graphs on the left are in logarithmic scale, whereas the graphs on the right are in linear scale.



(a) Time of processing new orders and matching old orders.



(b) Average time between order placement and response.



(c) Maximal number of new orders per second.

Figure 4.5: Dependency of the performance on the *matching density*. The dotted lines show experiments with one attribute, two values per attribute, 256 pending orders, and 256 new orders. The dashed lines are for three attributes, sixteen values per attribute, 8,192 pending orders, and 8,192 new orders. The solid lines are for ten attributes, 1,024 values per attribute, 262,144 pending orders, and 8,192 new orders. The graphs on the left are in logarithmic scale, whereas the graphs on the right are in linear scale.

4.2 Real Markets

The results of experiments with real-world markets have been similar to those of artificial tests. First, we have applied the system to a used-car market with eight attributes, which is an extended version of the example market. Second, we have experimented with a financial market, which involves trading of commercial paper.

4.2.1 Used Cars

We have tested the system on a car market with eight attributes: transmission, number of doors, interior color, exterior color, year, model, options, and mileage (see Figures 4.6 and 4.7). The market includes all models offered by AutoNation (www.autonation.com); it comprises seven interior colors, fifty-two exterior colors, 257 models, and 1,024 option packages. We have also defined three standard sets of interior colors, three sets of exterior colors, and forty-three sets of models.

We have run experiments with up to 260,000 pending orders; the control variables have included the number of pending orders, the number of new orders in the input queue, and the matching density. We have considered nineteen different settings for the number of pending orders, two settings for the number of new orders, and five settings for matching density. For each combination of settings, we have run two experiments, and we show the results in Figures 4.8–4.12. Observe that the system readily scales to markets with 260,000 orders, and that the performance in the car market is very similar to that in artificial markets.

In Figures 4.13–4.15, we show the dependency of the system's performance on each of the control variables. The main-loop and response times are linear in the number of orders, whereas throughput first increases and then slightly decreases with the number of orders. The system parses 500 to 1,000 orders per second, and generates 250 to 500 trades per second.

Attribute 1: Transmission (2 values)
Manual, automatic.

Attribute 2: Number of doors (3 values)
Two, three, four.

Attribute 3: Interior color (7 values)
Black, gray, white, tan, brown, blue, red.

Standard sets
Dark (2 values)
Medium (3 values)
Light (2 values)

Attribute 4: Exterior color (52 values)

Amazon Green	Dark Green Satin	Light Gray
Arizona Beige	Dark Toreador Red	Light Parchment Gold
Atlantic Blue	Deep Emerald Green	Light Sapphire Blue
Autumn Orange	Deep Jewel Green	Malibu Blue
Autumn Red	Deep Wedgewood Blue	Mandarin Gold
Black	Ebony	Medium Brown
Bright Amber	Electric Green	Medium Charcoal Blue
Bright Atlantic Blue	Estate Green	Medium Charcoal Green
Bright Red	Fort Knox Gold	Medium Gray
Bright Silver	Graphite Blue	Medium Royal Blue
Cabernet Red	Harvest Gold	Medium Steel Blue
Charcoal Green	Infra Red	Medium Titanium
Chesapeake Blue	Island Blue	Medium Wedgewood Blue
Chestnut	Ivory Parchment	Midnight Gray
Chrome Yellow	Jewel Green	Performance Red
Cloud White	Laser Red Tinted	Silver
Crystal White	Light Blue	
Dark Blue	Light Brass	

Standard sets
Dark (18 values)
Medium (20 values)
Light (14 values)

Figure 4.6: Attributes 1–4 of the used-car market (also see Figure 4.7).

Attribute 5: Year (106 values)
Integer values from 1896 to 2001.

Attribute 6: Model (257 values)
All models offered by AutoNation (www.autonation.com).

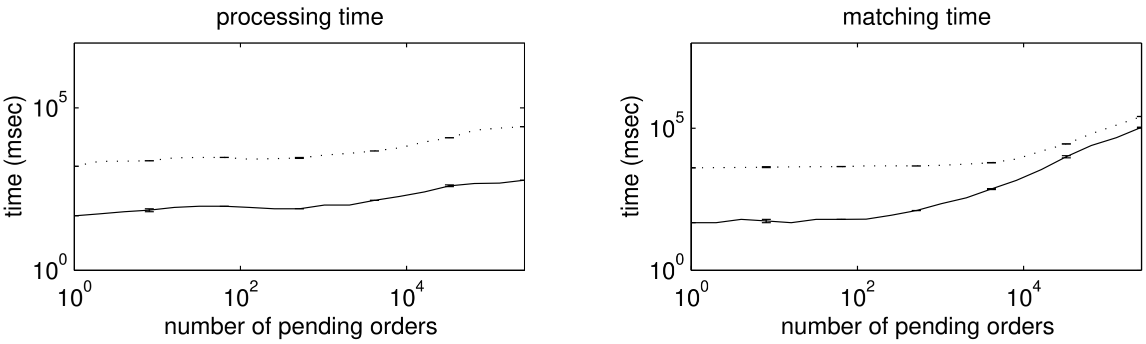
Standard sets

American (138 values)	Chevrolet (26 values)	Mazda (11 values)
European (48 values)	Chrysler (9 values)	Mercedes Benz (9 values)
Japanese (71 values)	Dodge (12 values)	Mercury (5 values)
	Ford (19 values)	Mitsubishi (6 values)
Sedans (139 values)	GMC (10 values)	Nissan (8 values)
Sports Cars (28 values)	Honda (8 values)	Oldsmobile (5 values)
SUVs (57 values)	Hyundai (6 values)	Plymouth (5 values)
Trucks (23 values)	Infiniti (4 values)	Pontiac (8 values)
Vans (10 values)	Isuzu (4 values)	Porsche (2 values)
	Jaguar (4 values)	Saturn (6 values)
Acura (6 values)	Jeep (4 values)	Subaru (3 values)
Audi (6 values)	Kia (5 values)	Suzuki (4 values)
BMW (7 values)	Land Rover (2 values)	Toyota (16 values)
Buick (5 values)	Lexus (8 values)	Volkswagen (7 values)
Cadillac (5 values)	Lincoln (5 values)	Volvo (7 values)

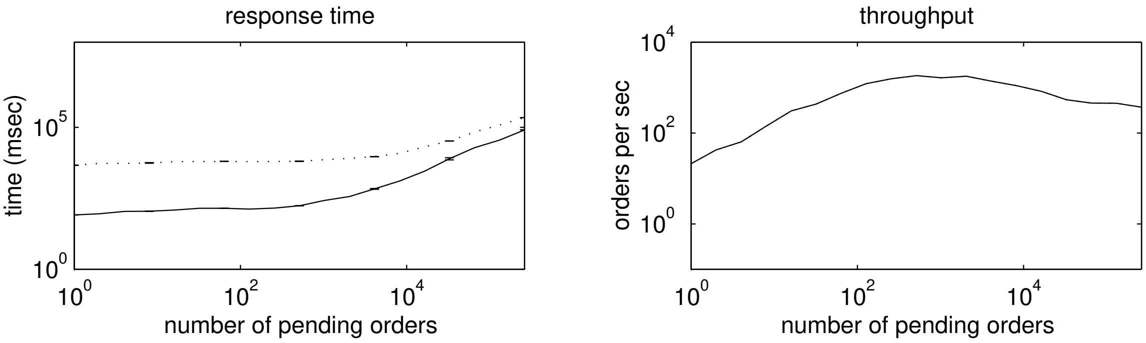
Attribute 7: Option package (1,024 values)
Standard combinations of options offered by AutoNation.

Attribute 8: Mileage (500,001 values)
Integer values from 0 to 500,000.

Figure 4.7: Attributes 5–8 of the used-car market (also see Figure 4.6).

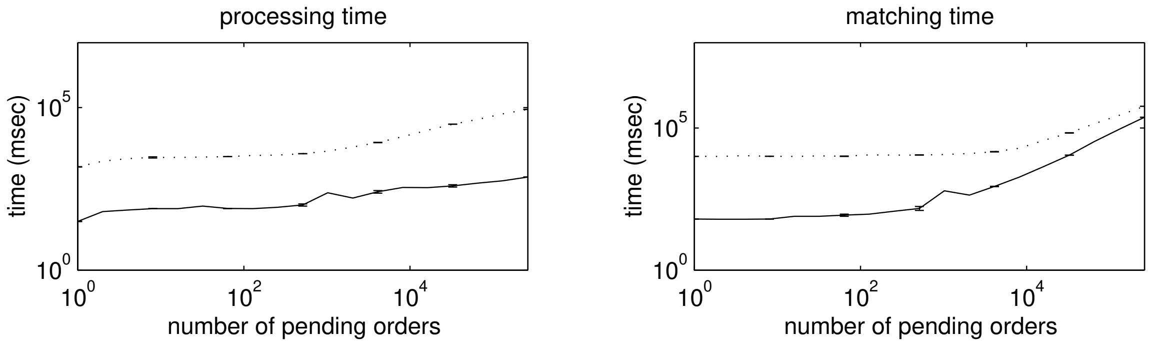


(a) Time of processing new orders (left) and matching old orders (right).

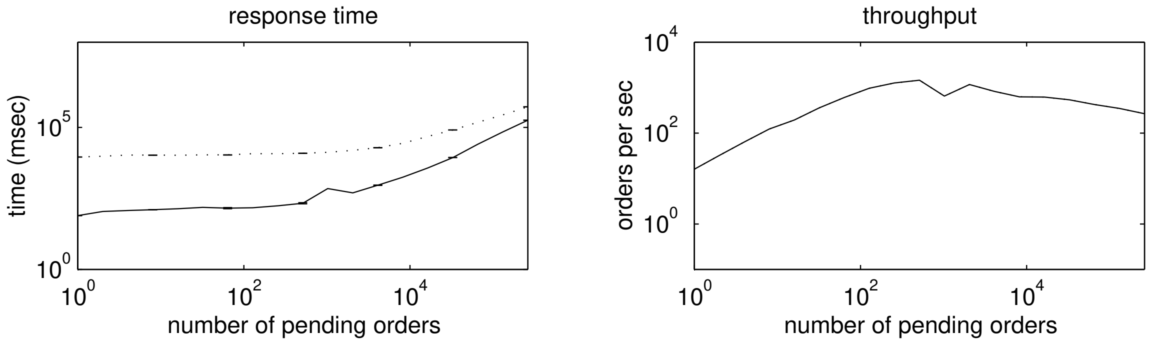


(b) Response time (left) and throughput (right).

Figure 4.8: Performance in the car market, for *matching density of 0.0001.* We show the dependency of the system's performance on the number of pending orders, for 256 new orders (solid lines) and 8,192 new orders (dotted lines). Both horizontal and vertical scales are logarithmic, and the vertical bars mark the minimal and maximal values of the time measurements.

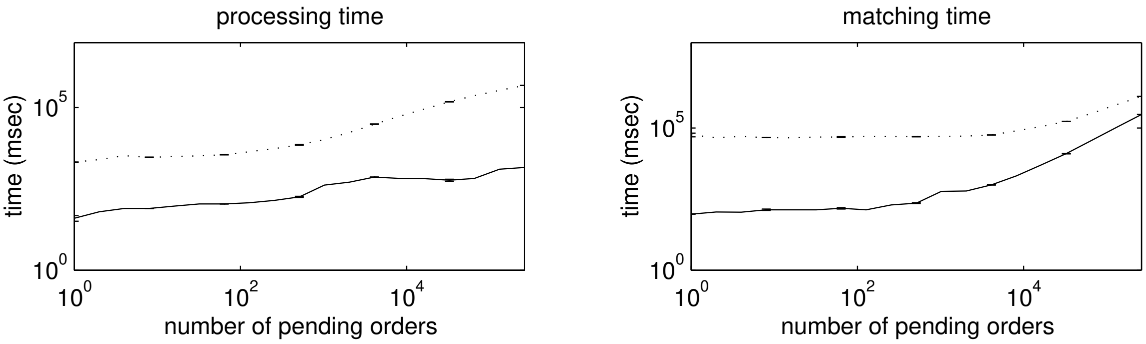


(a) Time of processing new orders (left) and matching old orders (right).

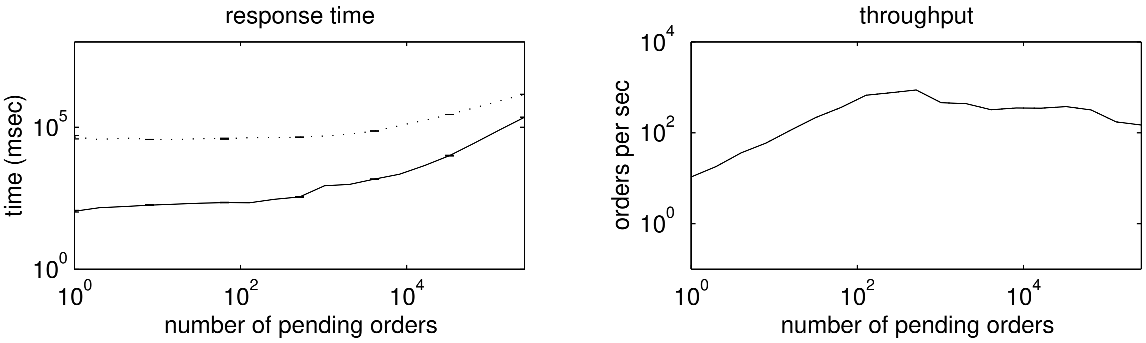


(b) Response time (left) and throughput (right).

Figure 4.9: Performance in the car market, for *matching density of 0.001.* We show results for 256 new orders (solid lines) and 8,192 new orders (dotted lines), with the minimal and maximal values (vertical bars).

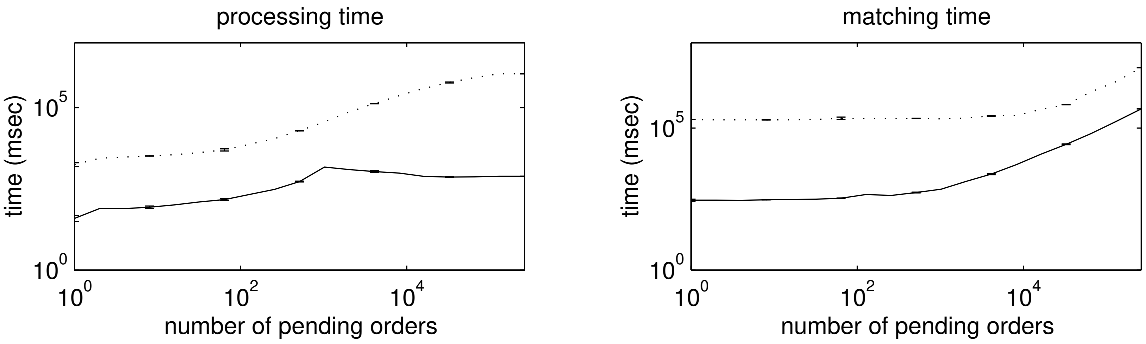


(a) Time of processing new orders (left) and matching old orders (right).

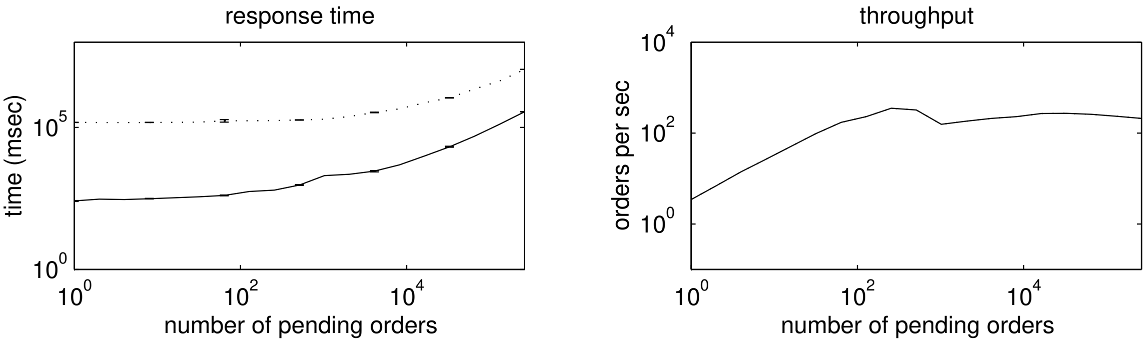


(b) Response time (left) and throughput (right).

Figure 4.10: Performance in the car market, for *matching density of 0.01.* We show results for 256 new orders (solid lines) and 8,192 new orders (dotted lines), with the minimal and maximal values (vertical bars).

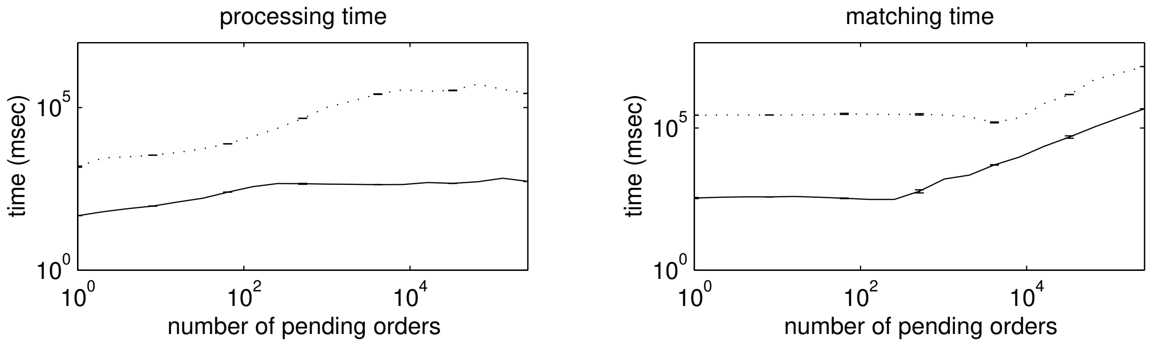


(a) Time of processing new orders (left) and matching old orders (right).

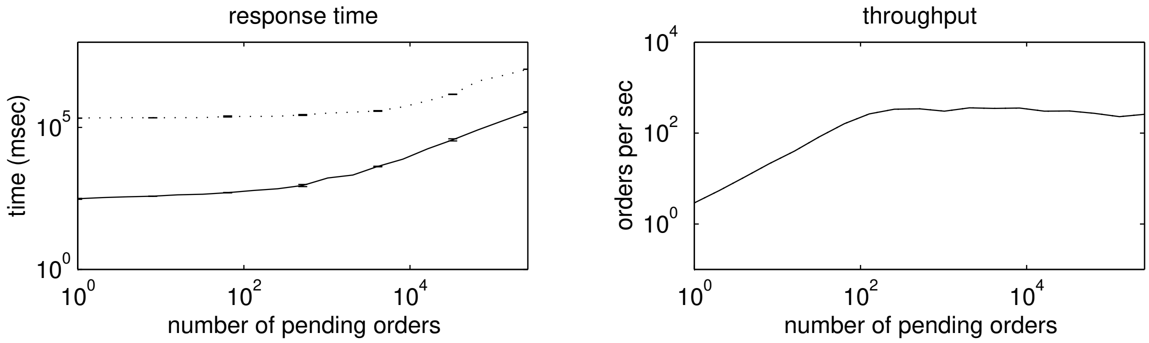


(b) Response time (left) and throughput (right).

Figure 4.11: Performance in the car market, for *matching density of 0.1.* We show results for 256 new orders (solid lines) and 8,192 new orders (dotted lines), with the minimal and maximal values (vertical bars).

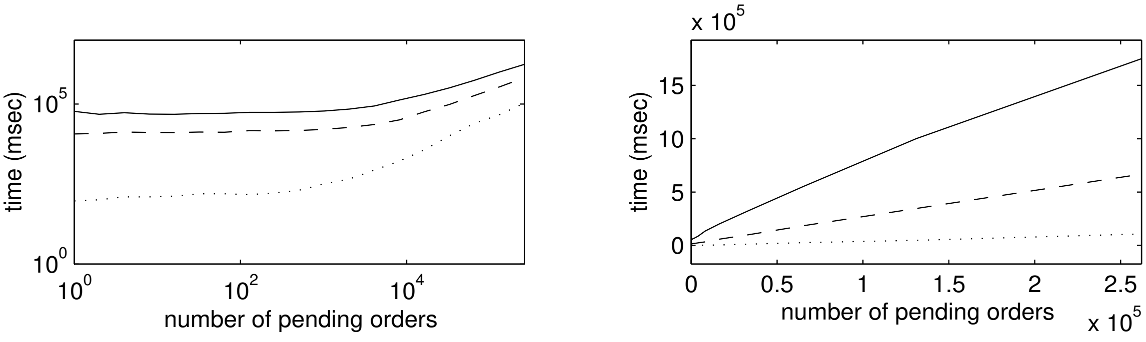


(a) Time of processing new orders (left) and matching old orders (right).

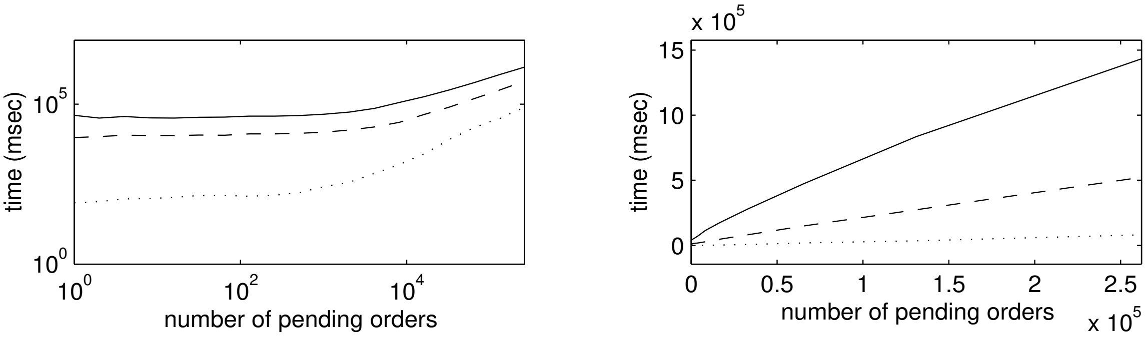


(b) Response time (left) and throughput (right).

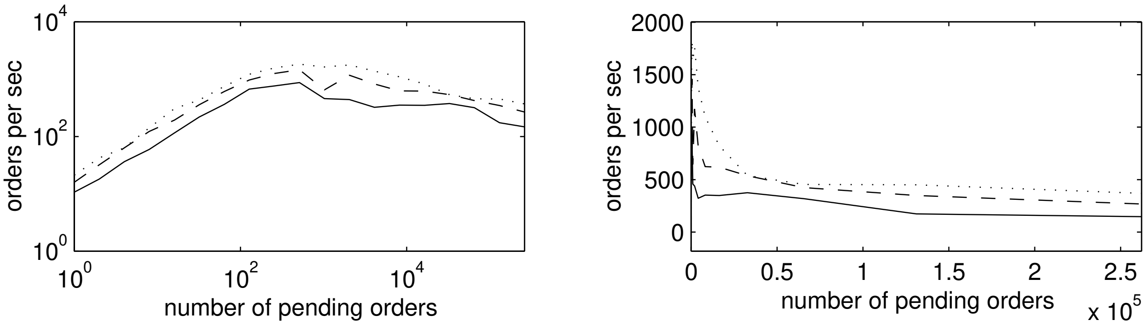
Figure 4.12: Performance in the car market, for *matching density of 1.* We show results for 256 new orders (solid lines) and 8,192 new orders (dotted lines), with the minimal and maximal values (vertical bars).



(a) Time of processing new orders and matching old orders.

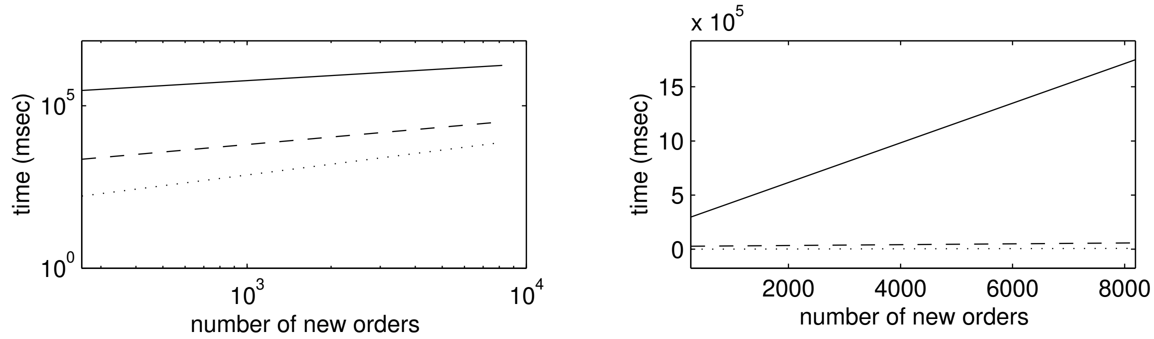


(b) Average time between order placement and response.

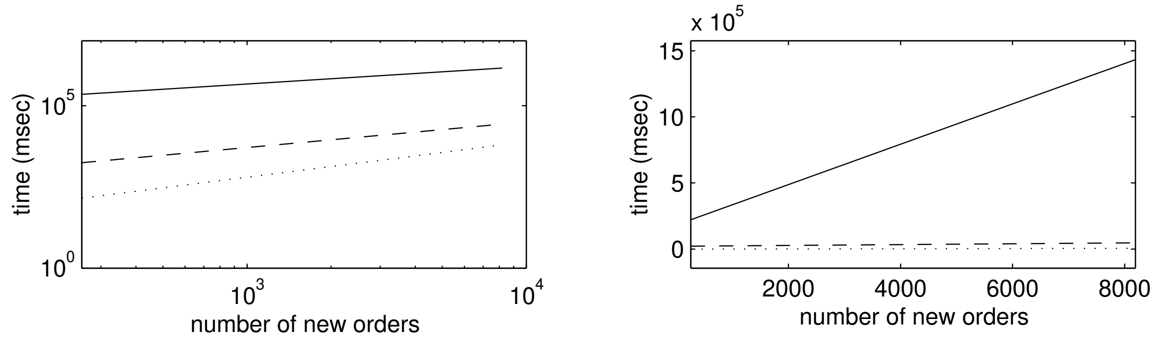


(c) Maximal number of new orders per second.

Figure 4.13: Dependency of the performance on the *number of pending orders,* for the car market. The dotted lines show experiments with 256 new orders and matching density of 0.0001. The dashed lines are for 8,192 new orders and matching density of 0.001. The solid lines are for 8,192 new orders and matching density of 0.01. The graphs on the left are in logarithmic scale, whereas the graphs on the right are in linear scale.

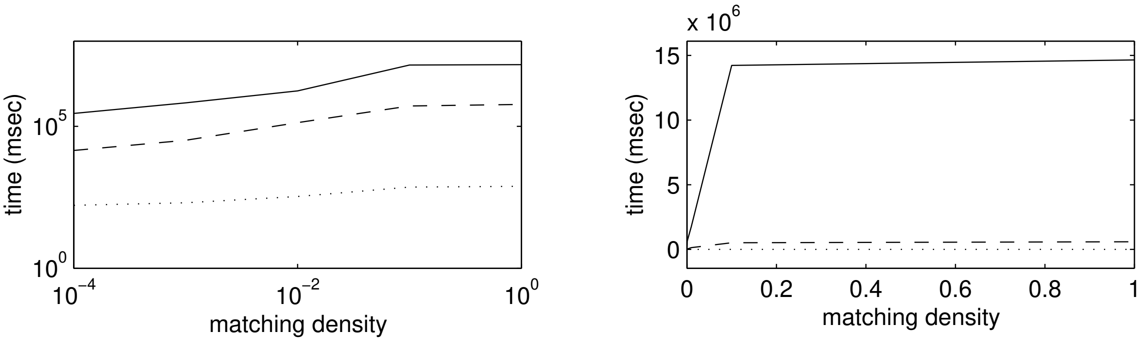


(a) Time of processing new orders and matching old orders.

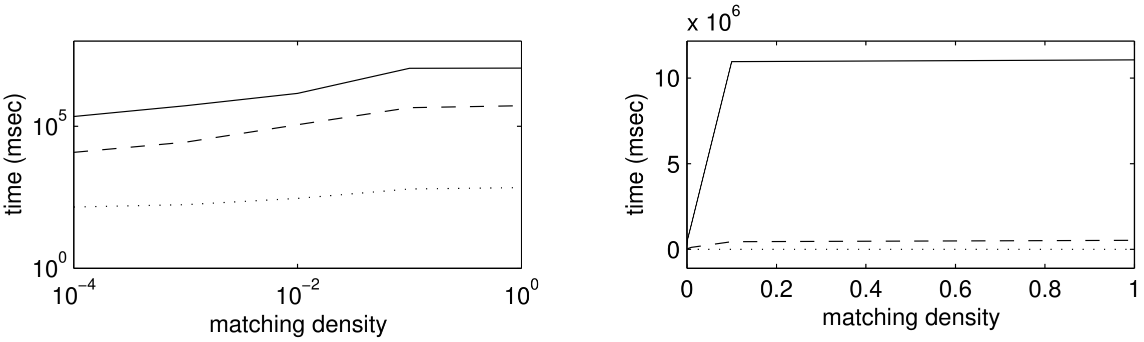


(b) Average time between order placement and response.

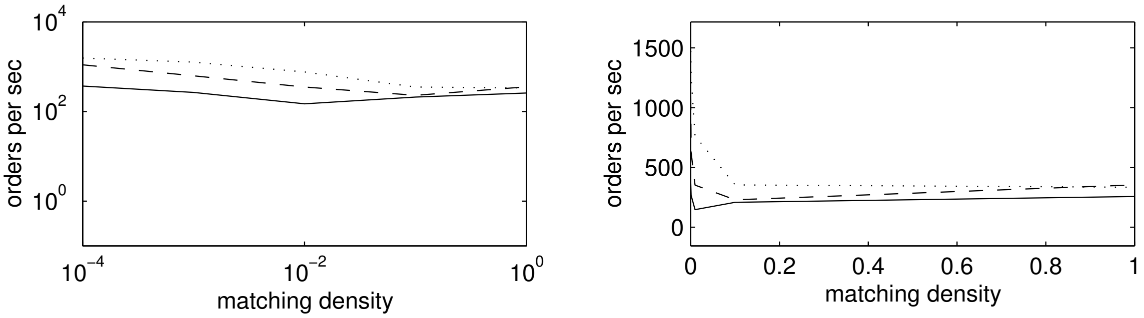
Figure 4.14: Dependency of the performance on the *number of new orders,* for the car market. The dotted lines show experiments with 256 pending orders and matching density of 0.0001. The dashed lines are for 8,192 pending orders and matching density of 0.001. The solid lines are for 262,144 pending orders and matching density of 0.01. The graphs on the left are in logarithmic scale, whereas the graphs on the right are in linear scale. We do not plot this dependency for the throughput, because the rate of incoming orders directly depends on the throughput, and thus we cannot view the number of new orders as an independent control variable.



(a) Time of processing new orders and matching old orders.



(b) Average time between order placement and response.



(c) Maximal number of new orders per second.

Figure 4.15: Dependency of the performance on the *matching density,* for the car market. The dotted lines show experiments with 256 pending orders and 256 new orders. The dashed lines are for 8,192 pending orders and 8,192 new orders. The solid lines are for 262,144 pending orders and 8,192 new orders. The graphs on the left are in logarithmic scale, whereas the graphs on the right are in linear scale.

4.2.2 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, for a certain period of time, and later returns their money along 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.

Investors usually describe the income from commercial paper in terms of the annual interest rate. For example, suppose that a company has issued a seven-day commercial paper, in the amount of \$10,000,000, with annual interest 10%. Then, the daily interest is $10\%/365 = 0.0274\%$, and the total amount of interest is about $\$10{,}000{,}000 \cdot 0.0274\% \cdot 7 = \$19{,}180$. The exact amount is slightly higher, \$19,197, because of compound interest.

The appropriate interest depends on the current rate of US Treasury bonds, and on the chance of a company's bankruptcy before the maturity date. The estimated chance of bankruptcy depends on the company's reputation and the paper's time until maturity. The investors expect less reliable companies to pay higher interest; furthermore, they expect that, the more time to maturity, the higher the interest. For example, a risky software provider should pay higher interest than a reliable utilities company, and a nine-month paper should carry higher interest than a one-month paper of the same company.

After investors buy a commercial paper, they may resell it on a secondary market, before the maturity date. For example, suppose that Katie has bought a three-month paper in May, and then decided that she needs money in June. Then, she may resell the paper and keep part of the interest; if the rate has not changed, she will get one-month interest. On the other hand, if the interest rate of the company's paper

Attribute 1: Maturity date (2550 values)
All dates from 1/1/2001 to 12/31/2010, excluding weekends and holidays.

Attribute 2: Company (5000 values)
5000 US companies.

Standard sets

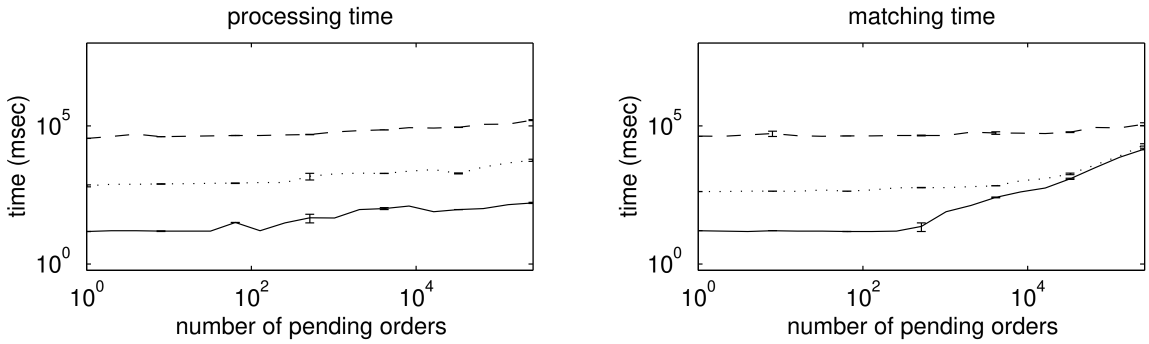
Automobiles (100 values)	S&P AA+ rating (535 values)
Banks (250 values)	S&P AA rating (535 values)
Capital Goods (500 values)	S&P AA− rating (535 values)
Consumer Durables (200 values)	S&P A+ rating (758 values)
Diversified Financials (200 values)	S&P A rating (759 values)
Energy (250 values)	S&P A− rating (758 values)
Food and Tobacco (350 values)	S&P BBB+ rating (356 values)
Health Care (150 values)	S&P BBB rating (357 values)
Hotels and Restaurants (100 values)	S&P BBB− rating (357 values)
Household Products (150 values)	
Insurance (150 values)	Moody Aaa rating (100 values)
Materials (450 values)	Moody Aa+ rating (441 values)
Media (100 values)	Moody Aa rating (441 values)
Pharmaceuticals (250 values)	Moody Aa− rating (441 values)
Retail (400 values)	Moody A+ rating (653 values)
Software (300 values)	Moody A rating (654 values)
Technology Equipment (550 values)	Moody A− rating (653 values)
Telecommunications (350 values)	Moody Baa+ rating (539 values)
Transportation (200 values)	Moody Baa rating (539 values)
	Moody Baa− rating (539 values)
S&P AAA rating (50 values)	

Figure 4.16: Attributes of the commercial-paper market.

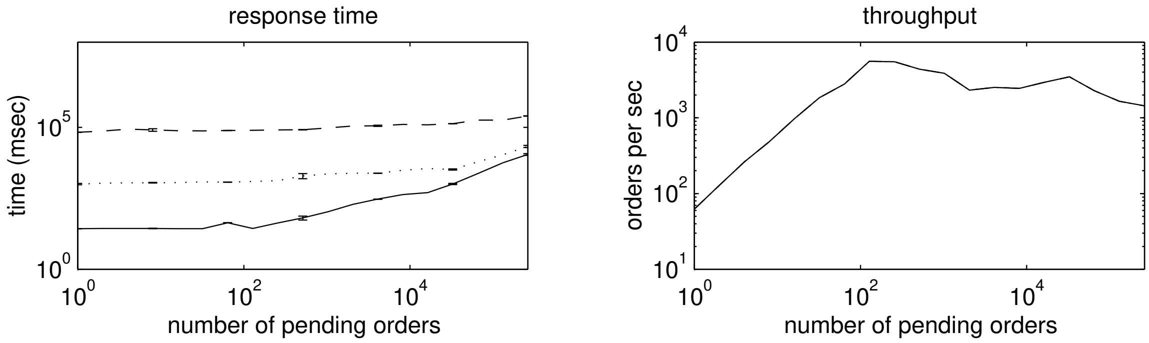
has changed, the sale price may be different. If Katie is lucky, she may get more than one-month interest, but in an unfavorable case she may get smaller interest or even lose part of her investment.

The description of commercial paper includes two attributes: the issuing company and maturity date. Furthermore, investors usually group companies by their credit rating and industry group. In Figure 4.16, we show the attributes, values, and standard sets of the commercial-paper market.

We have run experiments with up to 260,000 pending orders, using the same control variables and their settings as in the car-market tests, with the addition of a third setting for the number of new orders. We show the results in Figures 4.17–4.21, and plot the dependency of the system's performance on the control variables in Figures 4.22–4.24. The experiments have confirmed that the system scales to large markets, and that its performance in real-life markets is close to the results of the artificial tests; it parses 1,500 to 4,500 orders per second, and generates 700 to 2,000 trades per second.

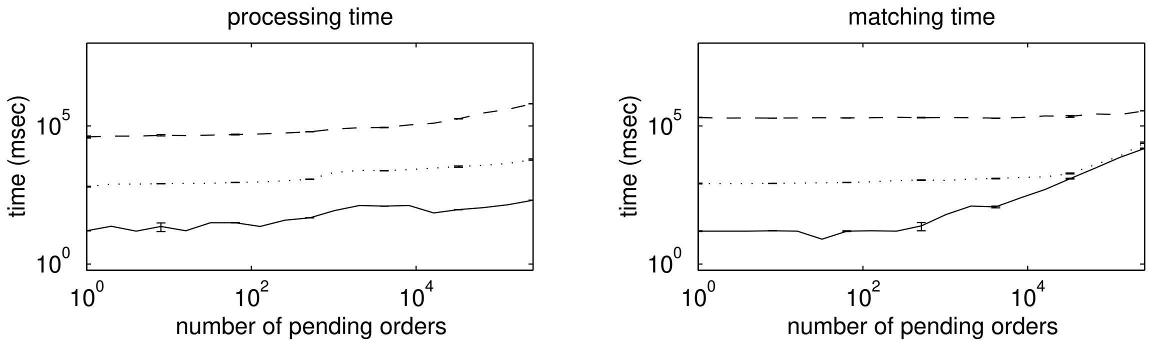


(a) Time of processing new orders (left) and matching old orders (right).

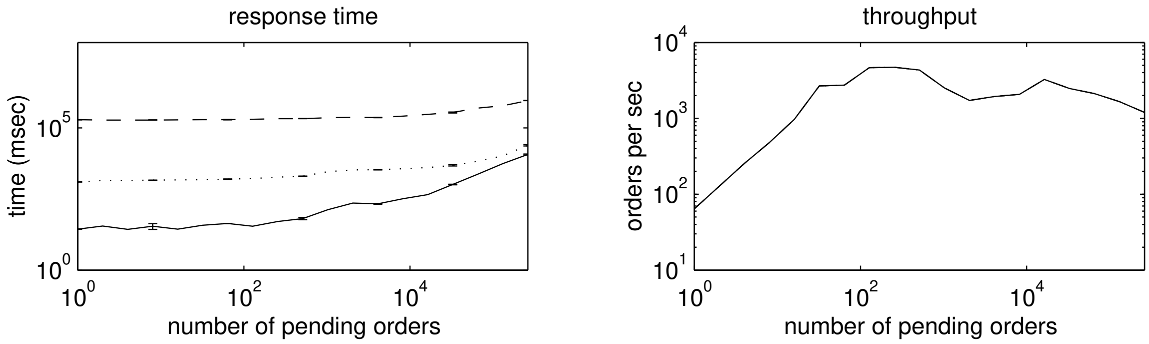


(b) Response time (left) and throughput (right).

Figure 4.17: Performance in the commercial-paper market, for *matching density of 0.0001.* We show the dependency of the system's performance on the number of pending orders, for 256 new orders (solid lines), 8,192 orders (dotted lines), and 262,144 orders (dashed lines). Both horizontal and vertical scales are logarithmic, and the vertical bars mark the minimal and maximal values of the time measurements.

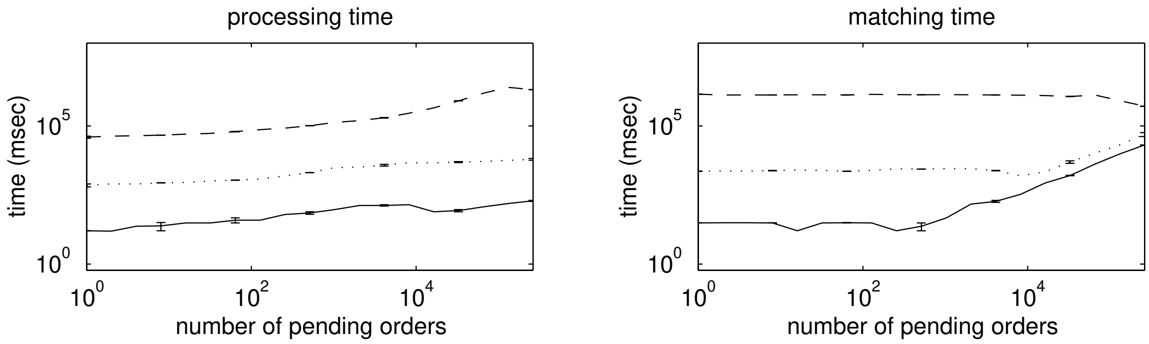


(a) Time of processing new orders (left) and matching old orders (right).

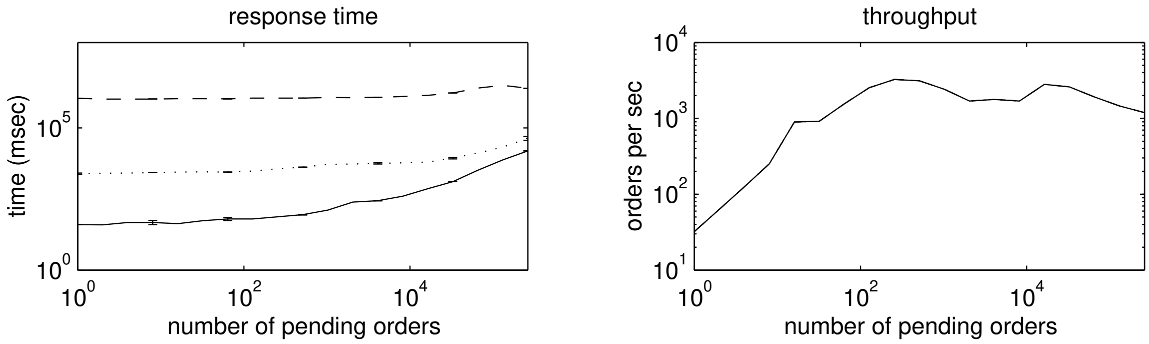


(b) Response time (left) and throughput (right).

Figure 4.18: Performance in the commercial-paper market, for *matching density of 0.001.* We show results for 256 new orders (solid lines), 8,192 orders (dotted lines), and 262,144 orders (dashed lines), with the minimal and maximal values (vertical bars).

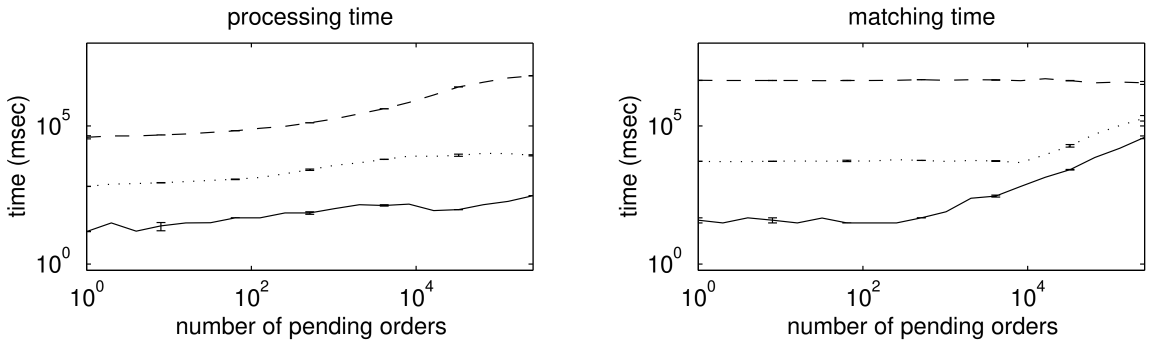


(a) Time of processing new orders (left) and matching old orders (right).

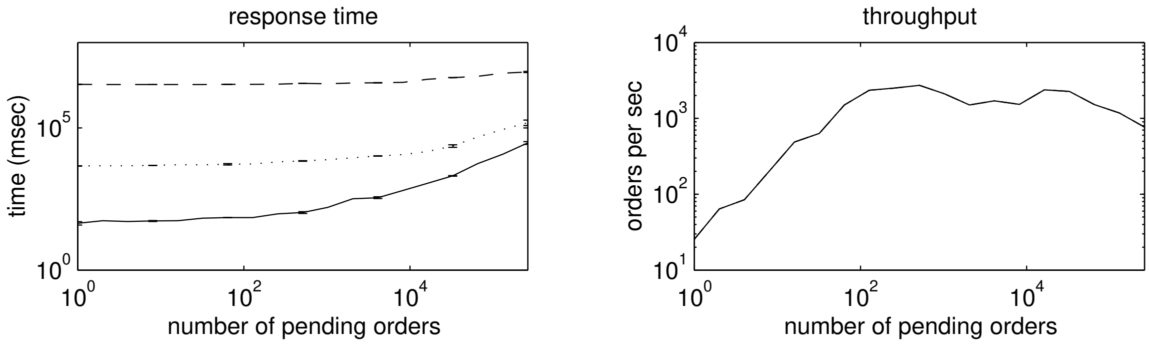


(b) Response time (left) and throughput (right).

Figure 4.19: Performance in the commercial-paper market, for *matching density of 0.01.* We show results for 256 new orders (solid lines), 8,192 orders (dotted lines), and 262,144 orders (dashed lines), with the minimal and maximal values (vertical bars).

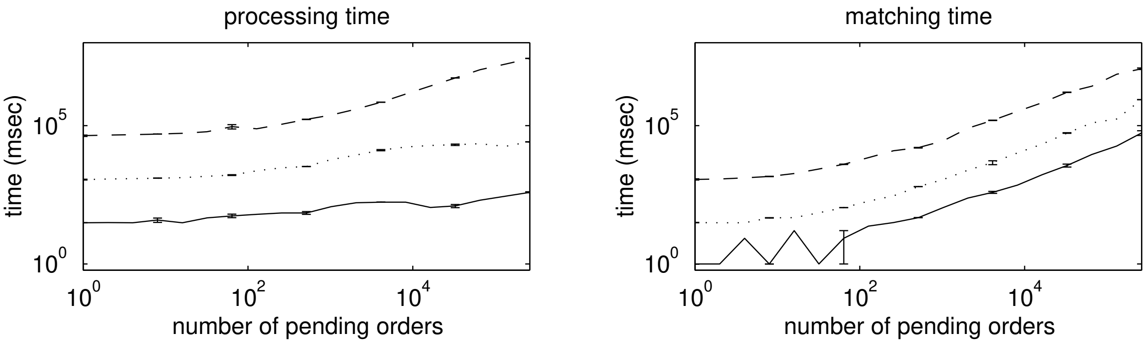


(a) Time of processing new orders (left) and matching old orders (right).

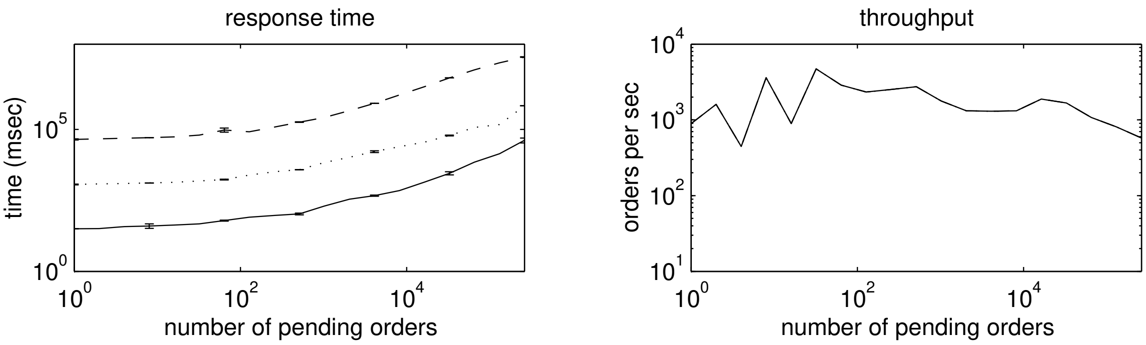


(b) Response time (left) and throughput (right).

Figure 4.20: Performance in the commercial-paper market, for *matching density of 0.1.* We show results for 256 new orders (solid lines), 8,192 orders (dotted lines), and 262,144 orders (dashed lines), with the minimal and maximal values (vertical bars).

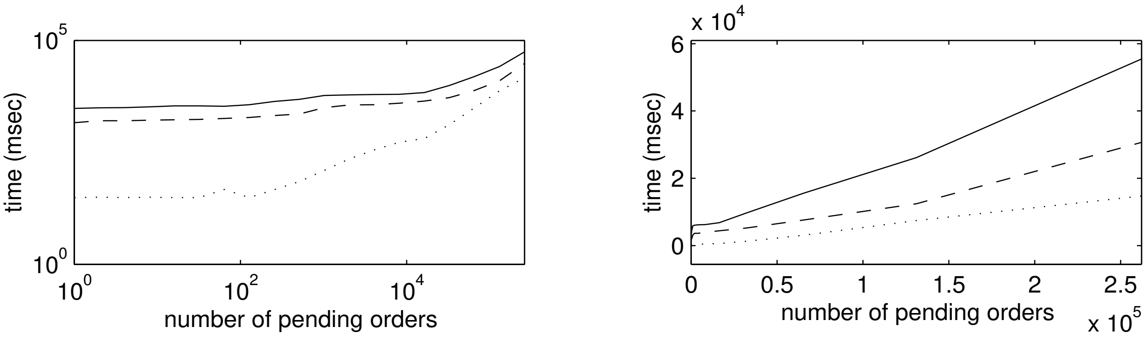


(a) Time of processing new orders (left) and matching old orders (right).

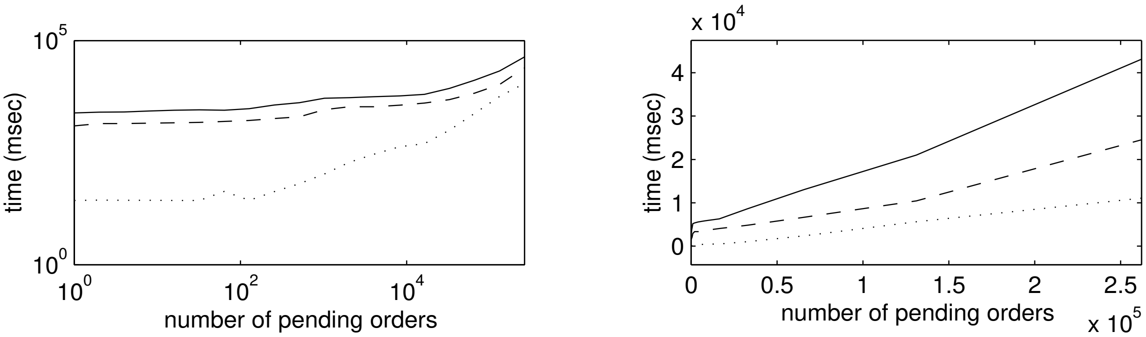


(b) Response time (left) and throughput (right).

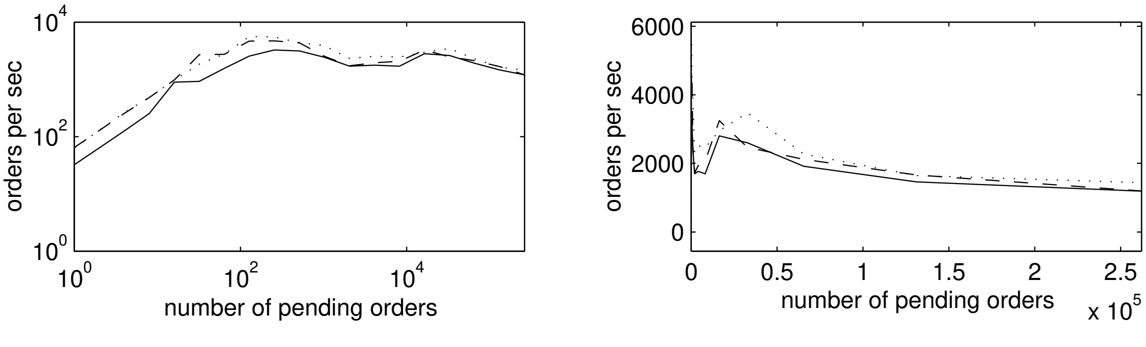
Figure 4.21: Performance in the commercial-paper market, for *matching density of 1.* We show results for 256 new orders (solid lines), 8,192 orders (dotted lines), and 262,144 orders (dashed lines), with the minimal and maximal values (vertical bars).



(a) Time of processing new orders and matching old orders.

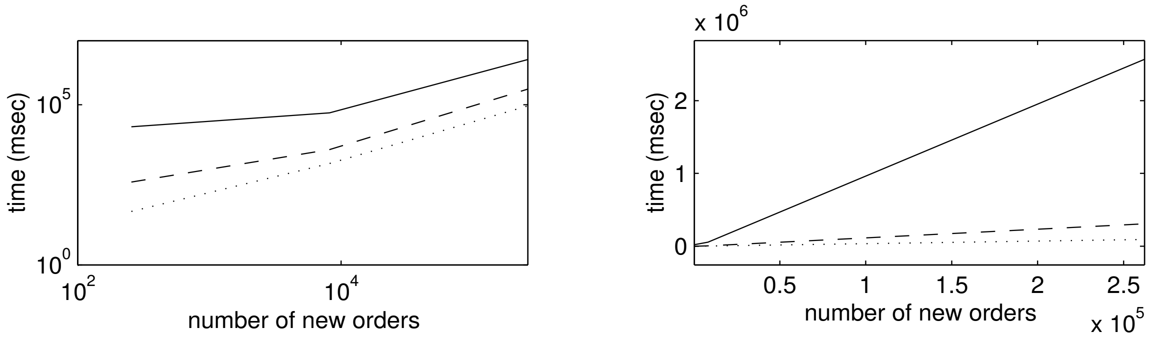


(b) Average time between order placement and response.

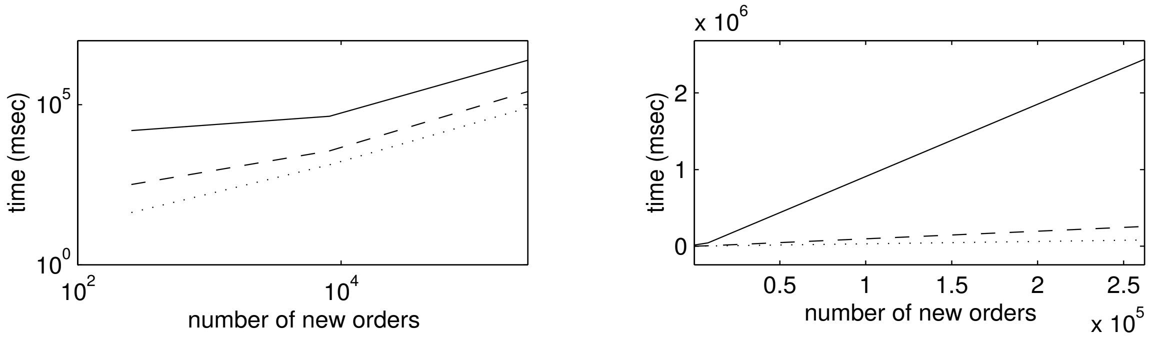


(c) Maximal number of new orders per second.

Figure 4.22: Dependency of the performance on the *number of pending orders,* for the commercial-paper market. The dotted lines show experiments with 256 new orders and matching density of 0.0001. The dashed lines are for 8,192 new orders and matching density of 0.001. The solid lines are for 8,192 new orders and matching density of 0.01. The graphs on the left are in logarithmic scale, whereas the graphs on the right are in linear scale.

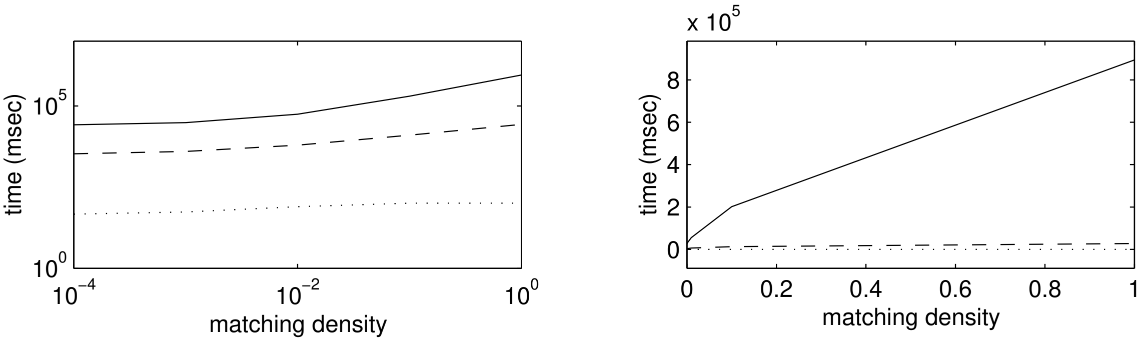


(a) Time of processing new orders and matching old orders.

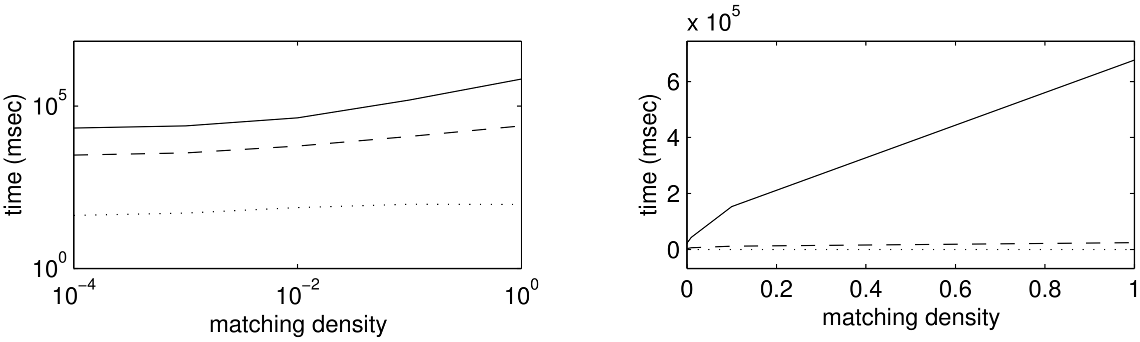


(b) Average time between order placement and response.

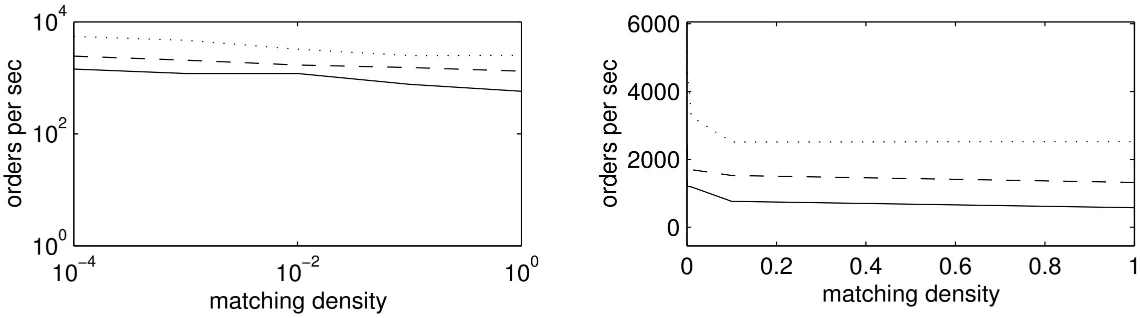
Figure 4.23: Dependency of the performance on the *number of new orders,* for the commercial-paper market. The dotted lines show experiments with 256 pending orders and matching density of 0.0001. The dashed lines are for 8,192 pending orders and matching density of 0.001. The solid lines are for 262,144 pending orders and matching density of 0.01. The graphs on the left are in logarithmic scale, whereas the graphs on the right are in linear scale. We do not plot this dependency for the throughput, because the rate of incoming orders directly depends on the throughput, and thus we cannot view the number of new orders as an independent control variable.



(a) Time of processing new orders and matching old orders.



(b) Average time between order placement and response.



(c) Maximal number of new orders per second.

Figure 4.24: Dependency of the performance on the *matching density,* for the commercial-paper market. The dotted lines show experiments with 256 pending orders and 256 new orders. The dashed lines are for 8,192 pending orders and 8,192 new orders. The solid lines are for 262,144 pending orders and 8,192 new orders. The graphs on the left are in logarithmic scale, whereas the graphs on the right are in linear scale.

Chapter 5

Concluding Remarks

The modern economy includes a variety of marketplaces with millions of participants, and the Internet has led to the development of new efficient markets. Computer scientists have studied algorithms for various auctions and standardized exchanges, but they have done little work on exchange markets for complex nonstandard commodities.

We have proposed a formal model for trading complex securities, and built an exchange that allows complex constraints in descriptions of buy orders. It supports markets with up to 260,000 orders and parses hundreds of new orders per second. We have included several heuristics for maximizing traders' satisfaction, and allowed traders to participate in choosing matches for their orders.

On the negative side, the developed system does not allow complex constraints in sell orders, does not guarantee optimal trades, and does not support soft constraints or preference functions. We plan to address these problems as part of the future research. We are also working on a distributed version of the exchange, which will improve scalability; it will include a central matcher and multiple preprocessing modules, whose roles are similar to that of stock brokers on Wall Street.

References

[Andersson and Ygge, 1998] Arne Andersson and Fredrik Ygge. Managing large scale computational markets. In *Proceedings of the Thirty-First Hawaiian International Conference on System Sciences*, volume 7, 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.

[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.

[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):315–330, 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, 2000] Martin Bichler. An experimental analysis of multi-attribute auctions. *Decision Support Systems*, 29:249–268, 2000.

[Cason and Friedman, 1999] Timothy Cason and Dan 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, Cheltenham, United Kingdom, 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.

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

[Fujishima *et al.*, 1999a] 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*, pages 548–553, 1999.

[Fujishima *et al.*, 1999b] Yuzo Fujishima, David McAdams, and Yoav Shoham.

Speeding up ascending-bid auctions. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 554–563, 1999.

[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.

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

[Jones and Koehler, 2000] Joni L. Jones and Gary J. Koehler. Incompletely specified combinatorial auctions. Unpublished manuscript, 2000.

[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.

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

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

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

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

[Parkes and Ungar, 2000] 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, 1999] David C. Parkes. *i*Bundle: An efficient ascending price bundle auction. In *Proceedings of the ACM Conference on Electronic Commerce*, pages 148–157, 1999.

[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*, 56:400–425, 1992.

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

[Sandholm and Suri, 2000] 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.

[Sandholm *et al.*, 2001] 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*, 2001. To appear.

[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. Approaches 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 341–348, 2000.

[Trenton, 1964] Rudolf W. Trenton. *Basic Economics.* Meredith Publishing Company, New York, NY, 1964.

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

Appendices

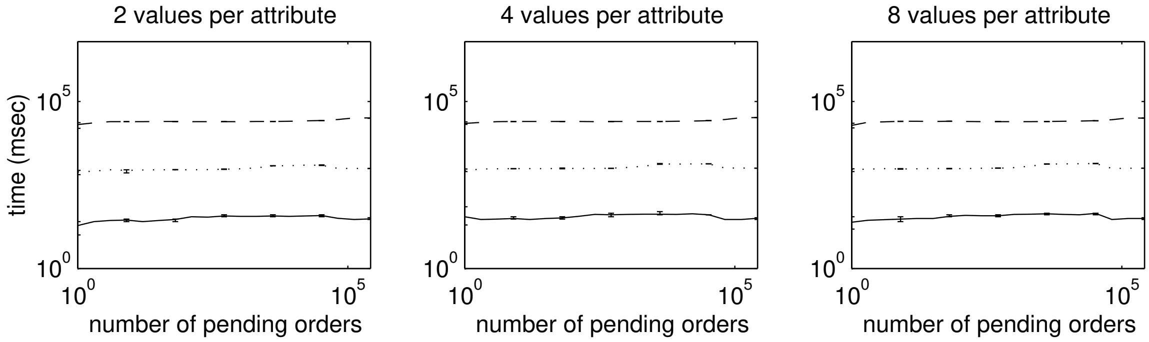
Appendix A

Experiments with a Small Number of Attributes

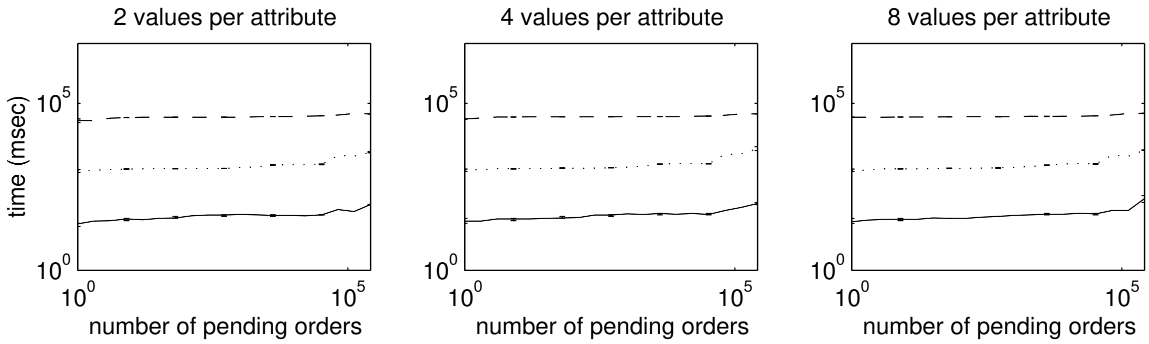
We give results for markets with a small number of attributes; specifically, we experiment with one, three, and ten attributes. We have explained the setup for these experiments in Section 4.1.

A.1 Processing Time

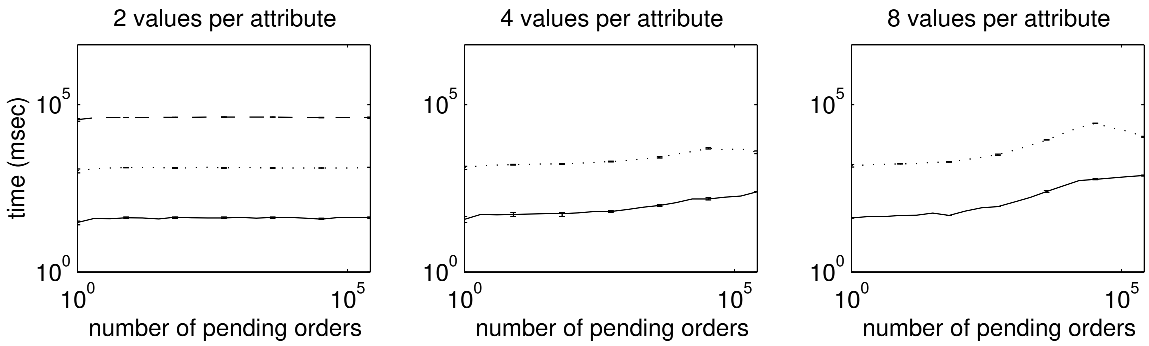
We show the mean time of processing new orders in the first half of the system's top-level loop (see Figure 3.2a), for various settings of control variables. We also mark the minimal and maximal values of the time measurements.



(a) Market with one attribute.

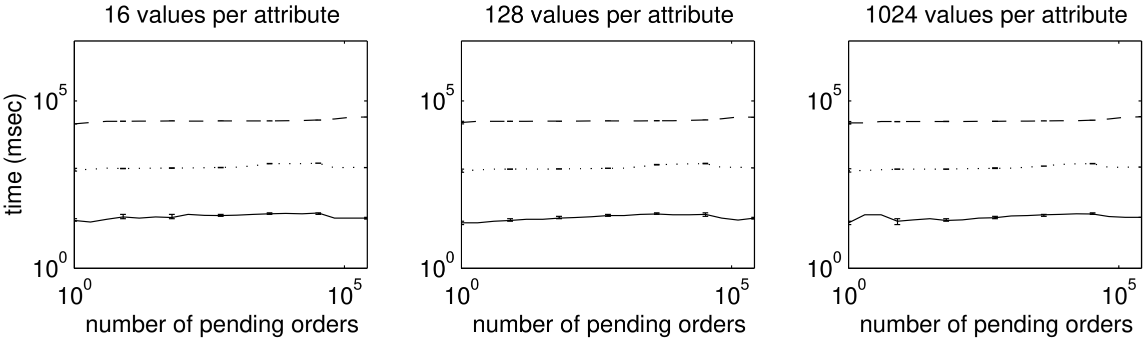


(b) Market with three attributes.

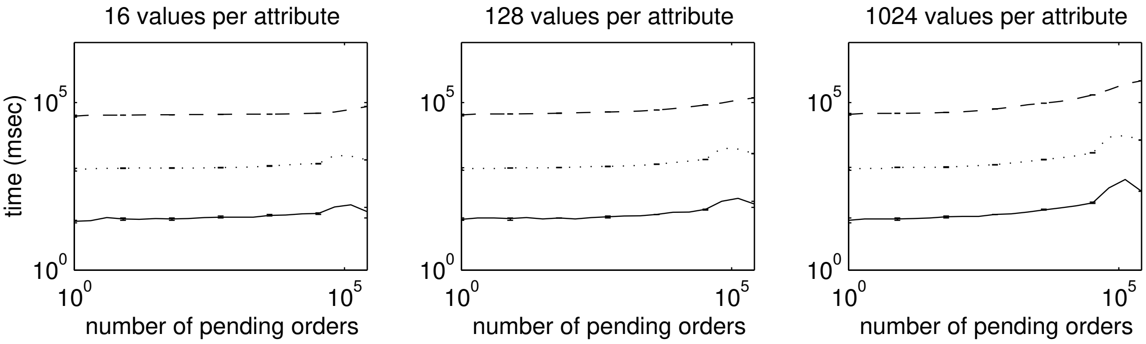


(c) Market with ten attributes.

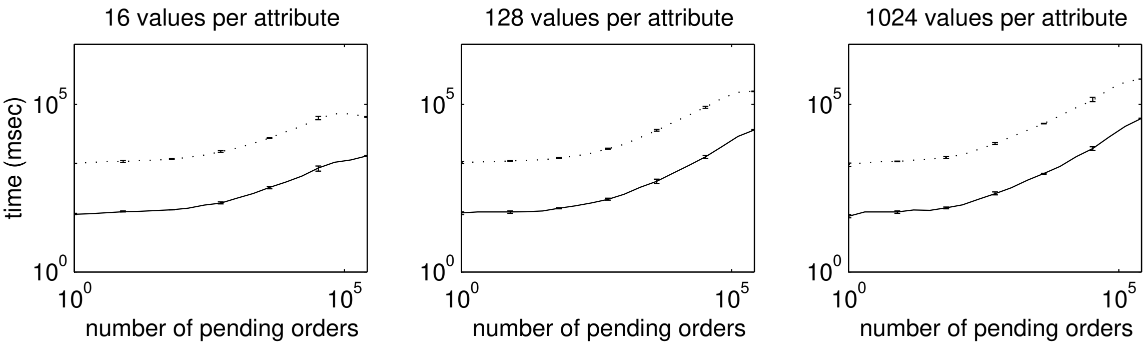
Figure A.1: Time of processing the new orders, for *matching density of 0.0001.* We consider markets with two values per attribute (left), four values per attribute (middle), and eight values per attribute (right). We show the dependency of the processing time on the number of pending orders, for 256 new orders (solid lines), 8,192 orders (dotted lines), and 262,144 orders (dashed lines). Both horizontal and vertical scales are logarithmic, and the vertical bars mark the minimal and maximal values of the time measurements.



(a) Market with one attribute.

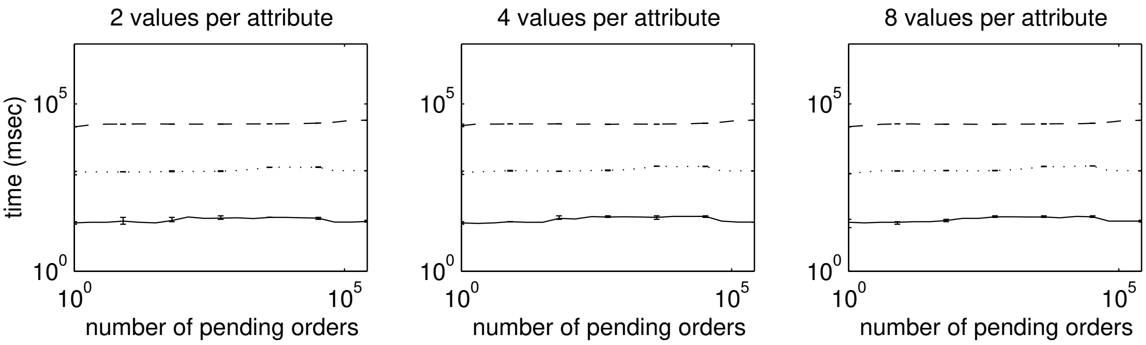


(b) Market with three attributes.

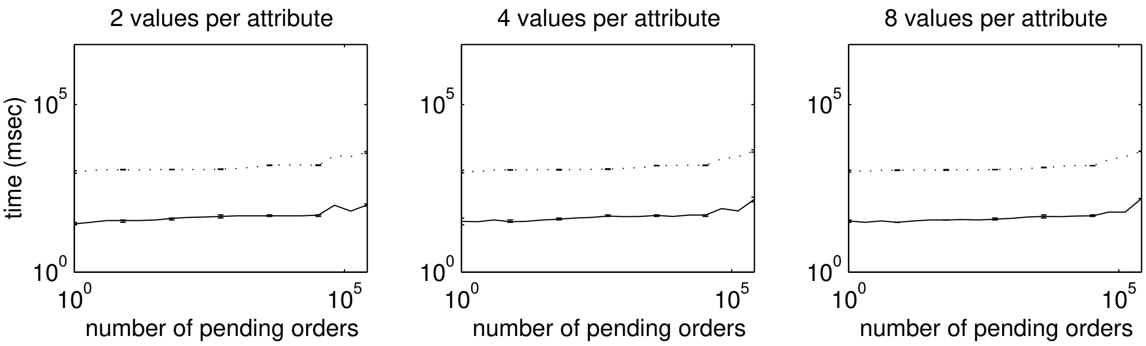


(c) Market with ten attributes.

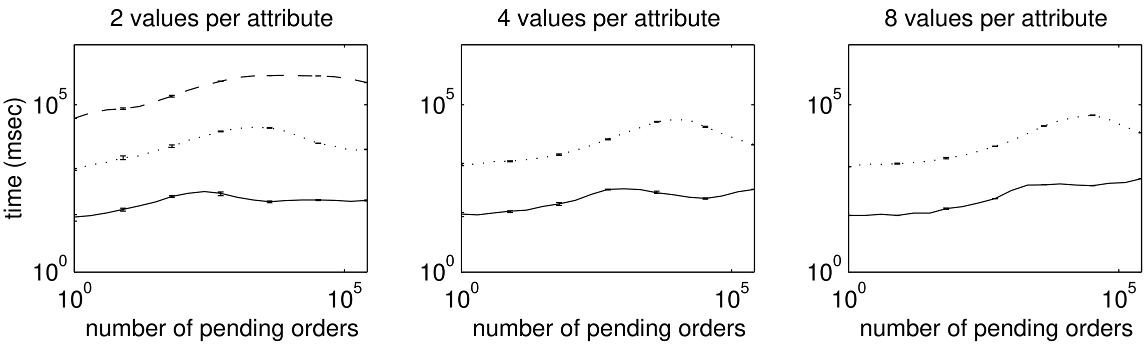
Figure A.2: Time of processing the new orders, for *matching density of 0.0001.* We consider markets with 16 values per attribute (left), 128 values per attribute (middle), and 1,024 values per attribute (right); the legend is the same as in Figure A.1.



(a) Market with one attribute.

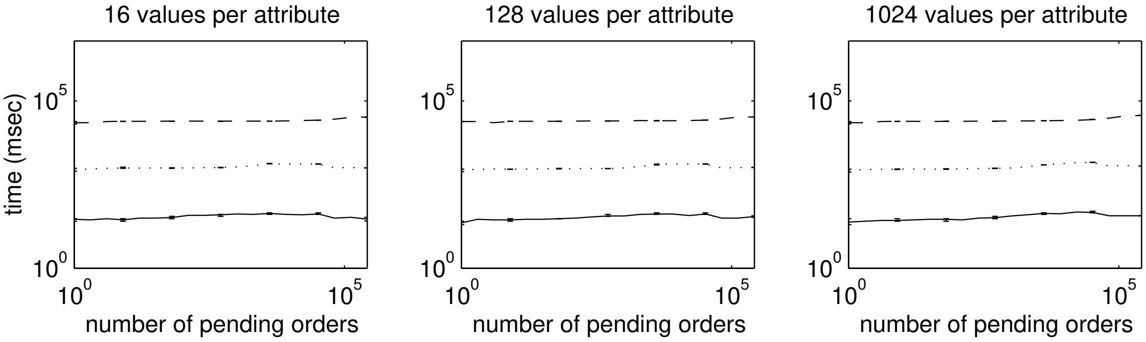


(b) Market with three attributes.

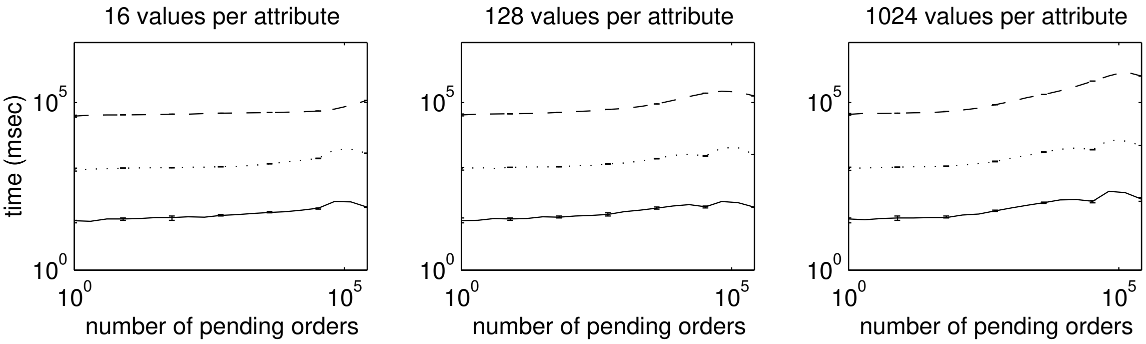


(c) Market with ten attributes.

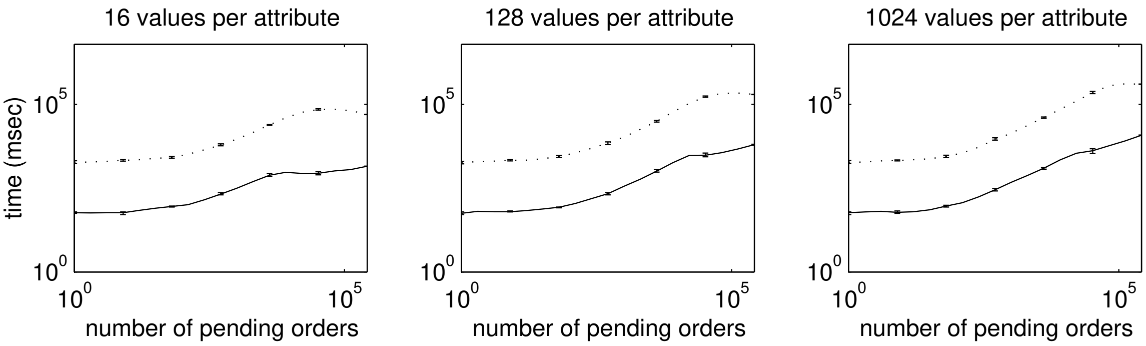
Figure A.3: Time of processing the new orders, for *matching density of 0.001.* We consider markets with two values per attribute (left), four values per attribute (middle), and eight values per attribute (right); the legend is the same as in Figure A.1.



(a) Market with one attribute.

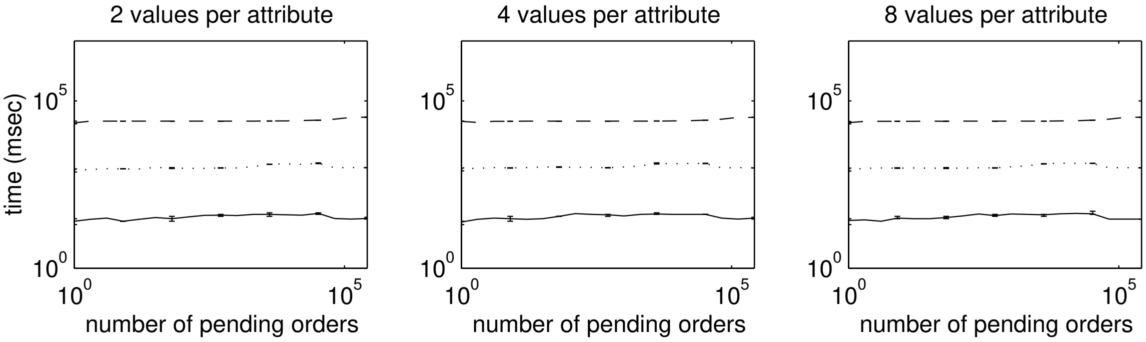


(b) Market with three attributes.

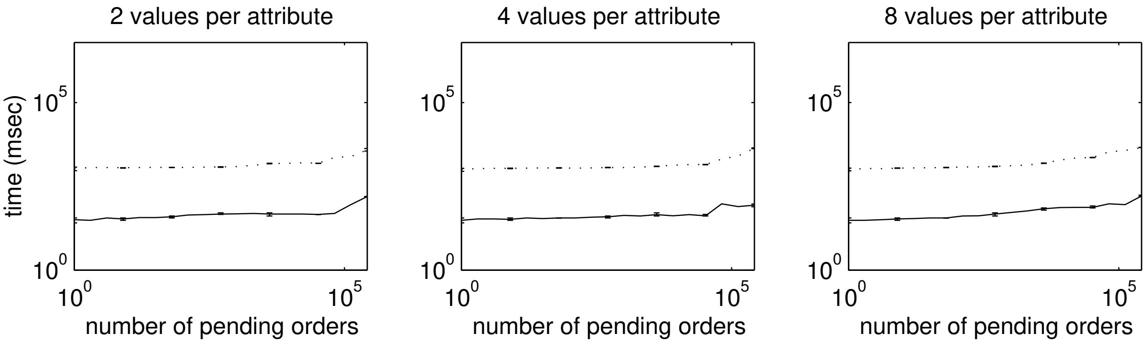


(c) Market with ten attributes.

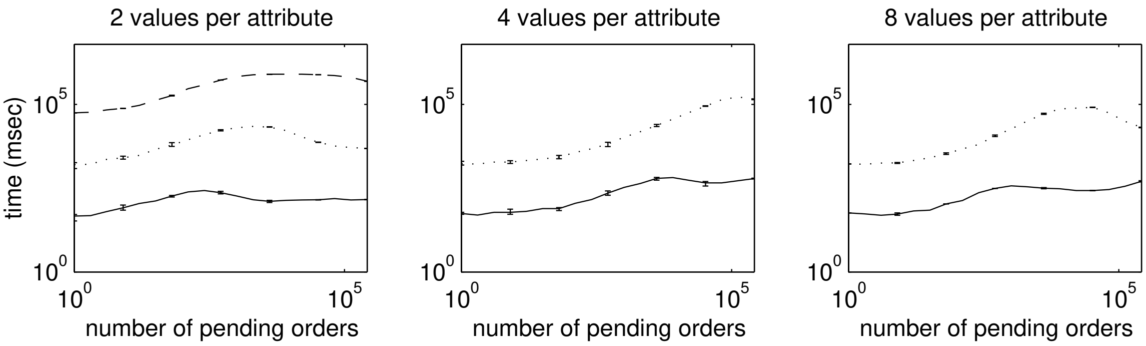
Figure A.4: Time of processing the new orders, for *matching density of 0.001.* We consider markets with 16 values per attribute (left), 128 values per attribute (middle), and 1,024 values per attribute (right); the legend is the same as in Figure A.1.



(a) Market with one attribute.

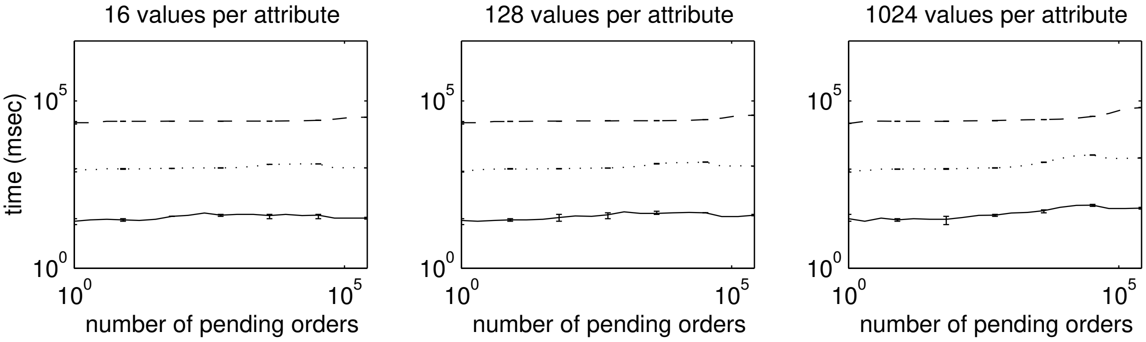


(b) Market with three attributes.

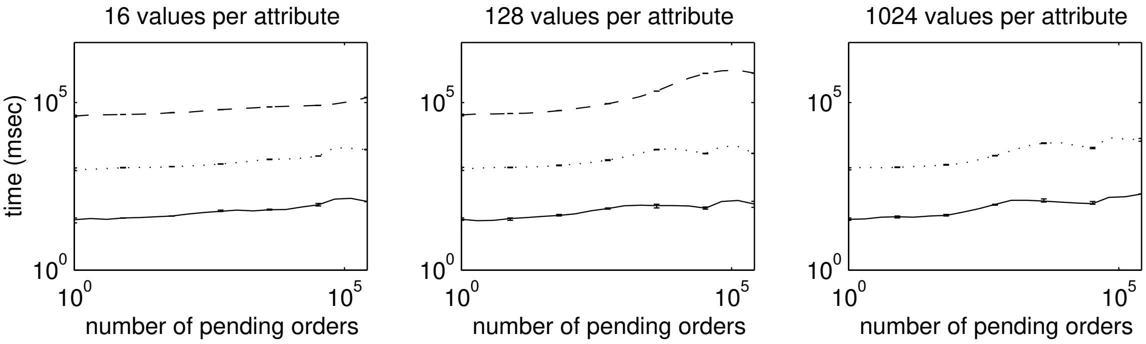


(c) Market with ten attributes.

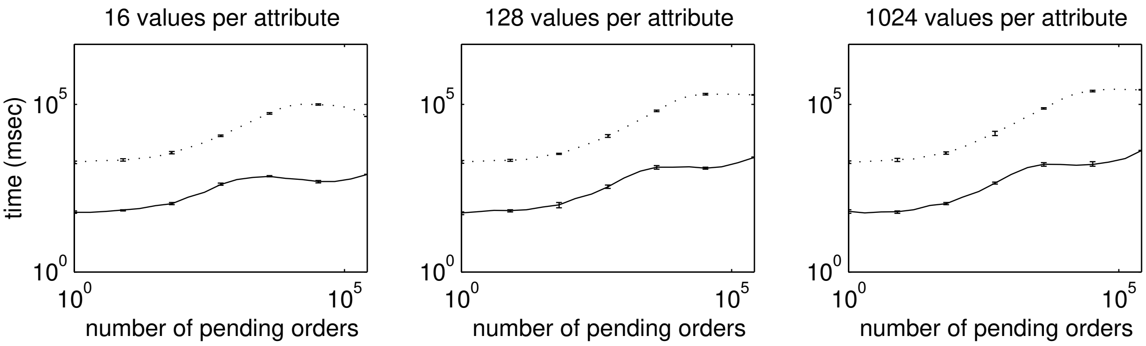
Figure A.5: Time of processing the new orders, for *matching density of 0.01.* We consider markets with two values per attribute (left), four values per attribute (middle), and eight values per attribute (right); the legend is the same as in Figure A.1.



(a) Market with one attribute.

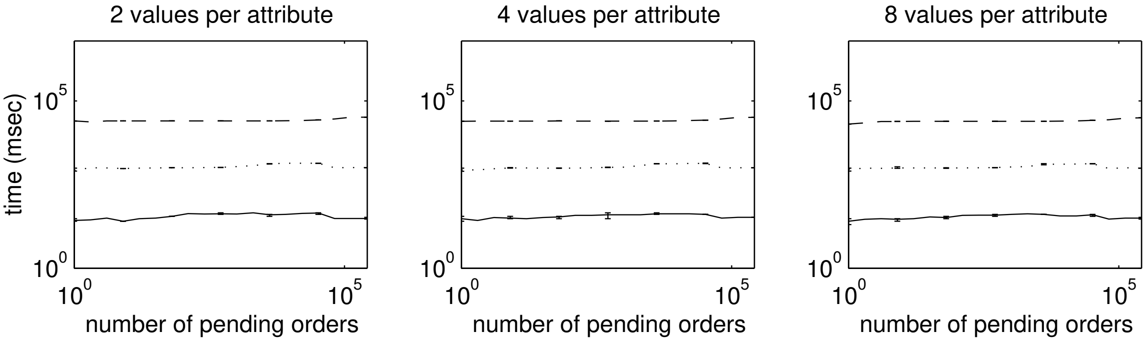


(b) Market with three attributes.

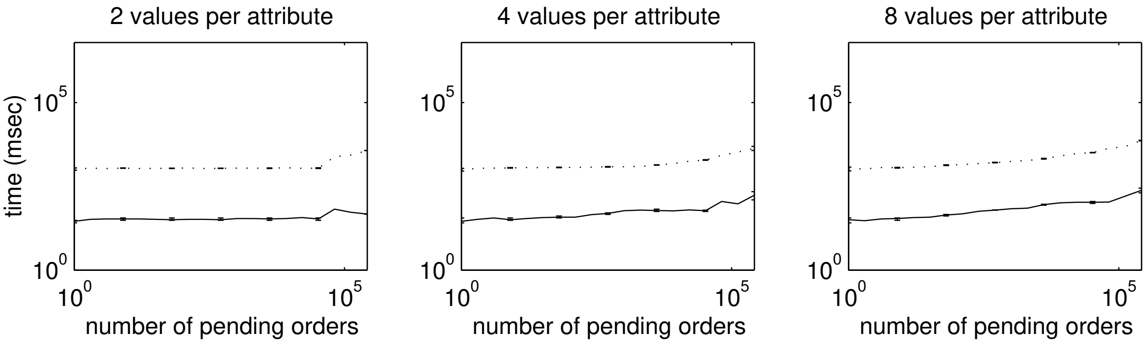


(c) Market with ten attributes.

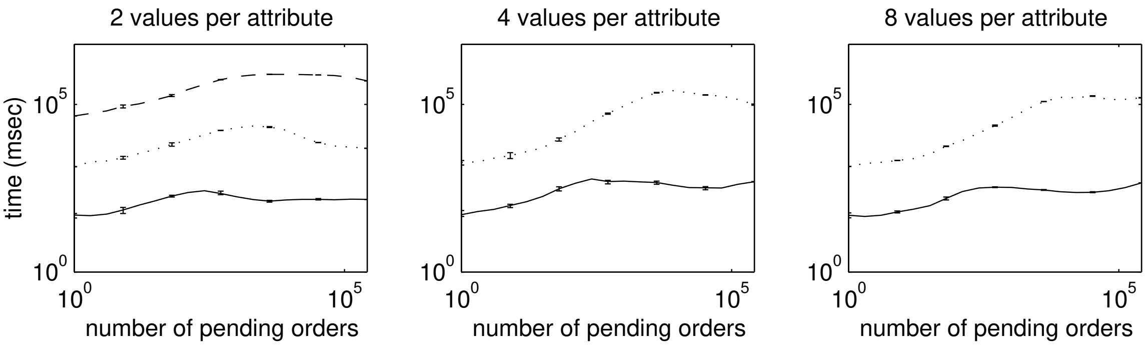
Figure A.6: Time of processing the new orders, for *matching density of 0.01.* We consider markets with 16 values per attribute (left), 128 values per attribute (middle), and 1,024 values per attribute (right); the legend is the same as in Figure A.1.



(a) Market with one attribute.

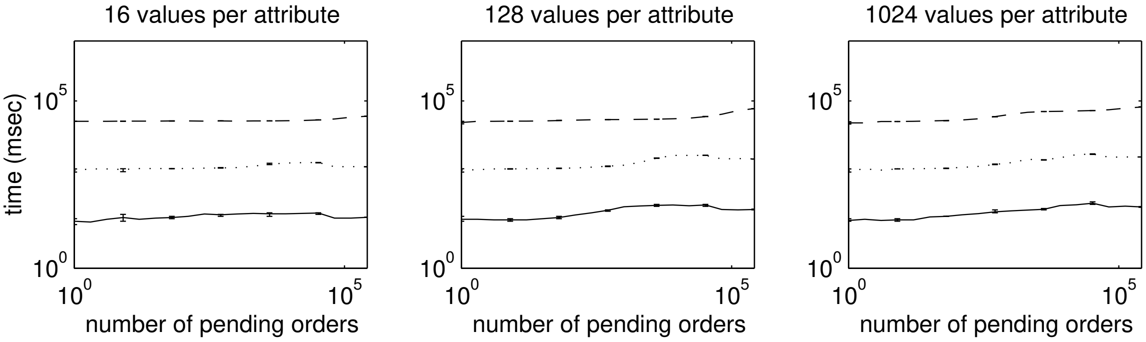


(b) Market with three attributes.

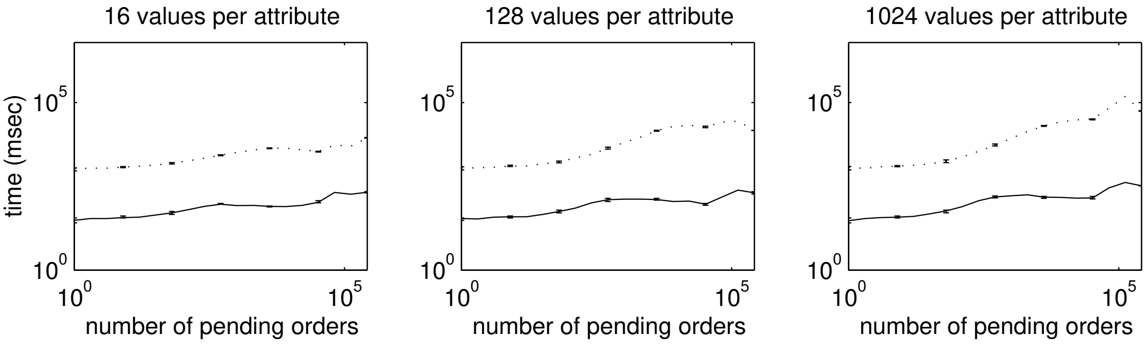


(c) Market with ten attributes.

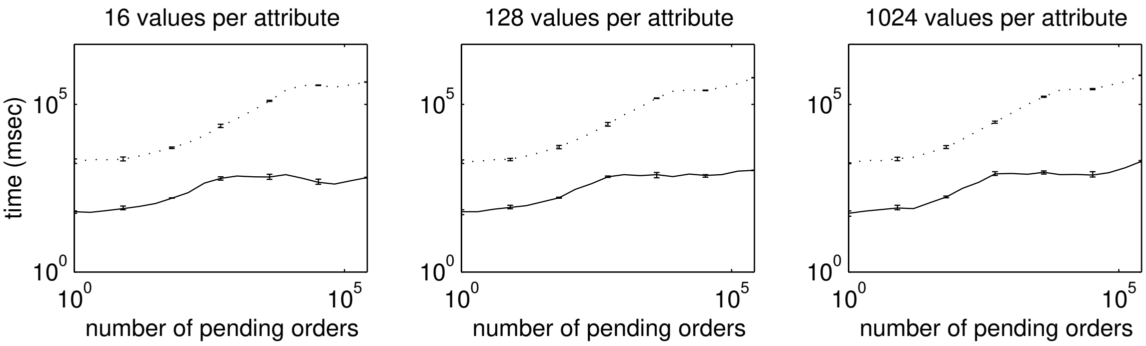
Figure A.7: Time of processing the new orders, for *matching density of 0.1.* We consider markets with two values per attribute (left), four values per attribute (middle), and eight values per attribute (right); the legend is the same as in Figure A.1.



(a) Market with one attribute.

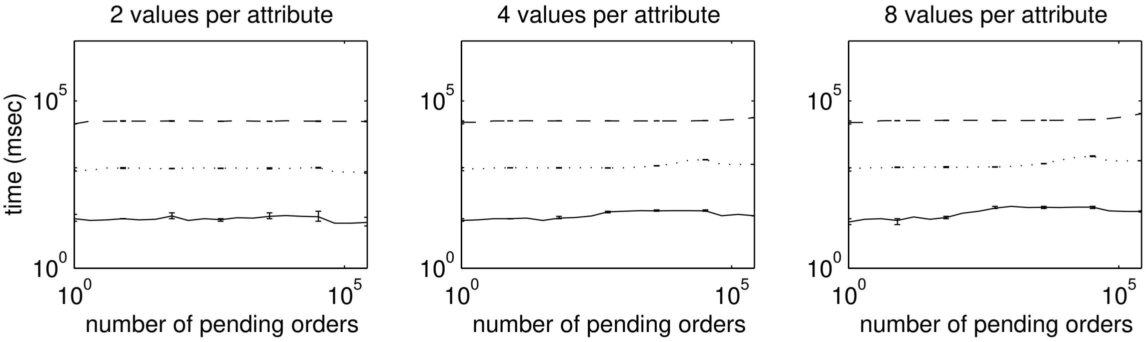


(b) Market with three attributes.

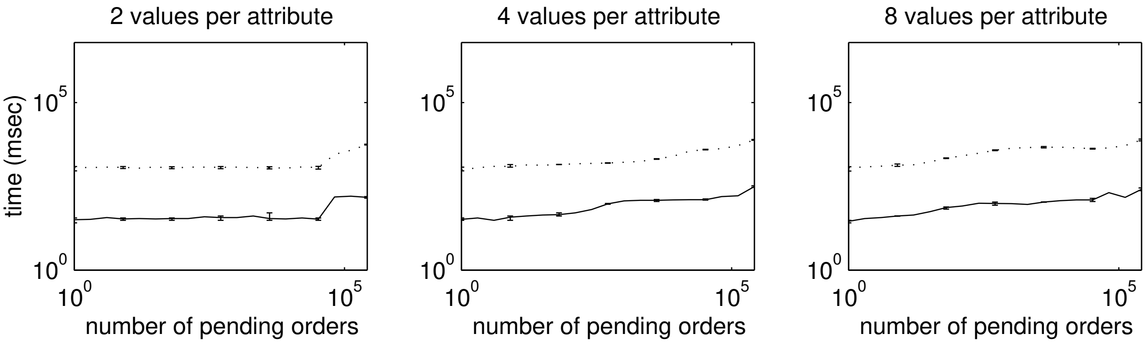


(c) Market with ten attributes.

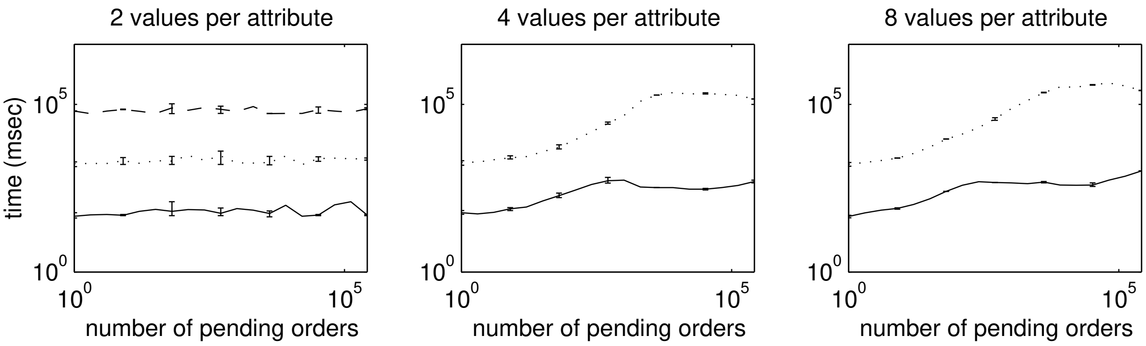
Figure A.8: Time of processing the new orders, for *matching density of 0.1.* We consider markets with 16 values per attribute (left), 128 values per attribute (middle), and 1,024 values per attribute (right); the legend is the same as in Figure A.1.



(a) Market with one attribute.

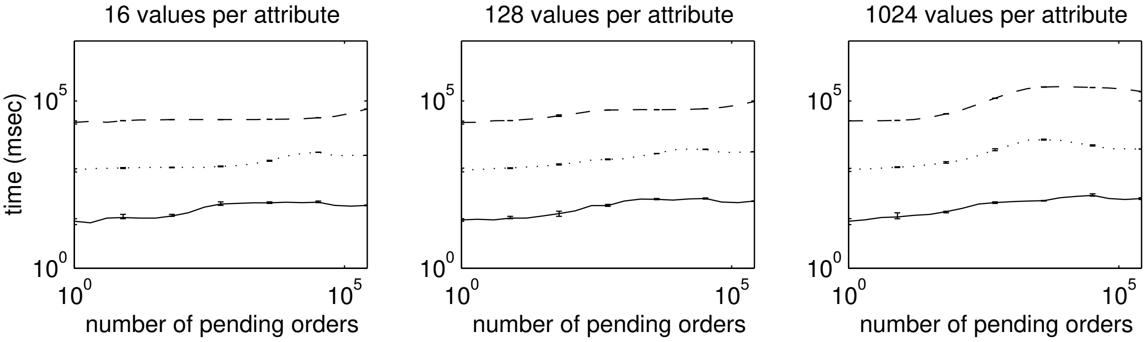


(b) Market with three attributes.

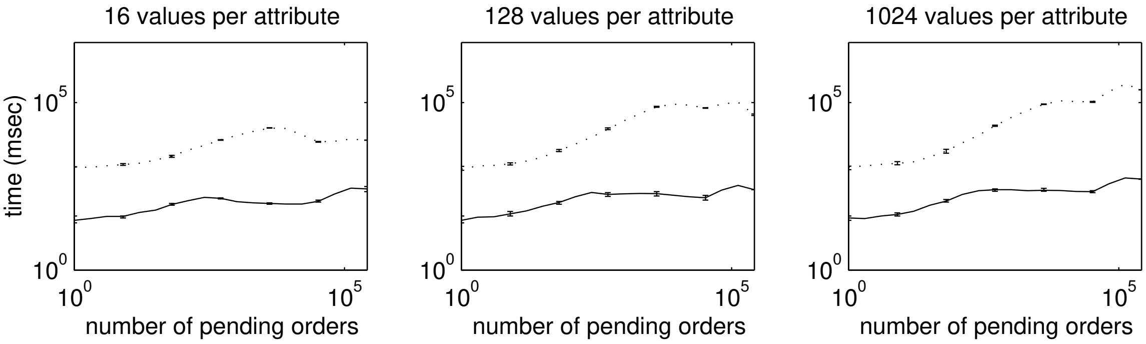


(c) Market with ten attributes.

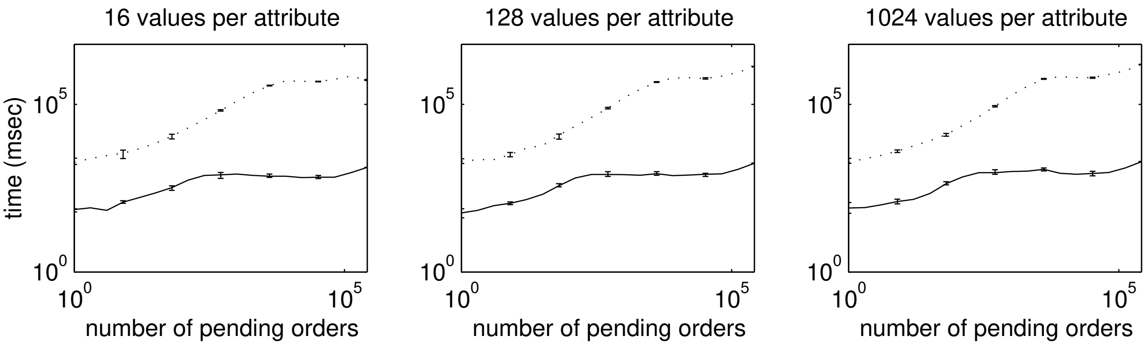
Figure A.9: Time of processing the new orders, for *matching density of 1.* We consider markets with two values per attribute (left), four values per attribute (middle), and eight values per attribute (right); the legend is the same as in Figure A.1.



(a) Market with one attribute.



(b) Market with three attributes.

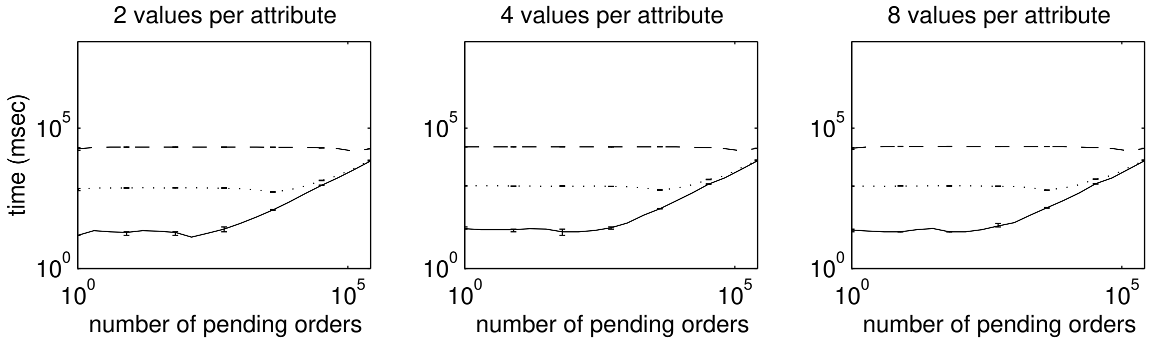


(c) Market with ten attributes.

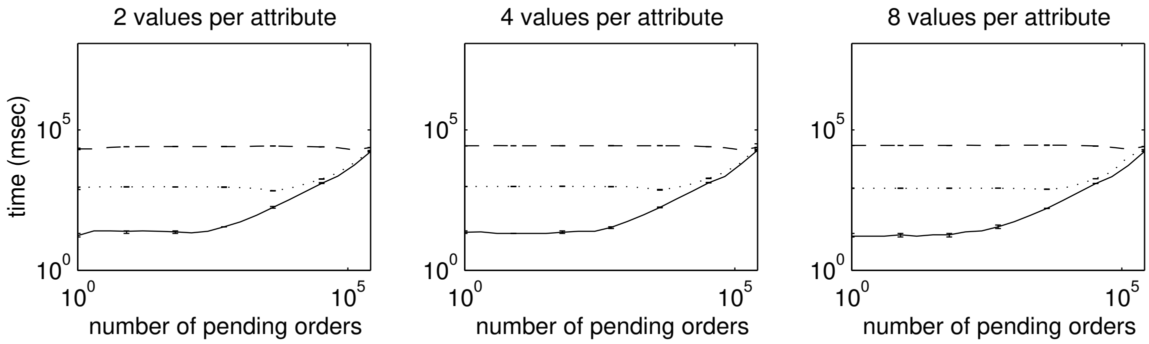
Figure A.10: Time of processing the new orders, for *matching density of 1.* We consider markets with 16 values per attribute (left), 128 values per attribute (middle), and 1,024 values per attribute (right); the legend is the same as in Figure A.1.

A.2 Matching Time

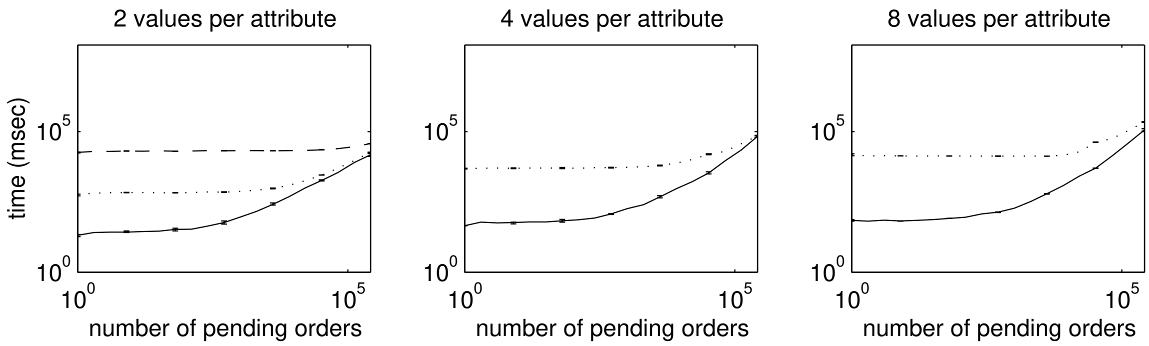
We give the mean time of matching all pending buy orders, in the second half of the system's top-level loop (see Figure 3.2a), along with the minimal and maximal values of the time measurements.



(a) Market with one attribute.

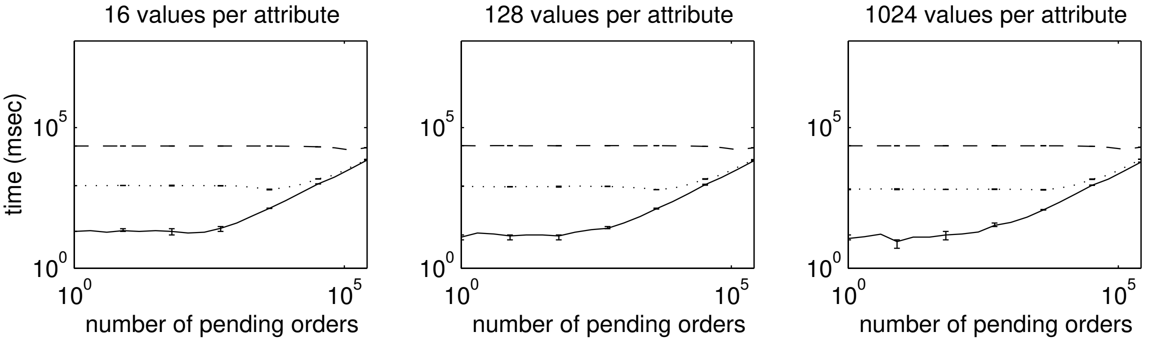


(b) Market with three attributes.

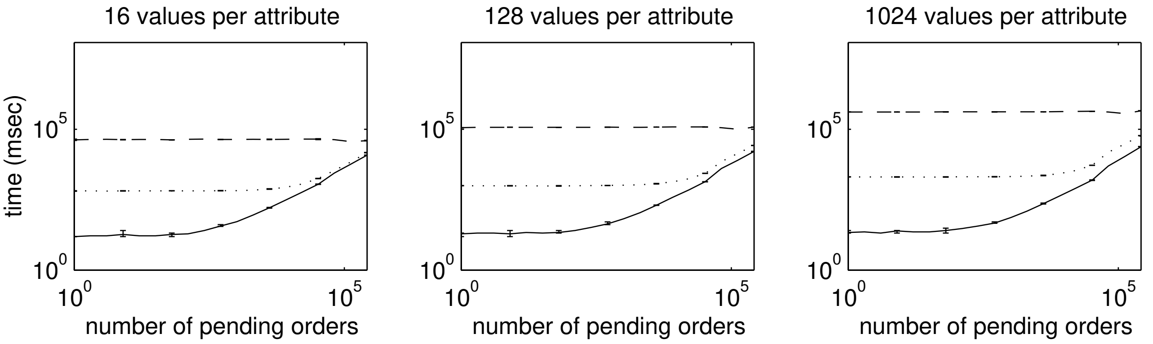


(c) Market with ten attributes.

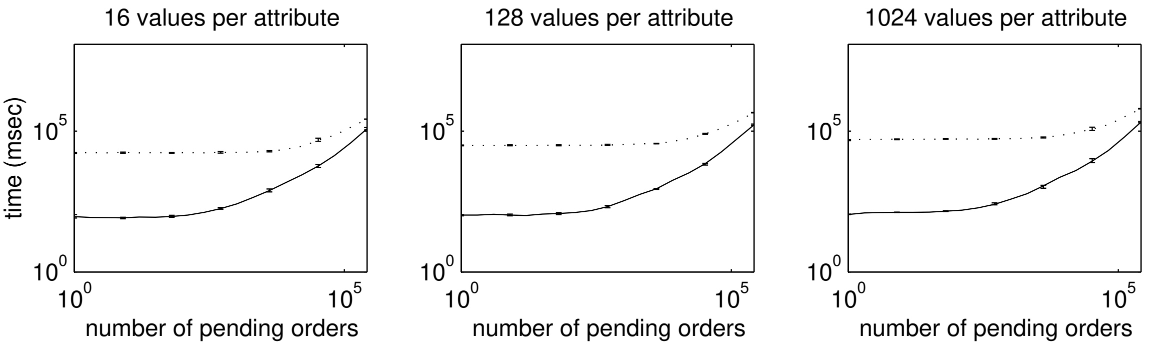
Figure A.11: Time of matching the pending orders, for *matching density of 0.0001.* We consider markets with two values per attribute (left), four values per attribute (middle), and eight values per attribute (right). We show the dependency of the matching time on the number of pending orders, for 256 new orders (solid lines), 8,192 orders (dotted lines), and 262,144 orders (dashed lines). Both horizontal and vertical scales are logarithmic, and the vertical bars mark the minimal and maximal values of the time measurements.



(a) Market with one attribute.

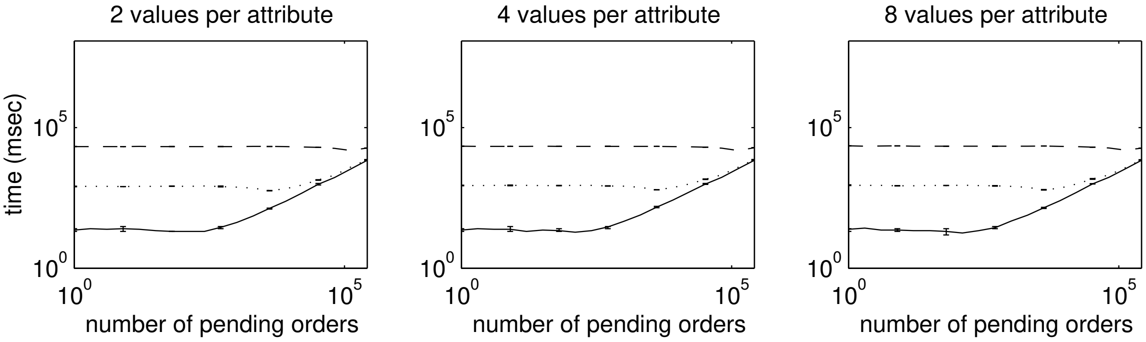


(b) Market with three attributes.

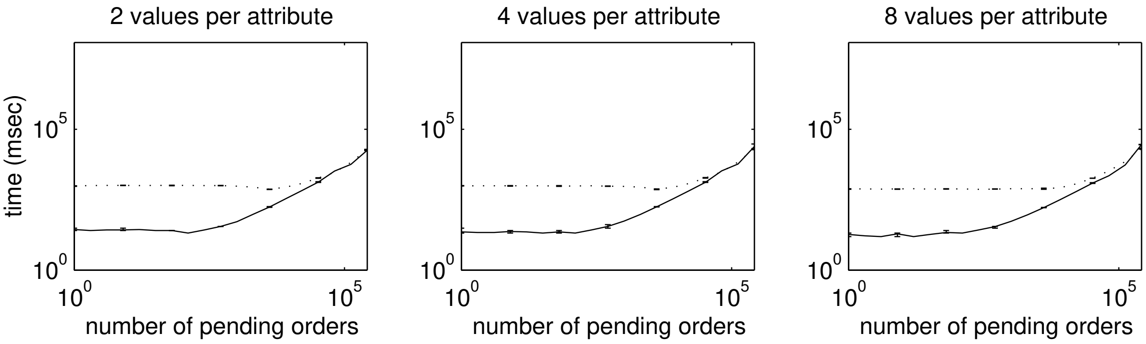


(c) Market with ten attributes.

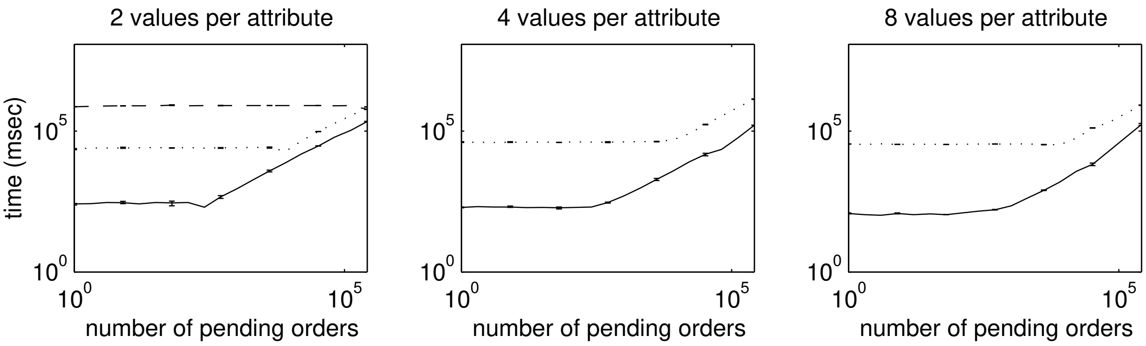
Figure A.12: Time of matching the pending orders, for *matching density of 0.0001.* We consider markets with 16 values per attribute (left), 128 values per attribute (middle), and 1,024 values per attribute (right); the legend is the same as in Figure A.11.



(a) Market with one attribute.

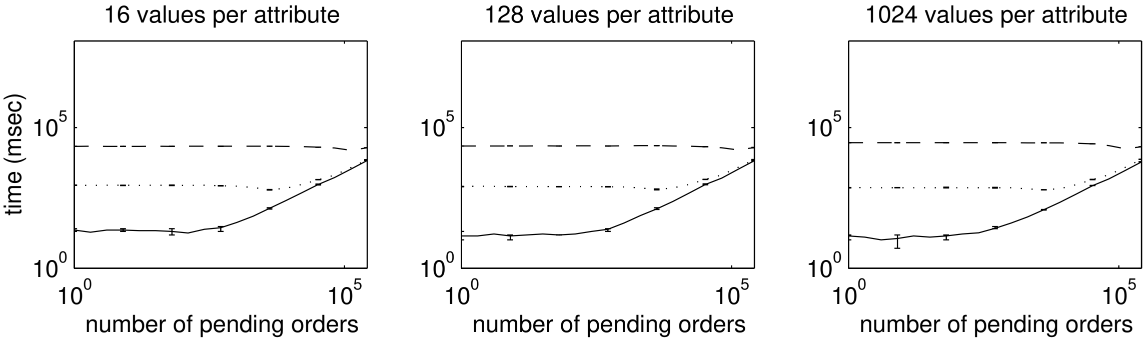


(b) Market with three attributes.

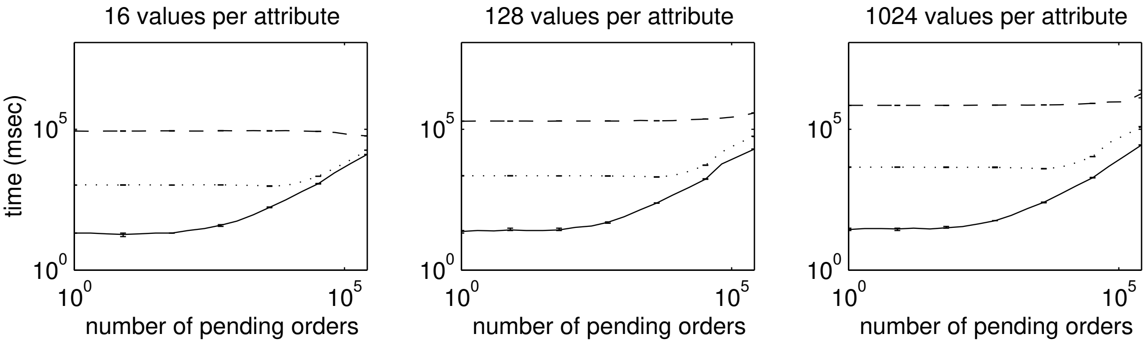


(c) Market with ten attributes.

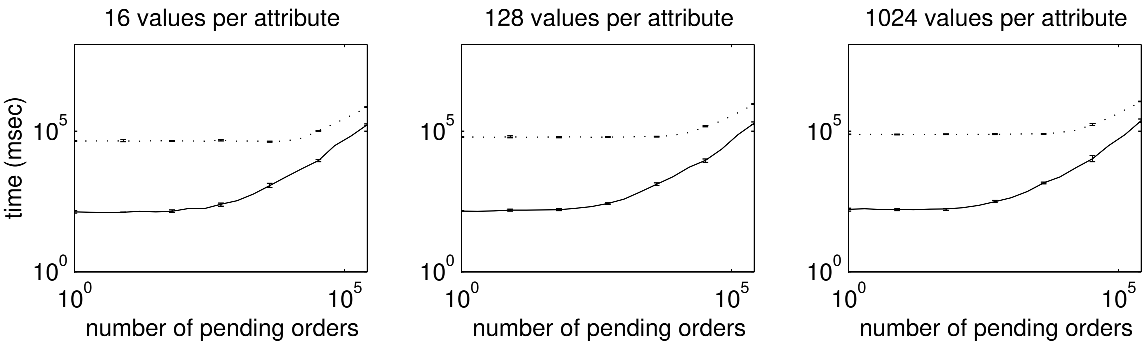
Figure A.13: Time of matching the pending orders, for *matching density of 0.001.* We consider markets with two values per attribute (left), four values per attribute (middle), and eight values per attribute (right); the legend is the same as in Figure A.11.



(a) Market with one attribute.

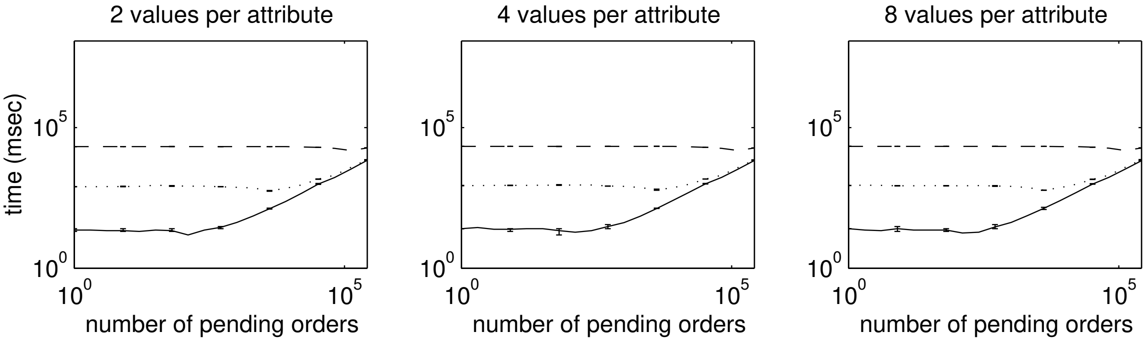


(b) Market with three attributes.

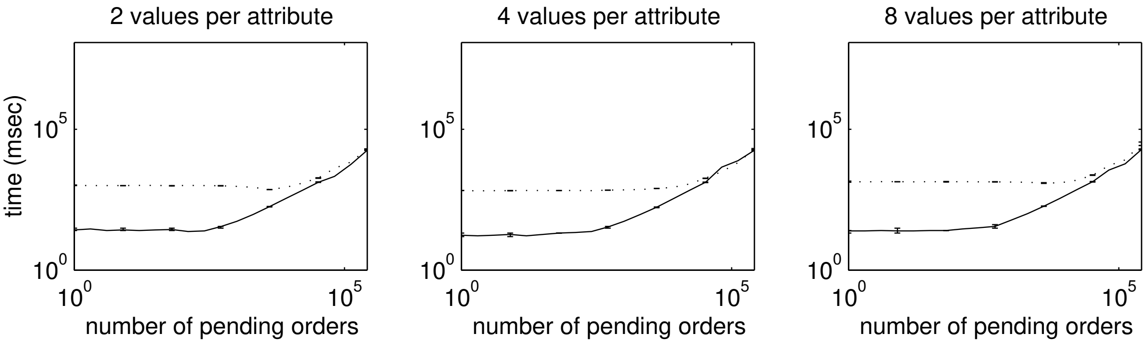


(c) Market with ten attributes.

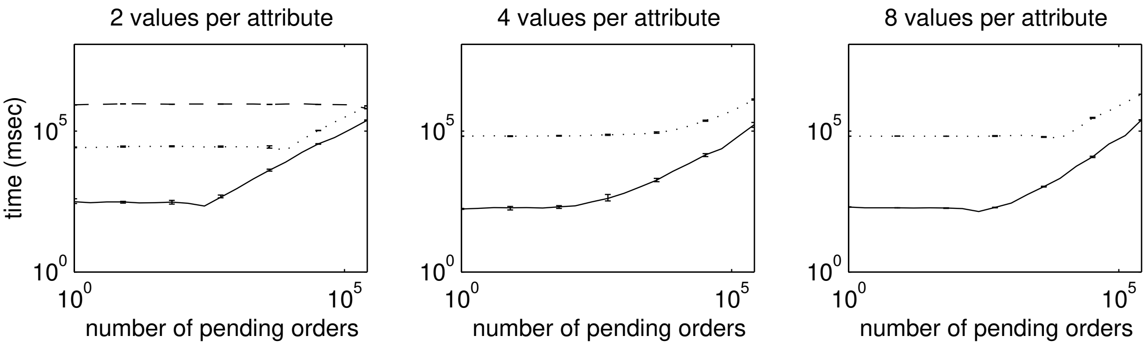
Figure A.14: Time of matching the pending orders, for *matching density of 0.001.* We consider markets with 16 values per attribute (left), 128 values per attribute (middle), and 1,024 values per attribute (right); the legend is the same as in Figure A.11.



(a) Market with one attribute.

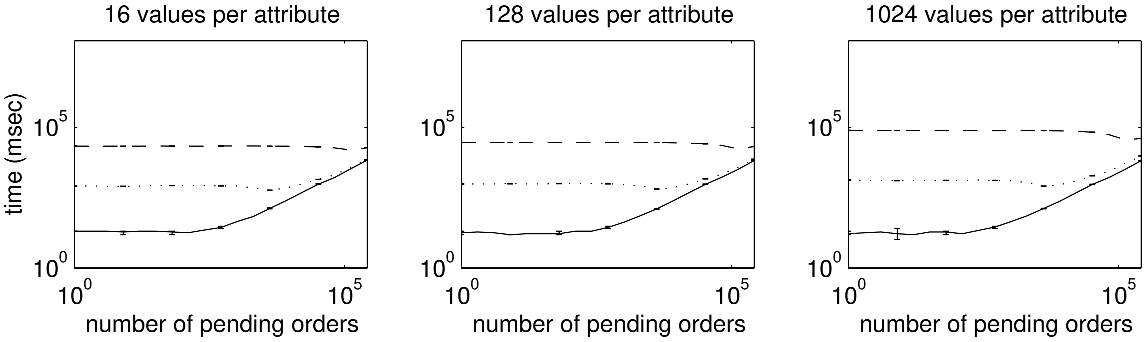


(b) Market with three attributes.

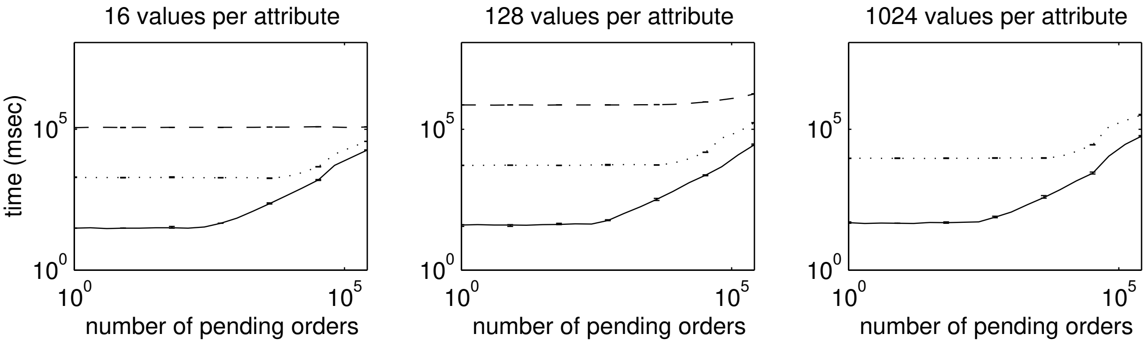


(c) Market with ten attributes.

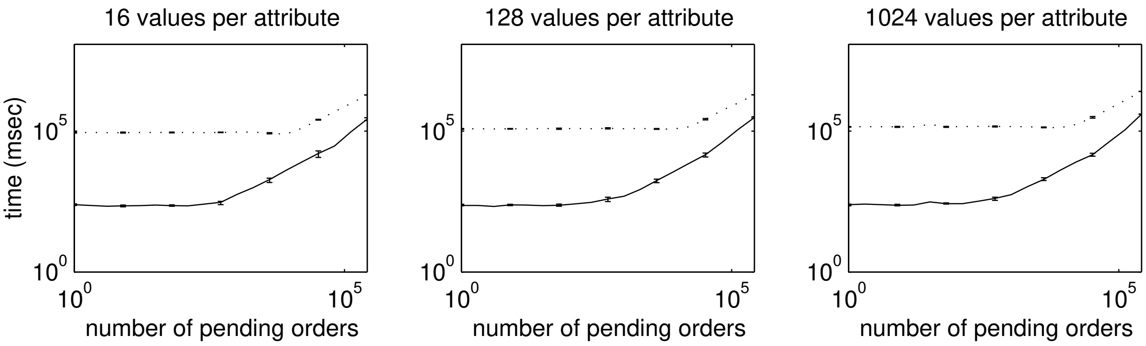
Figure A.15: Time of matching the pending orders, for *matching density of 0.01.* We consider markets with two values per attribute (left), four values per attribute (middle), and eight values per attribute (right); the legend is the same as in Figure A.11.



(a) Market with one attribute.

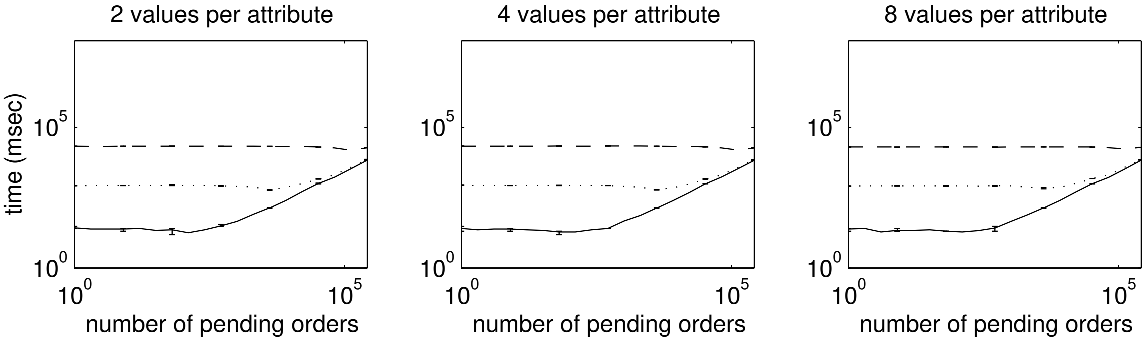


(b) Market with three attributes.

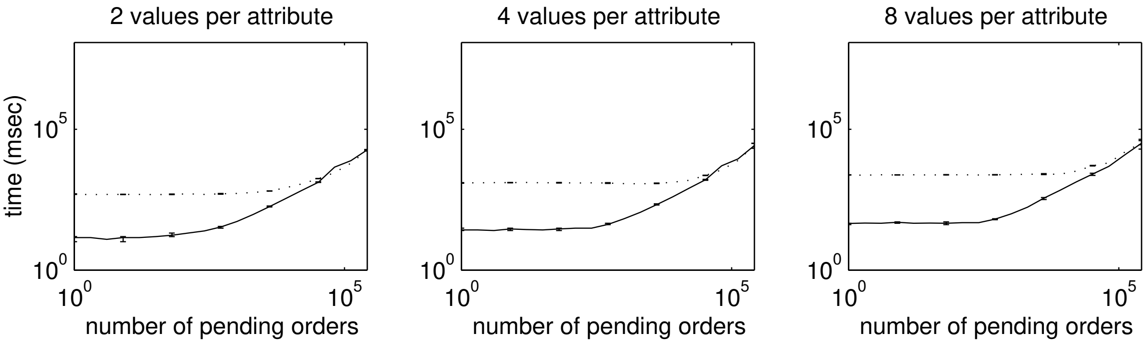


(c) Market with ten attributes.

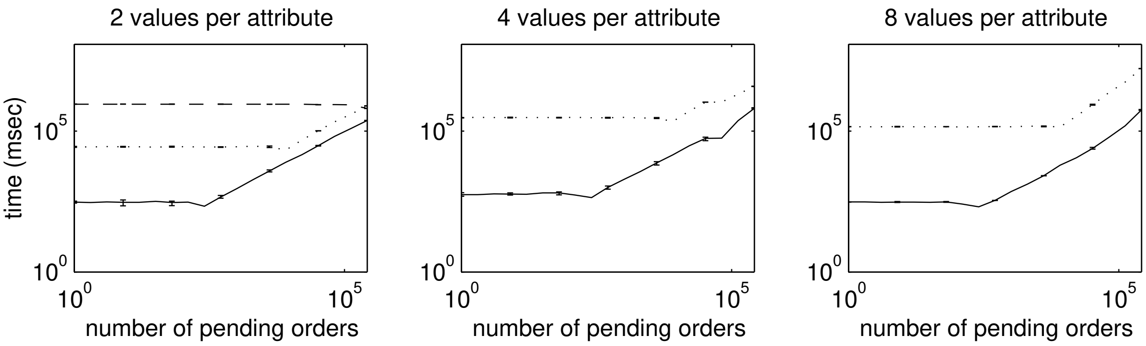
Figure A.16: Time of matching the pending orders, for *matching density of 0.01.* We consider markets with 16 values per attribute (left), 128 values per attribute (middle), and 1,024 values per attribute (right); the legend is the same as in Figure A.11.



(a) Market with one attribute.

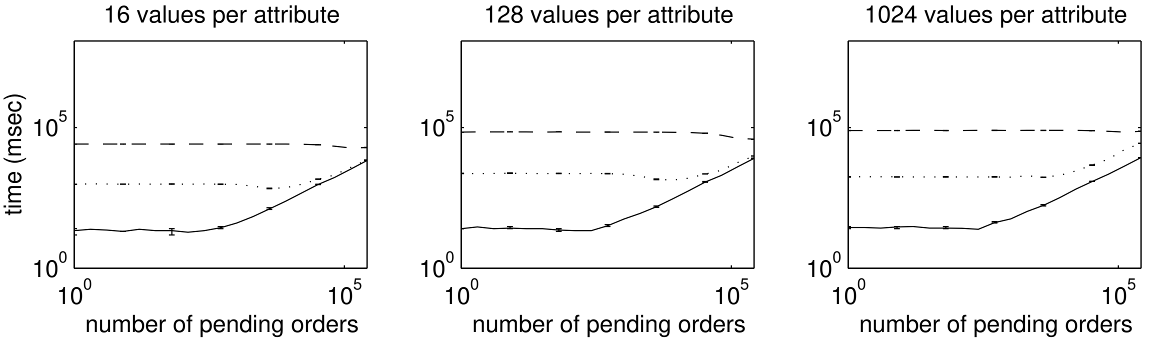


(b) Market with three attributes.

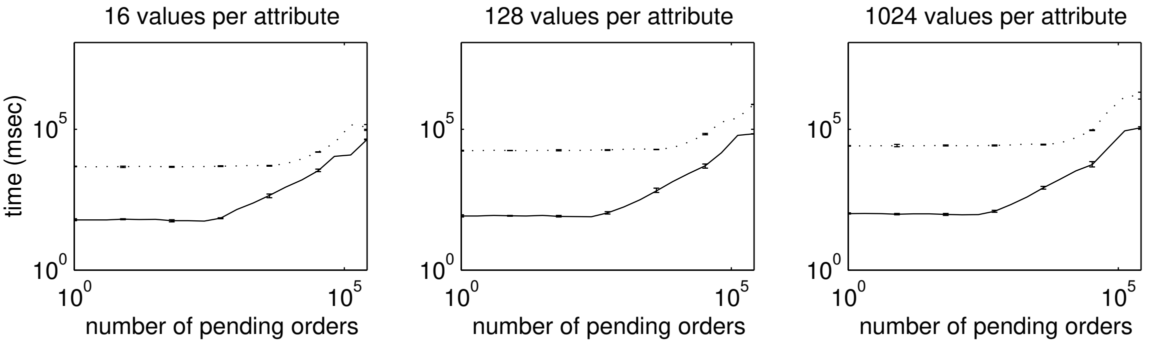


(c) Market with ten attributes.

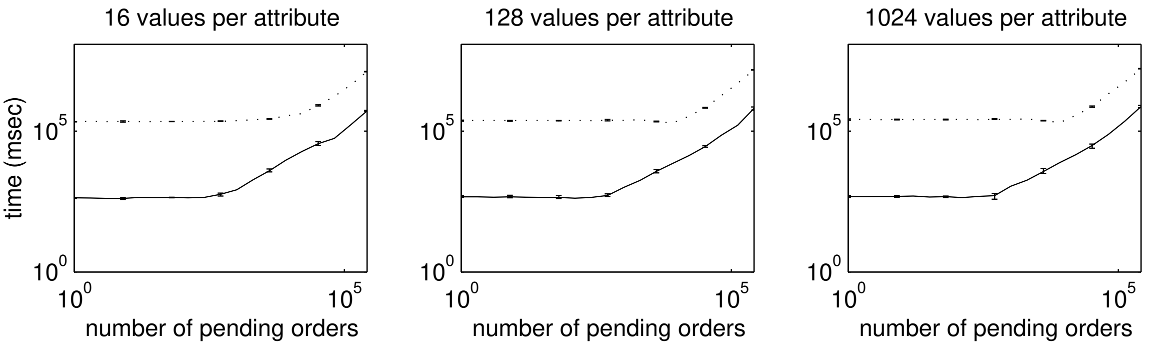
Figure A.17: Time of matching the pending orders, for *matching density of 0.1.* We consider markets with two values per attribute (left), four values per attribute (middle), and eight values per attribute (right); the legend is the same as in Figure A.11.



(a) Market with one attribute.

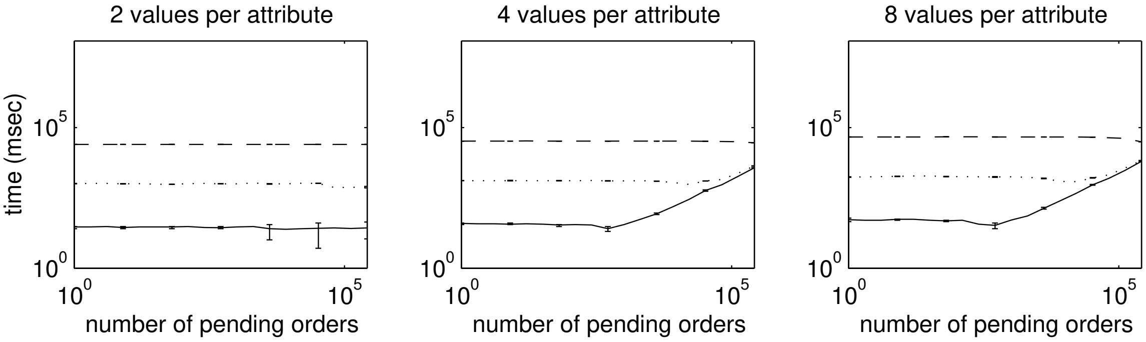


(b) Market with three attributes.

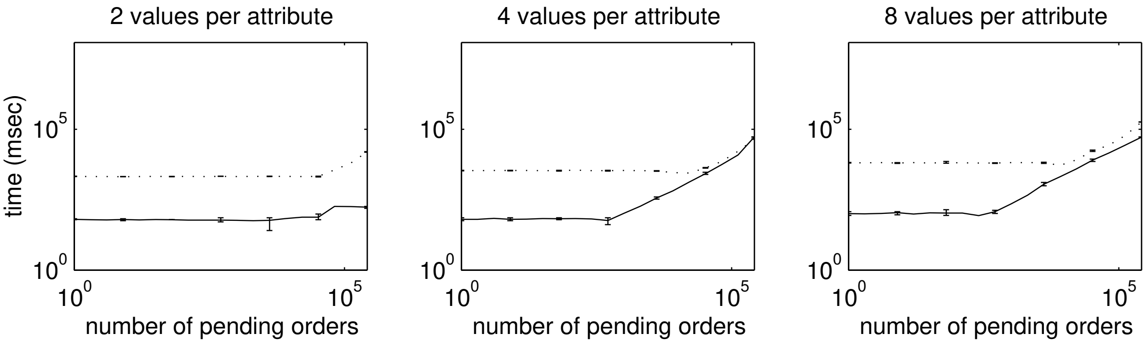


(c) Market with ten attributes.

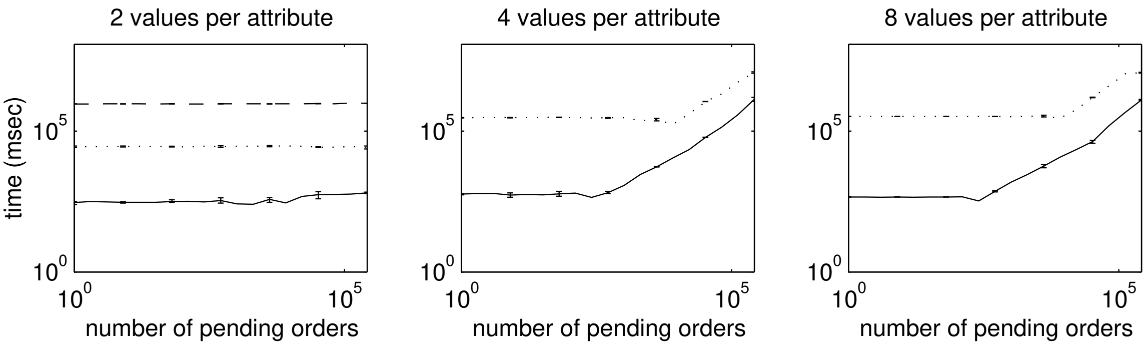
Figure A.18: Time of matching the pending orders, for *matching density of 0.1.* We consider markets with 16 values per attribute (left), 128 values per attribute (middle), and 1,024 values per attribute (right); the legend is the same as in Figure A.11.



(a) Market with one attribute.

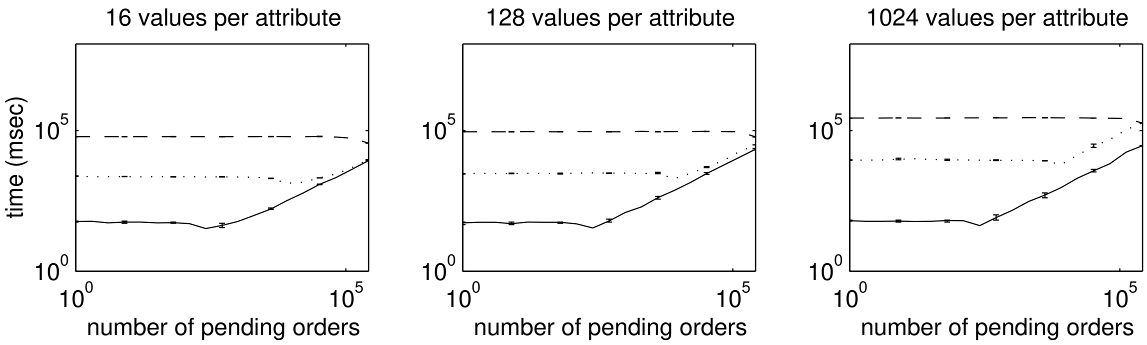


(b) Market with three attributes.

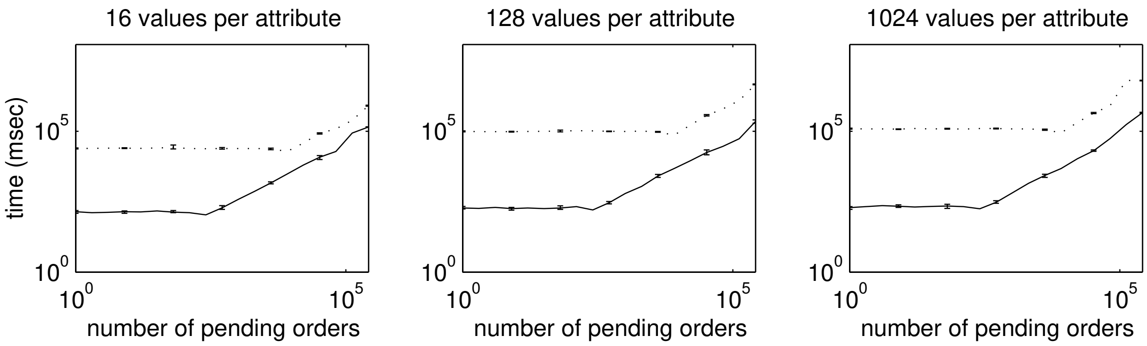


(c) Market with ten attributes.

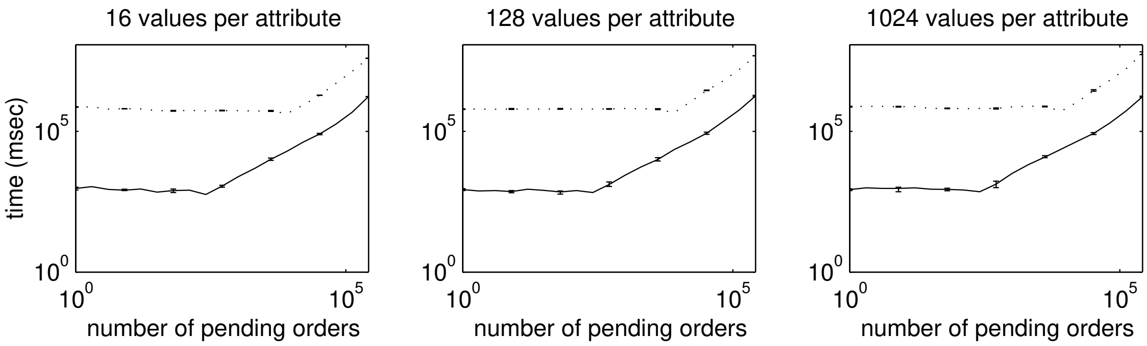
Figure A.19: Time of matching the pending orders, for *matching density of 1.* We consider markets with two values per attribute (left), four values per attribute (middle), and eight values per attribute (right); the legend is the same as in Figure A.11.



(a) Market with one attribute.



(b) Market with three attributes.

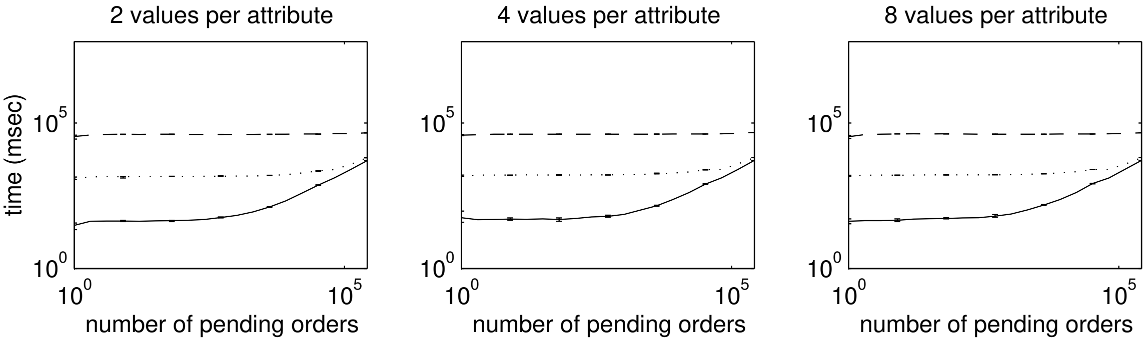


(c) Market with ten attributes.

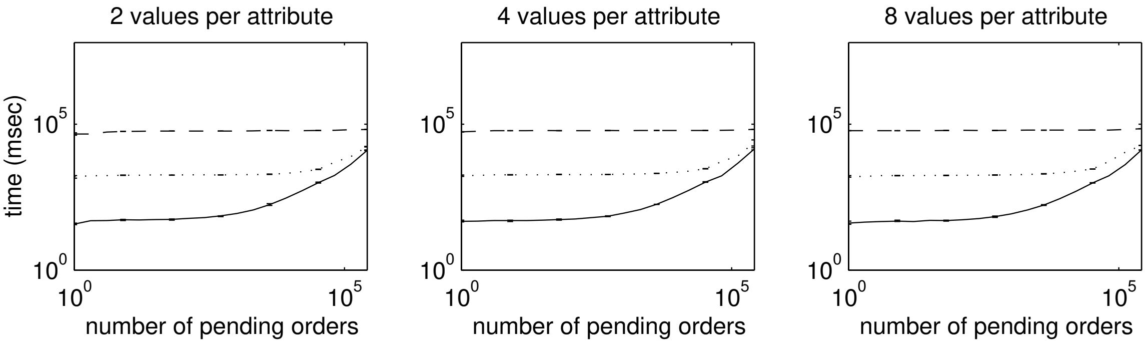
Figure A.20: Time of matching the pending orders, for *matching density of 1.* We consider markets with 16 values per attribute (left), 128 values per attribute (middle), and 1,024 values per attribute (right); the legend is the same as in Figure A.11.

A.3 Response Time

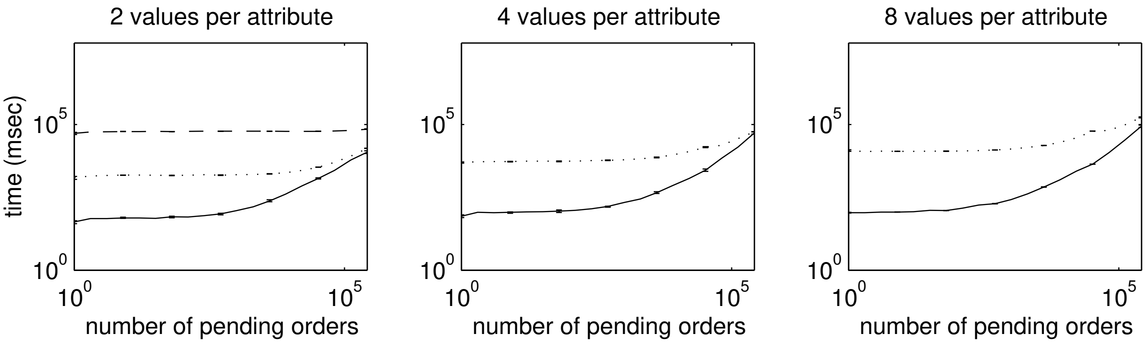
We plot the average time between placing a new order and receiving a response from the system. Recall that the response may not include a fill for the order; if the matcher does not find an immediate fill, it responds with a confirmation of the new order.



(a) Market with one attribute.

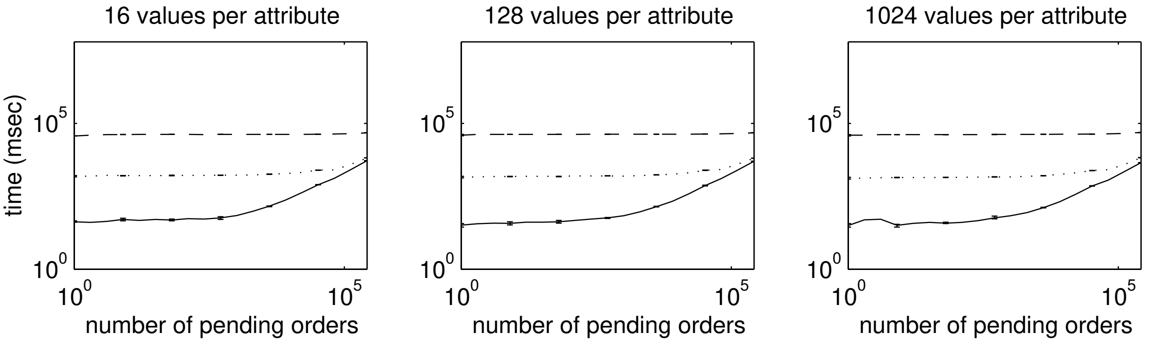


(b) Market with three attributes.

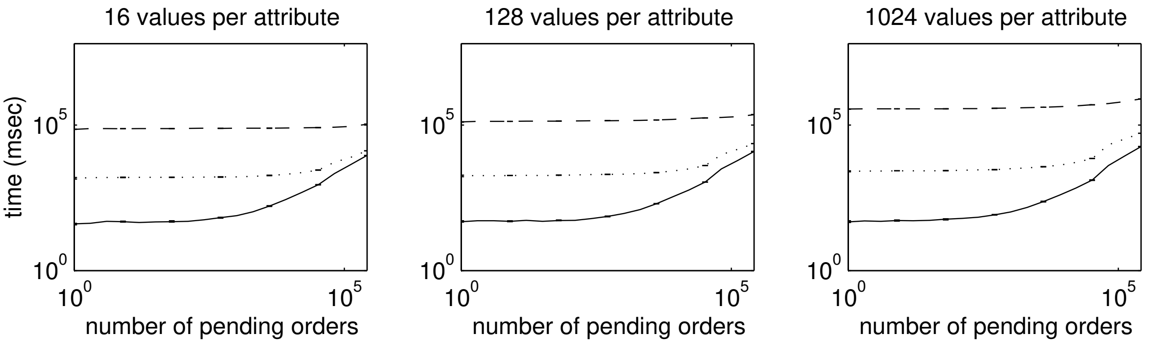


(c) Market with ten attributes.

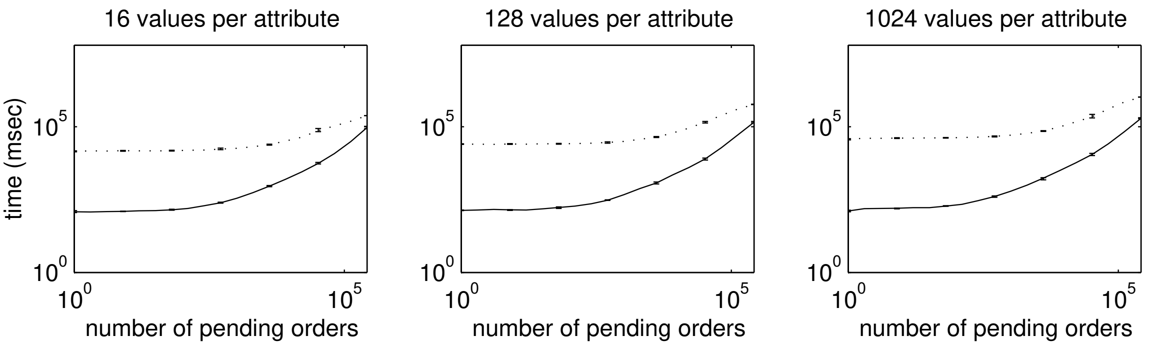
Figure A.21: Time between placing a new order and getting a response, for *matching density of 0.0001.* We consider markets with two values per attribute (left), four values per attribute (middle), and eight values per attribute (right). We show the dependency of the response time on the number of pending orders, for 256 new orders (solid lines), 8,192 orders (dotted lines), and 262,144 orders (dashed lines). Both horizontal and vertical scales are logarithmic, and the vertical bars mark the minimal and maximal values of the time measurements.



(a) Market with one attribute.

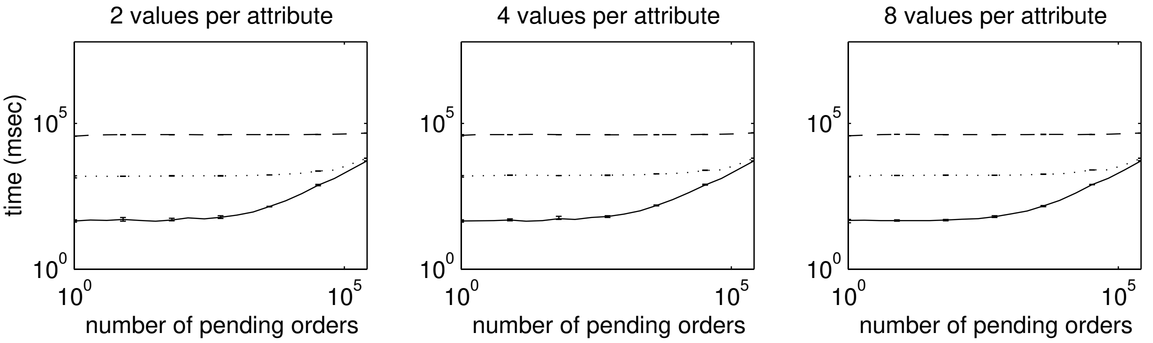


(b) Market with three attributes.

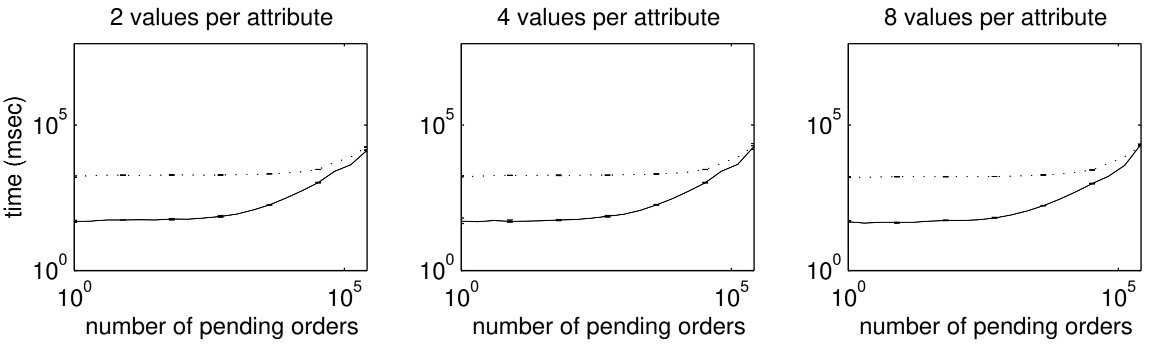


(c) Market with ten attributes.

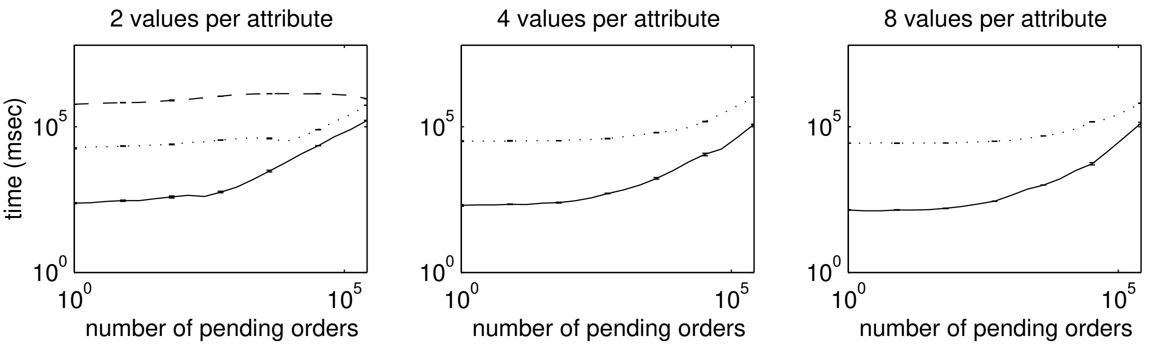
Figure A.22: Time between placing a new order and getting a response, for *matching density of 0.0001.* We consider markets with 16 values per attribute (left), 128 values per attribute (middle), and 1,024 values per attribute (right); the legend is the same as in Figure A.21.



(a) Market with one attribute.

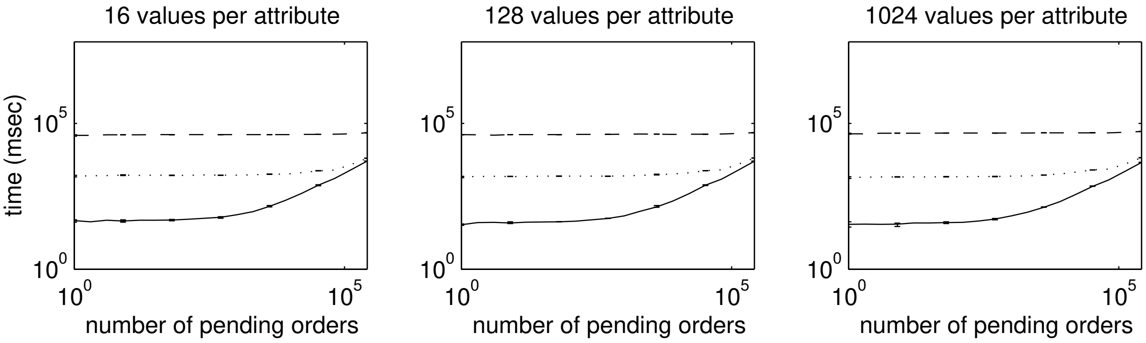


(b) Market with three attributes.

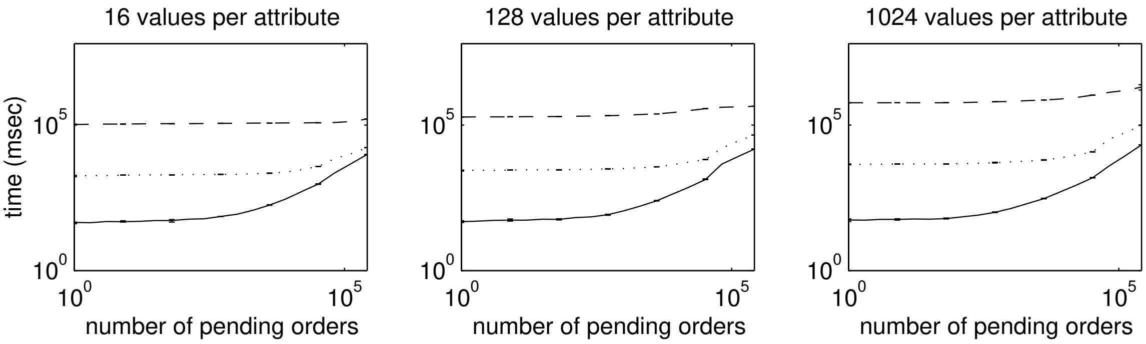


(c) Market with ten attributes.

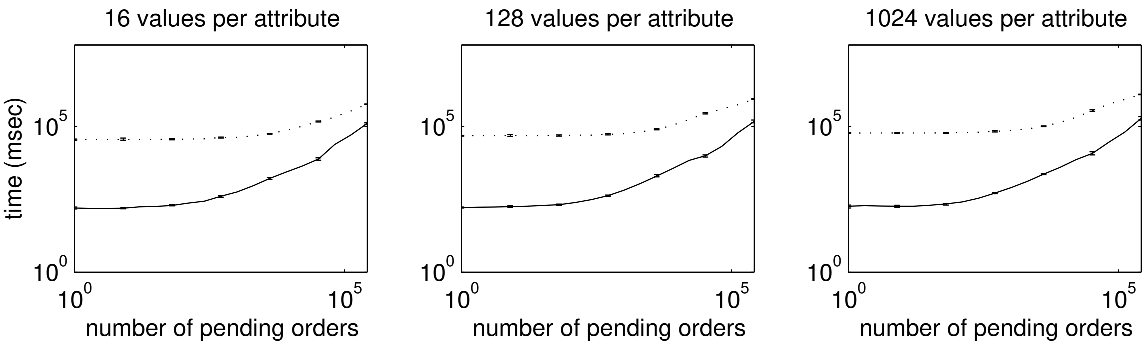
Figure A.23: Time between placing a new order and getting a response, for *matching density of 0.001.* We consider markets with two values per attribute (left), four values per attribute (middle), and eight values per attribute (right); the legend is the same as in Figure A.21.



(a) Market with one attribute.

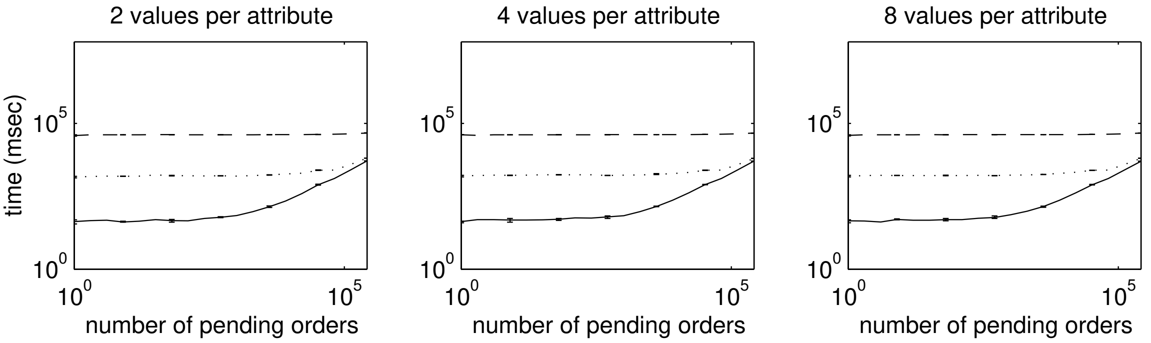


(b) Market with three attributes.

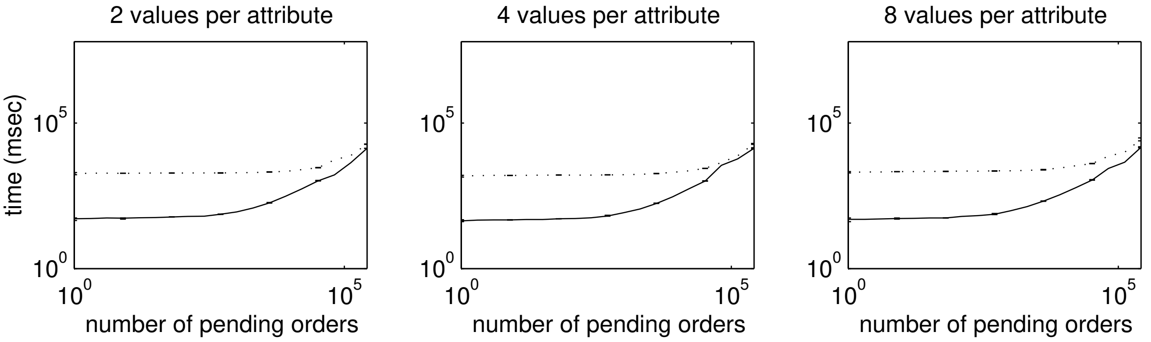


(c) Market with ten attributes.

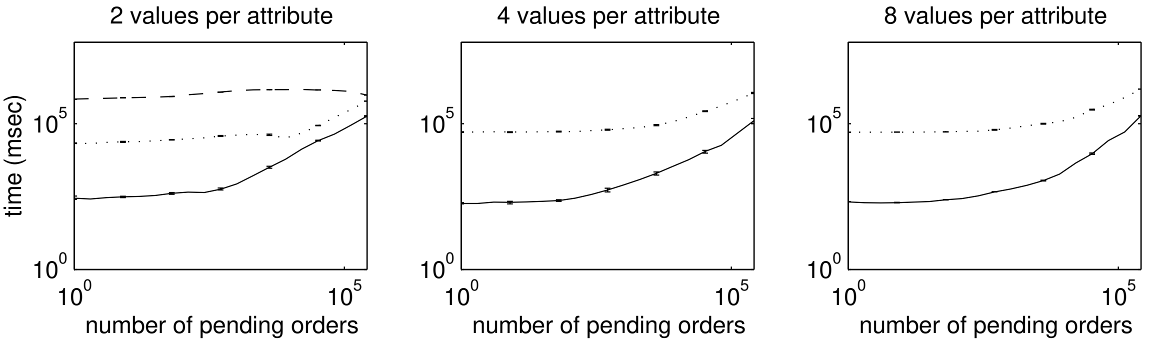
Figure A.24: Time between placing a new order and getting a response, for *matching density of 0.001.* We consider markets with 16 values per attribute (left), 128 values per attribute (middle), and 1,024 values per attribute (right); the legend is the same as in Figure A.21.



(a) Market with one attribute.

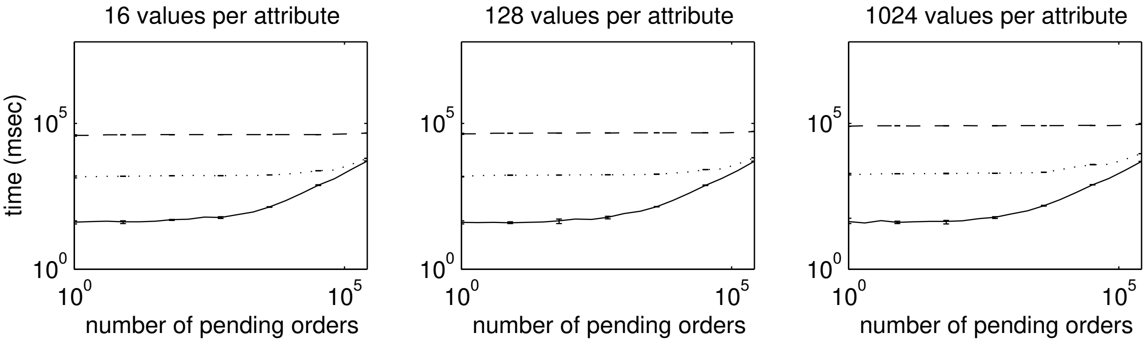


(b) Market with three attributes.

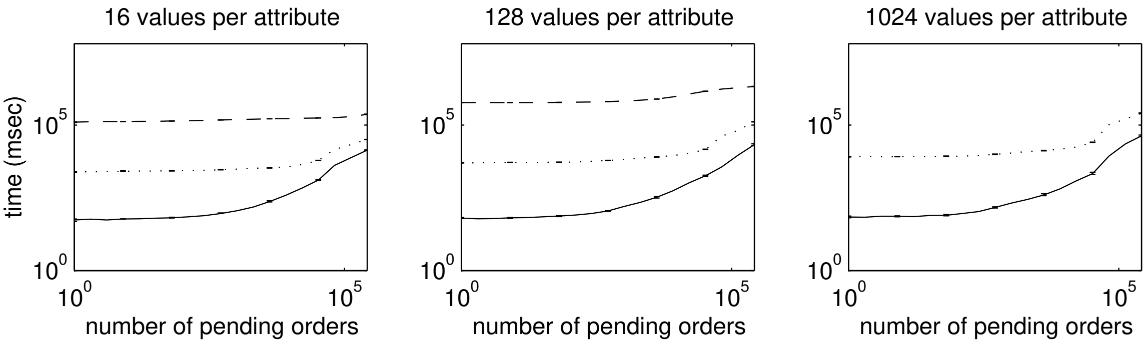


(c) Market with ten attributes.

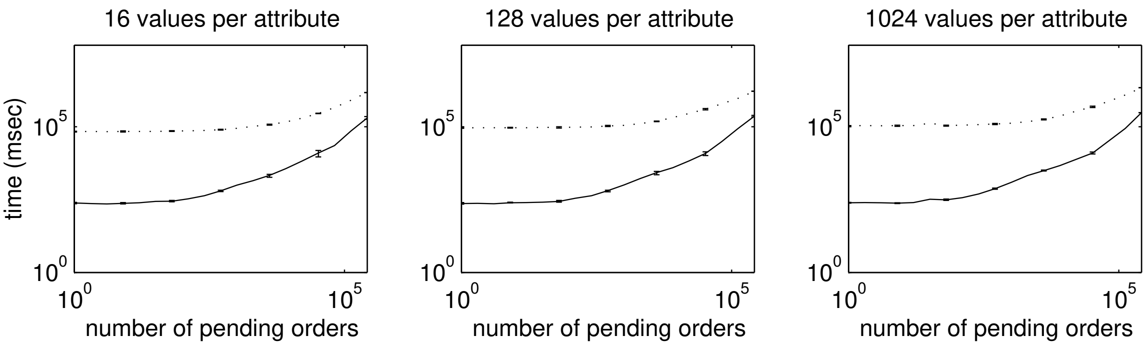
Figure A.25: Time between placing a new order and getting a response, for *matching density of 0.01.* We consider markets with two values per attribute (left), four values per attribute (middle), and eight values per attribute (right); the legend is the same as in Figure A.21.



(a) Market with one attribute.

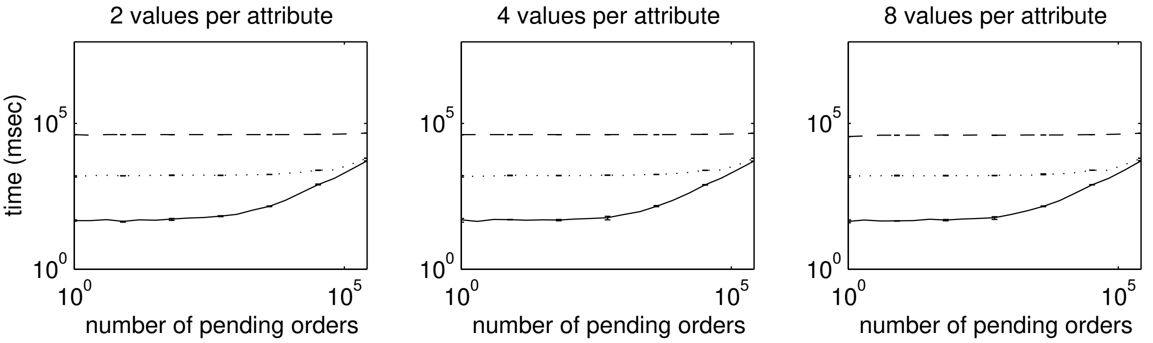


(b) Market with three attributes.

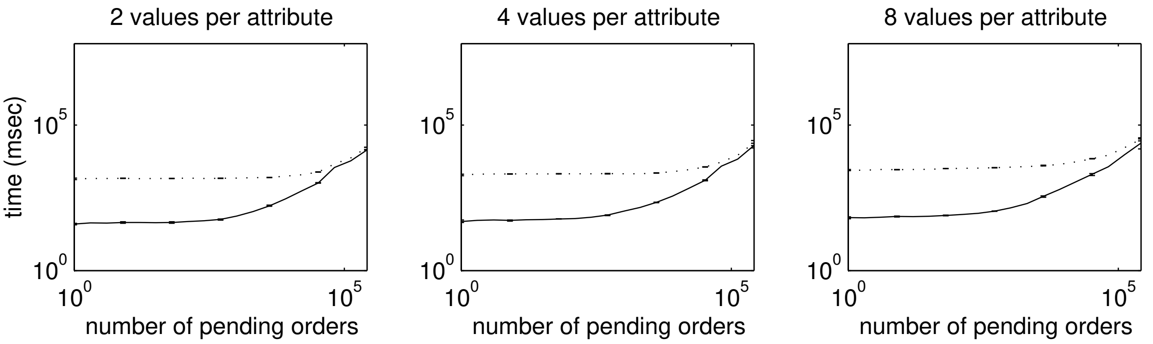


(c) Market with ten attributes.

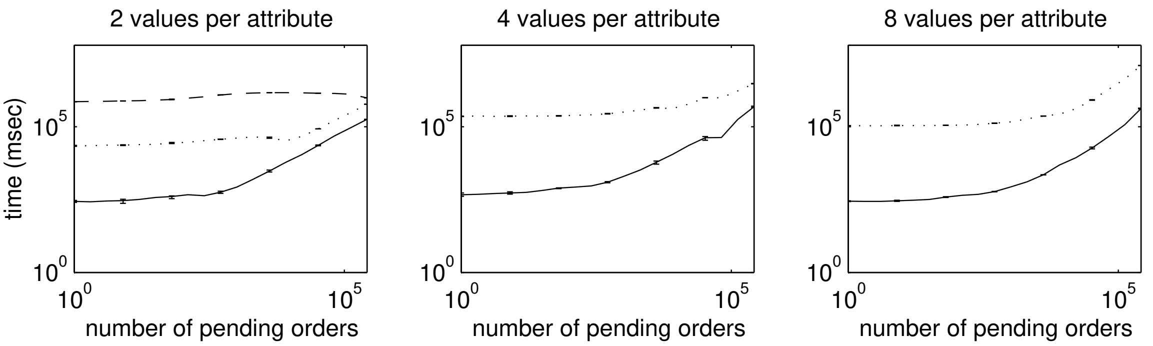
Figure A.26: Time between placing a new order and getting a response, for *matching density of 0.01.* We consider markets with 16 values per attribute (left), 128 values per attribute (middle), and 1,024 values per attribute (right); the legend is the same as in Figure A.21.



(a) Market with one attribute.

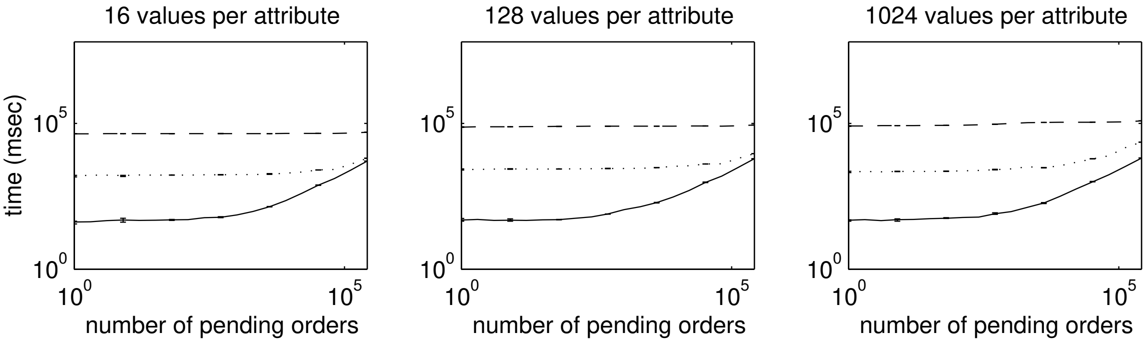


(b) Market with three attributes.

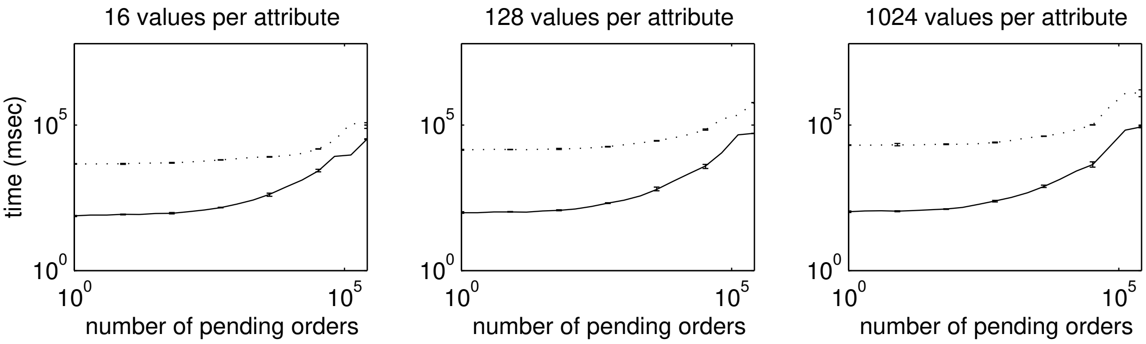


(c) Market with ten attributes.

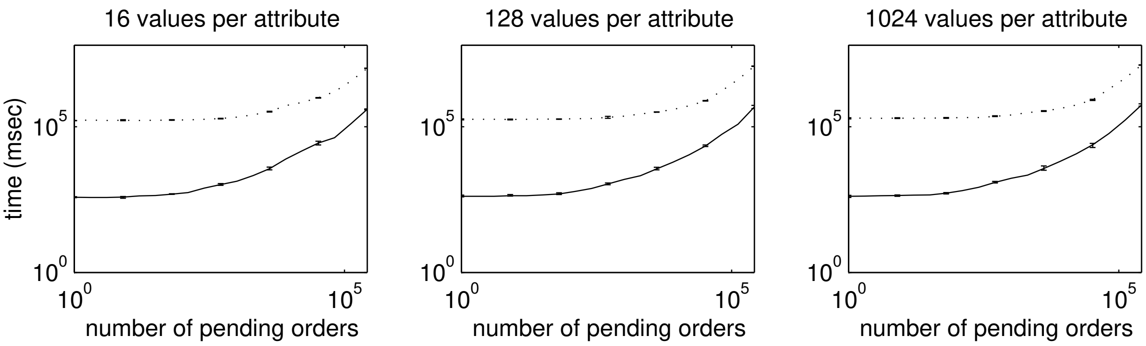
Figure A.27: Time between placing a new order and getting a response, for *matching density of 0.1.* We consider markets with two values per attribute (left), four values per attribute (middle), and eight values per attribute (right); the legend is the same as in Figure A.21.



(a) Market with one attribute.

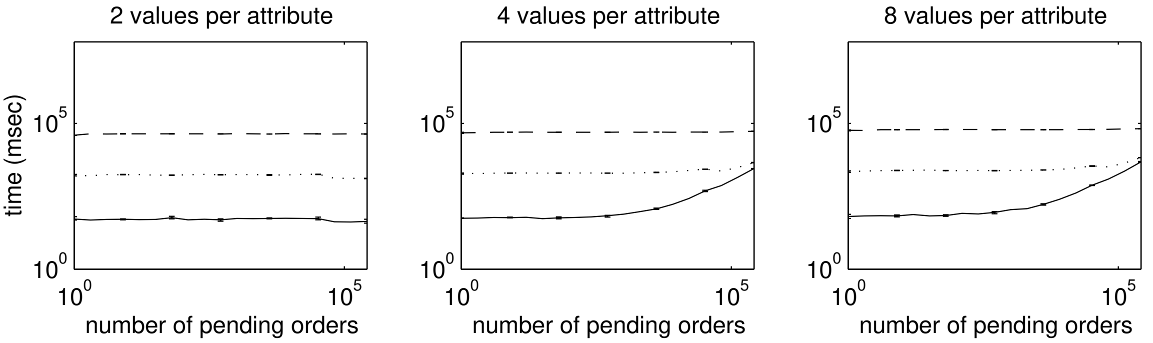


(b) Market with three attributes.

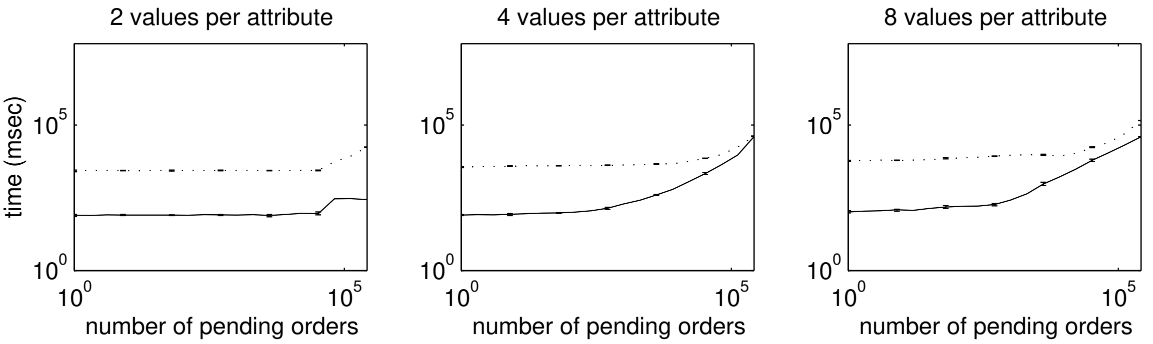


(c) Market with ten attributes.

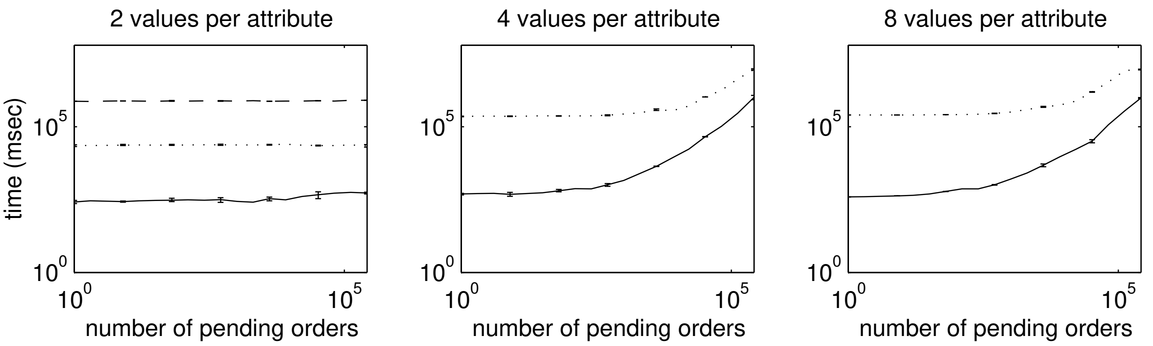
Figure A.28: Time between placing a new order and getting a response, for *matching density of 0.1.* We consider markets with 16 values per attribute (left), 128 values per attribute (middle), and 1,024 values per attribute (right); the legend is the same as in Figure A.21.



(a) Market with one attribute.

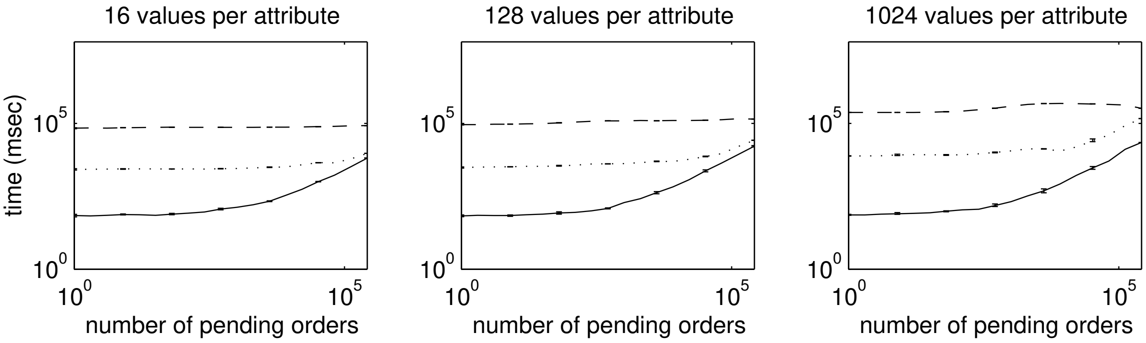


(b) Market with three attributes.

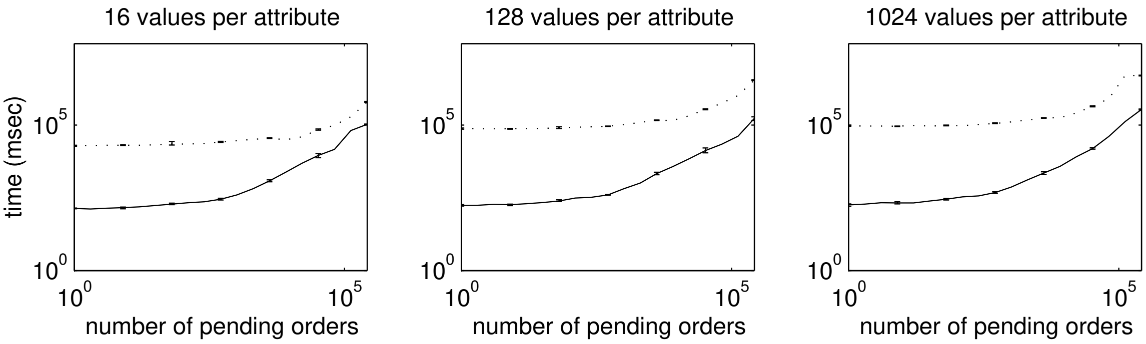


(c) Market with ten attributes.

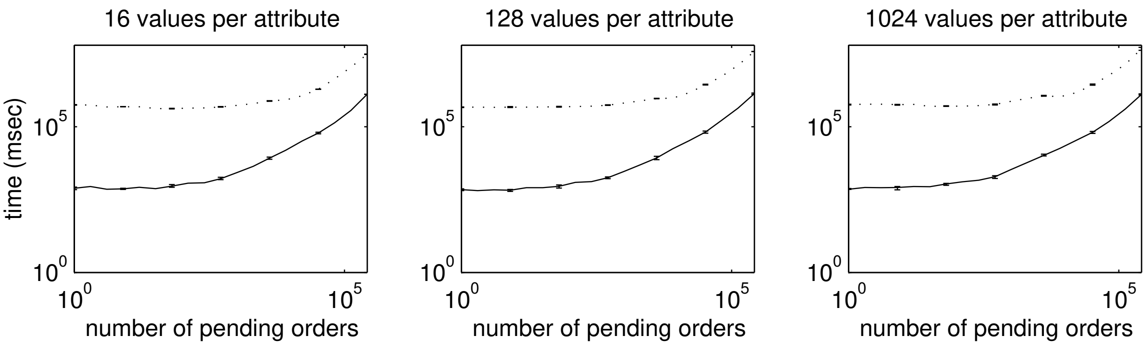
Figure A.29: Time between placing a new order and getting a response, for *matching density of 1.* We consider markets with two values per attribute (left), four values per attribute (middle), and eight values per attribute (right); the legend is the same as in Figure A.21.



(a) Market with one attribute.



(b) Market with three attributes.

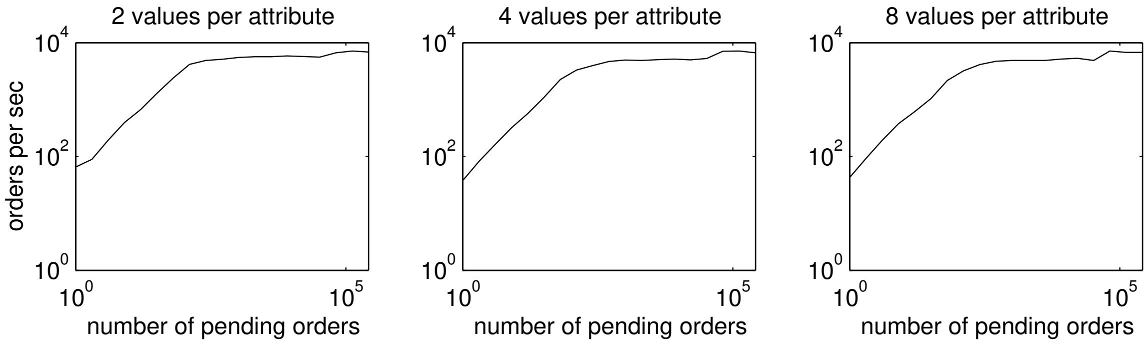


(c) Market with ten attributes.

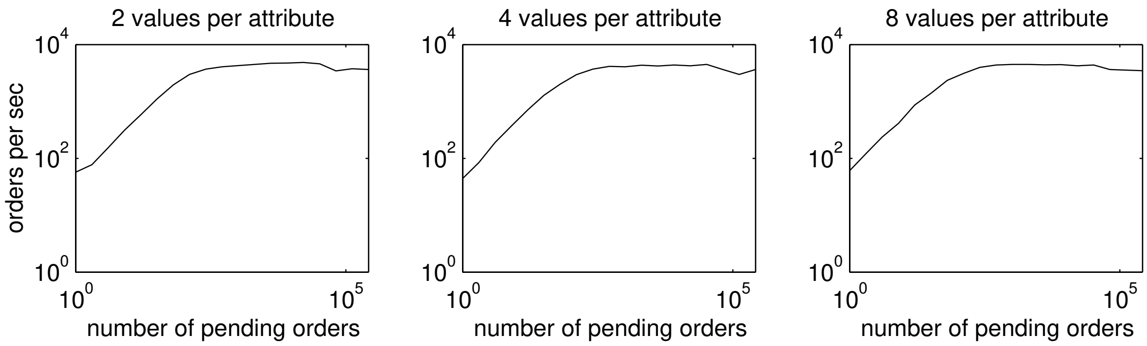
Figure A.30: Time between placing a new order and getting a response, for *matching density of 1.* We consider markets with 16 values per attribute (left), 128 values per attribute (middle), and 1,024 values per attribute (right); the legend is the same as in Figure A.21.

A.4 Maximal Throughput

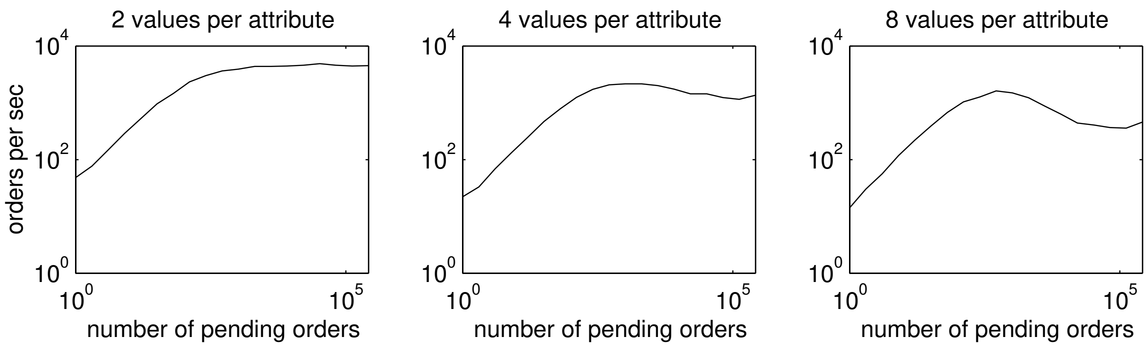
We show the limit on the number of new orders per second. If the matcher gets more orders, it has to reject some of them, to prevent overflow of the message queue.



(a) Market with one attribute.

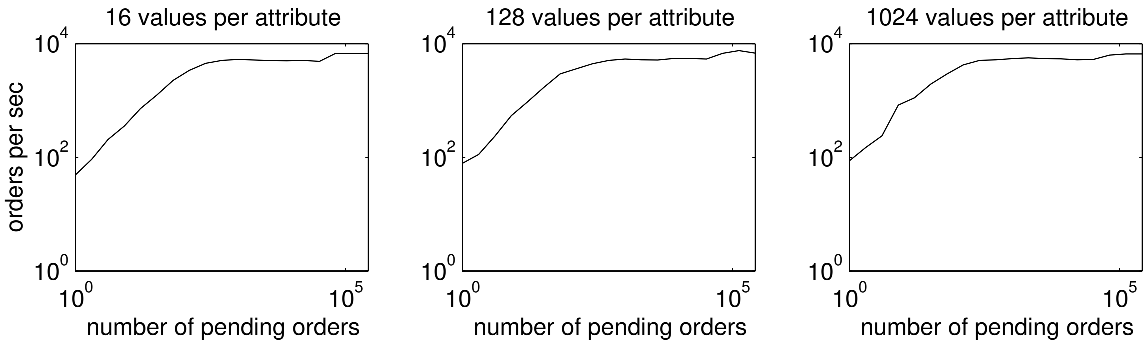


(b) Market with three attributes.

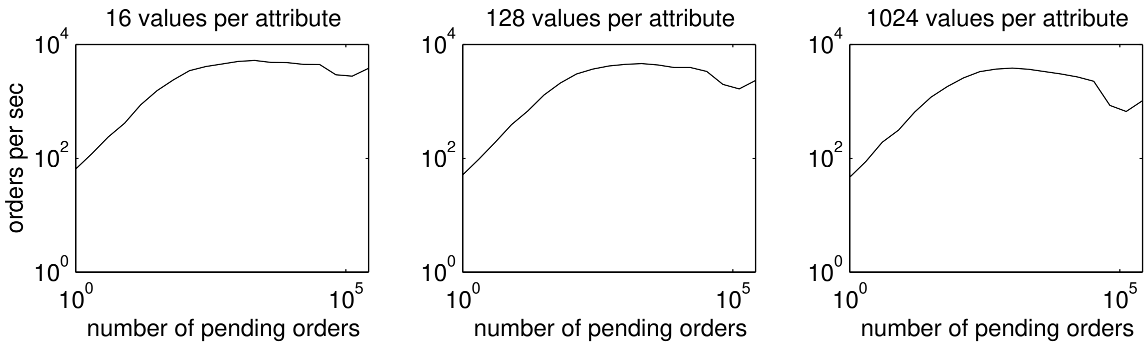


(c) Market with ten attributes.

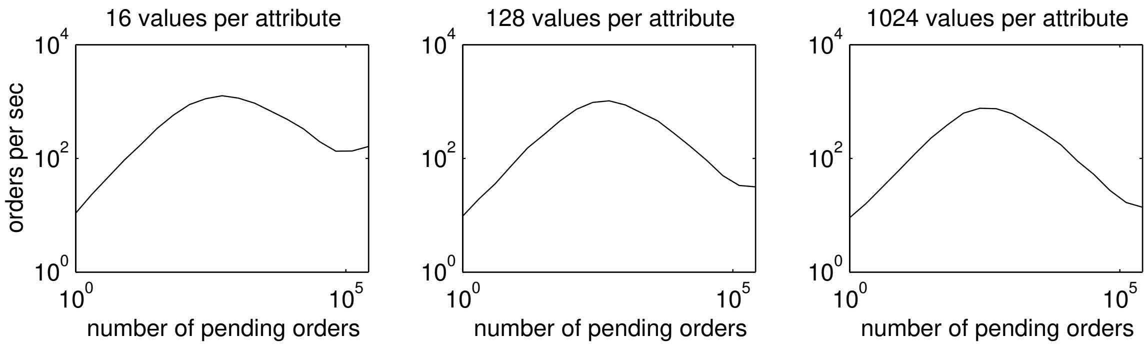
Figure A.31: Maximal number of orders per second, for *matching density of 0.0001.* We consider markets with two values per attribute (left), four values per attribute (middle), and eight values per attribute (right). We show the dependency of the system's throughput on the number of pending orders. Both horizontal and vertical scales are logarithmic.



(a) Market with one attribute.

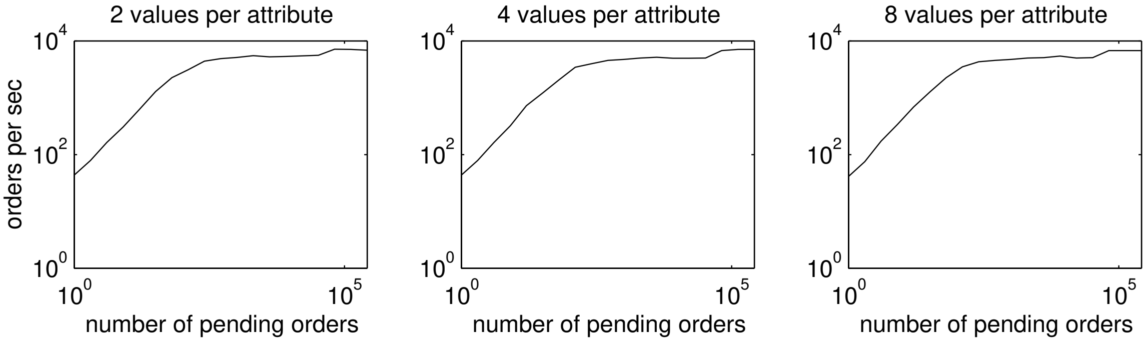


(b) Market with three attributes.

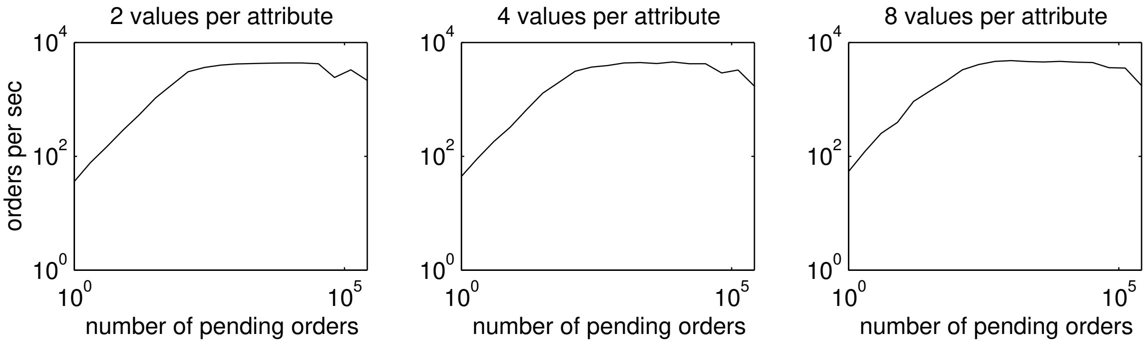


(c) Market with ten attributes.

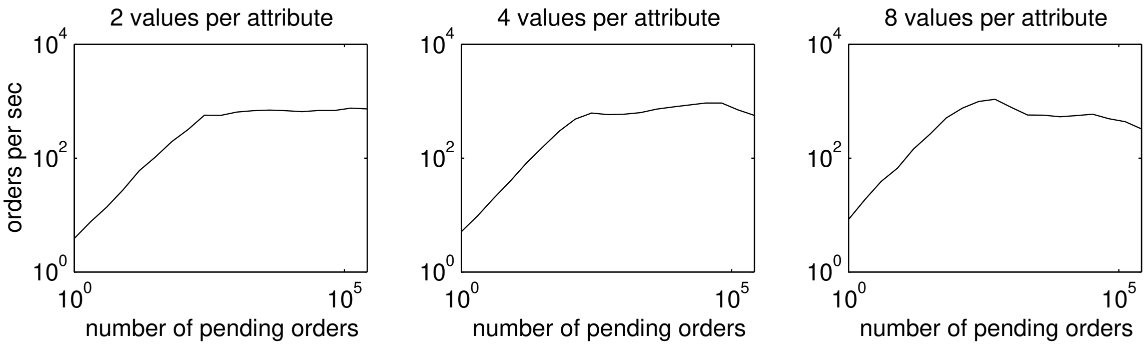
Figure A.32: Maximal number of orders per second, for *matching density of 0.0001.* We consider markets with 16 values per attribute (left), 128 values per attribute (middle), and 1,024 values per attribute (right).



(a) Market with one attribute.

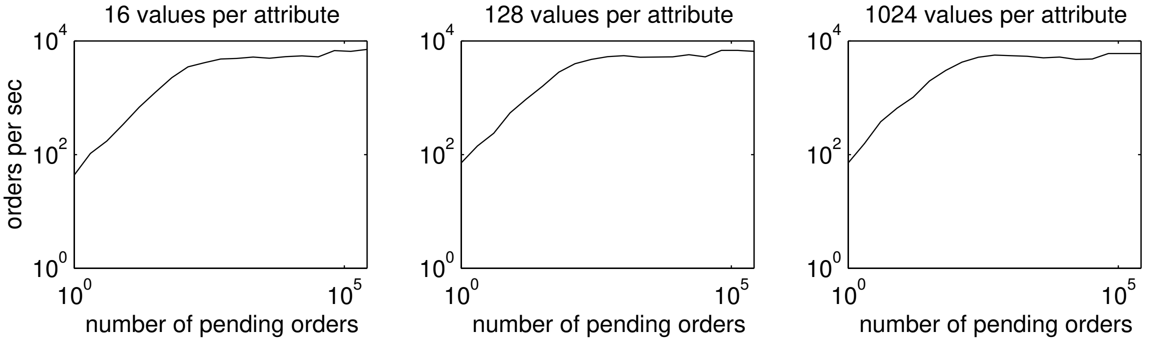


(b) Market with three attributes.

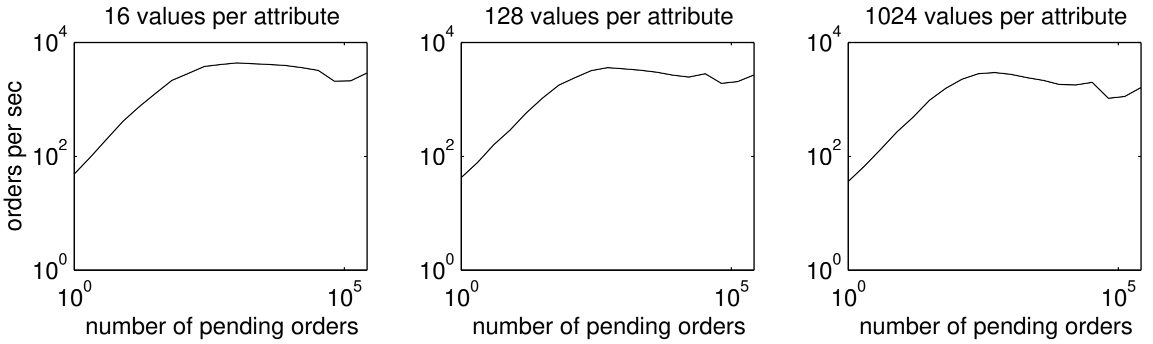


(c) Market with ten attributes.

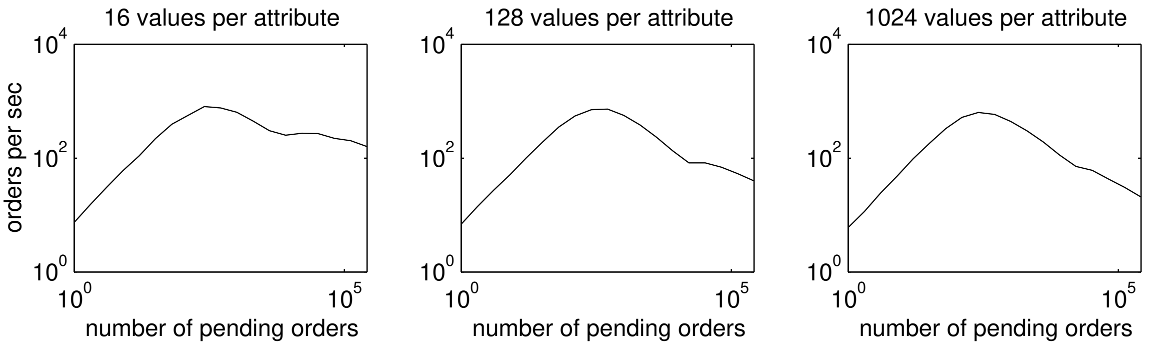
Figure A.33: Maximal number of orders per second, for *matching density of 0.001.* We consider markets with two values per attribute (left), four values per attribute (middle), and eight values per attribute (right).



(a) Market with one attribute.

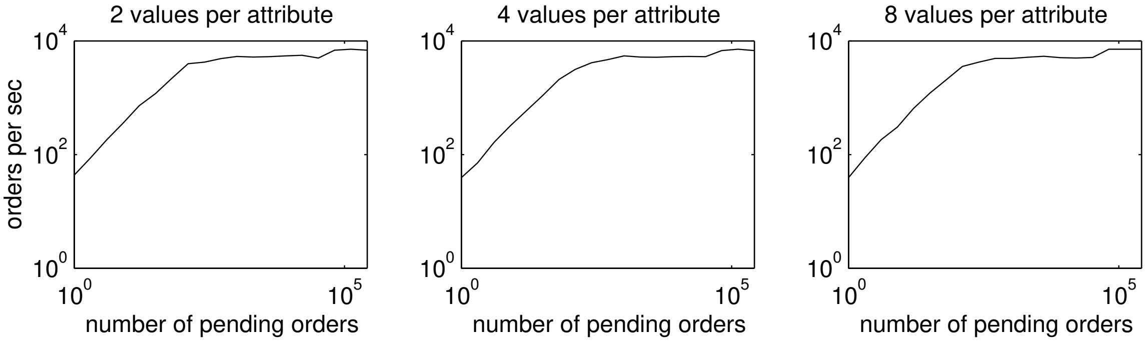


(b) Market with three attributes.

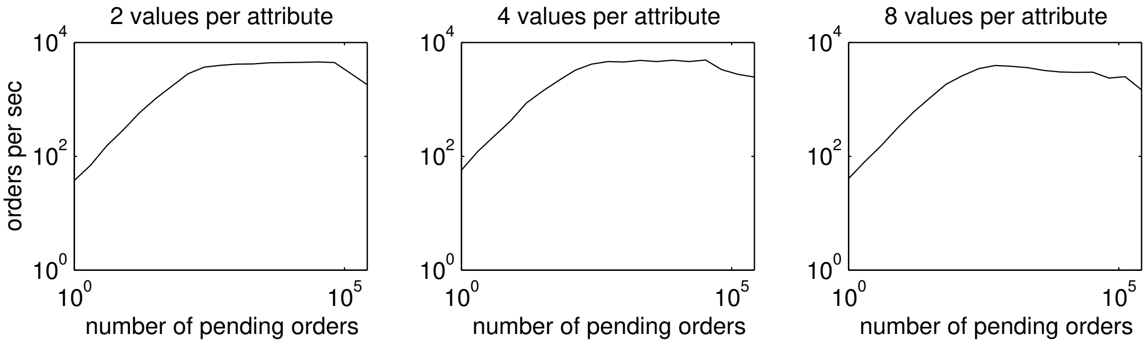


(c) Market with ten attributes.

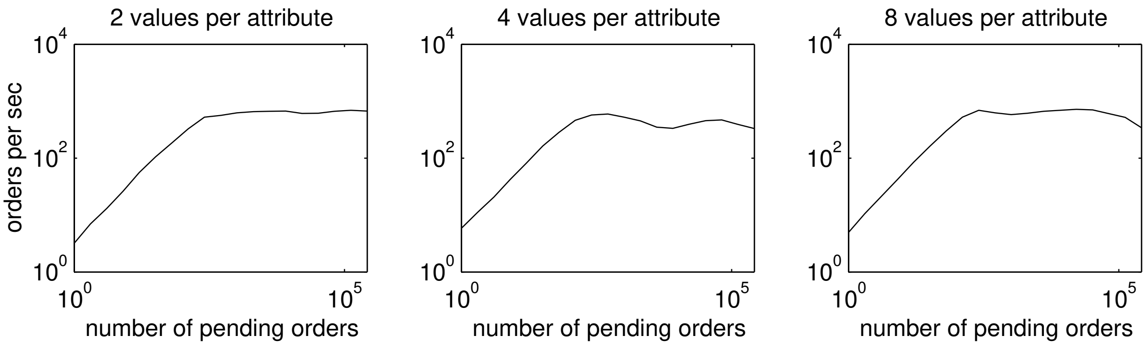
Figure A.34: Maximal number of orders per second, for *matching density of 0.001.* We consider markets with 16 values per attribute (left), 128 values per attribute (middle), and 1,024 values per attribute (right).



(a) Market with one attribute.

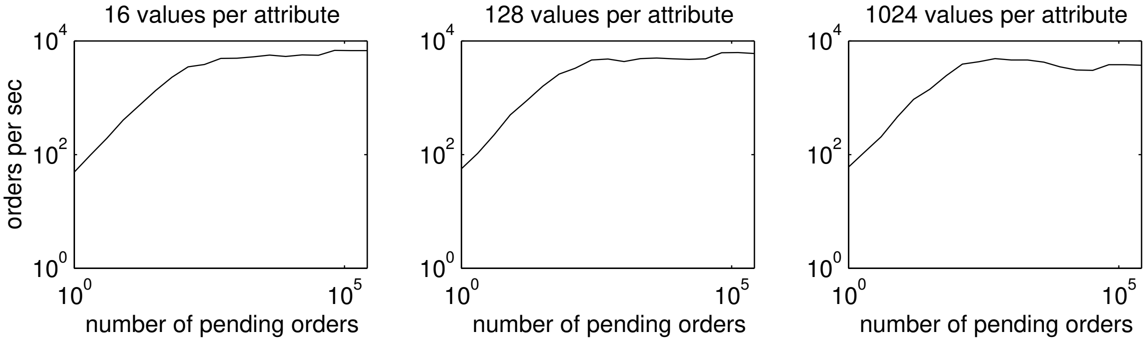


(b) Market with three attributes.

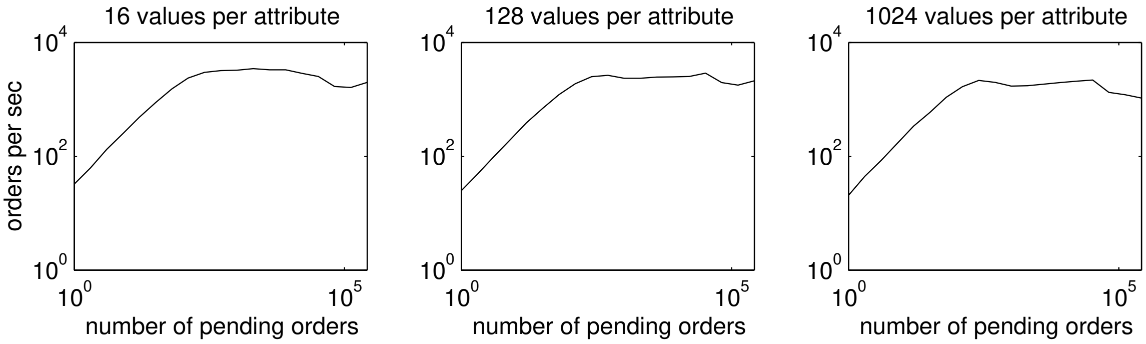


(c) Market with ten attributes.

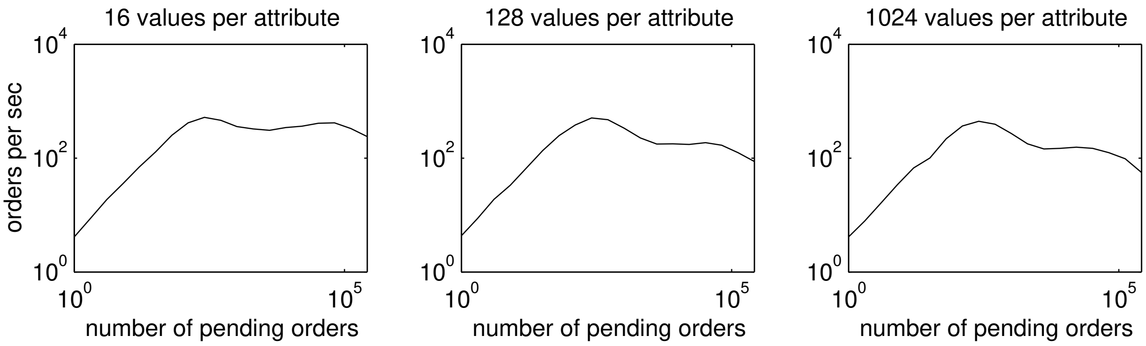
Figure A.35: Maximal number of orders per second, for *matching density of 0.01.* We consider markets with two values per attribute (left), four values per attribute (middle), and eight values per attribute (right).



(a) Market with one attribute.

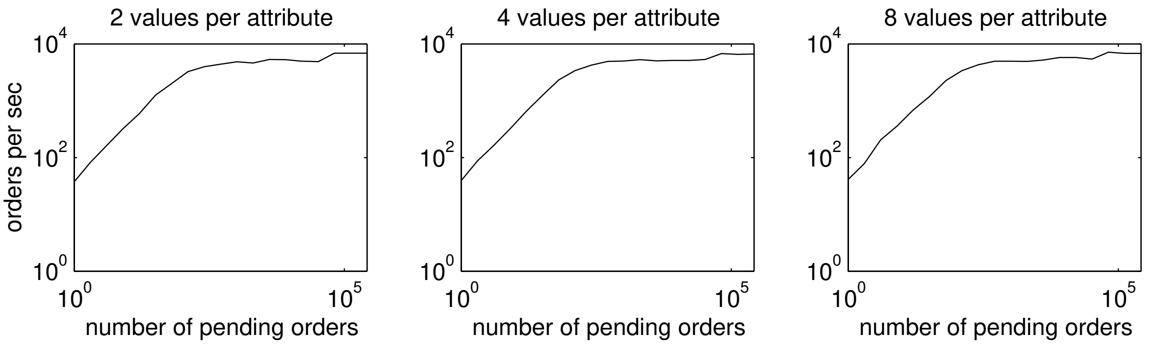


(b) Market with three attributes.

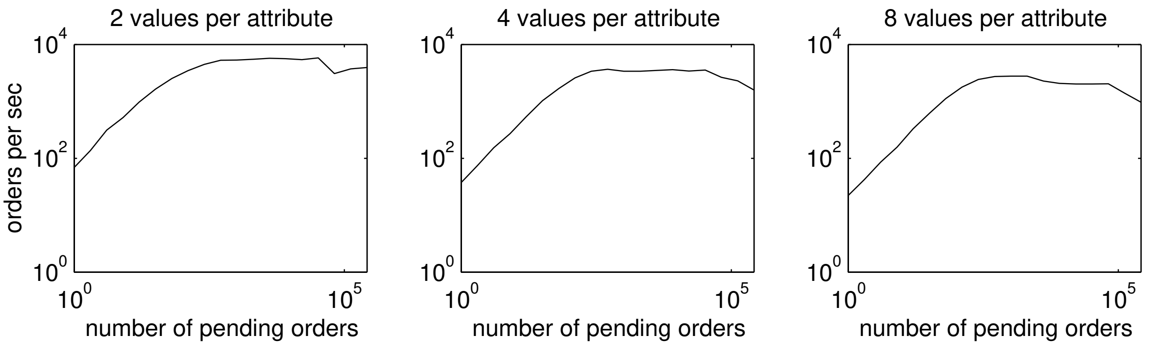


(c) Market with ten attributes.

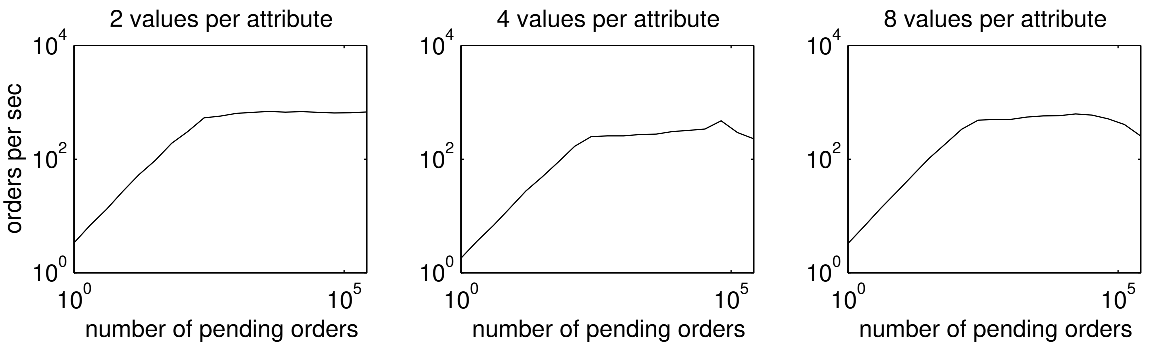
Figure A.36: Maximal number of orders per second, for *matching density of 0.01.* We consider markets with 16 values per attribute (left), 128 values per attribute (middle), and 1,024 values per attribute (right).



(a) Market with one attribute.

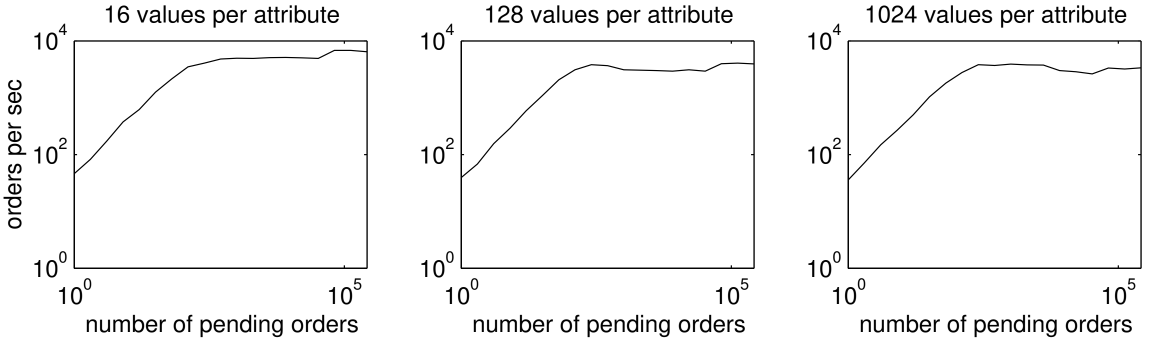


(b) Market with three attributes.

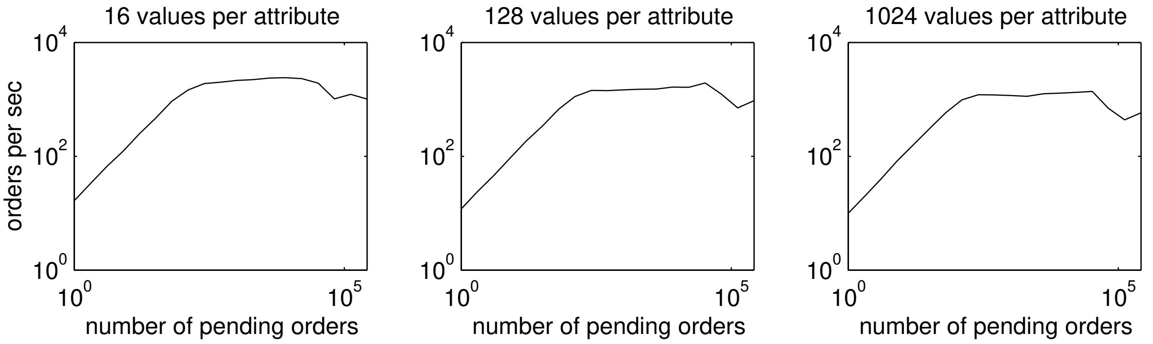


(c) Market with ten attributes.

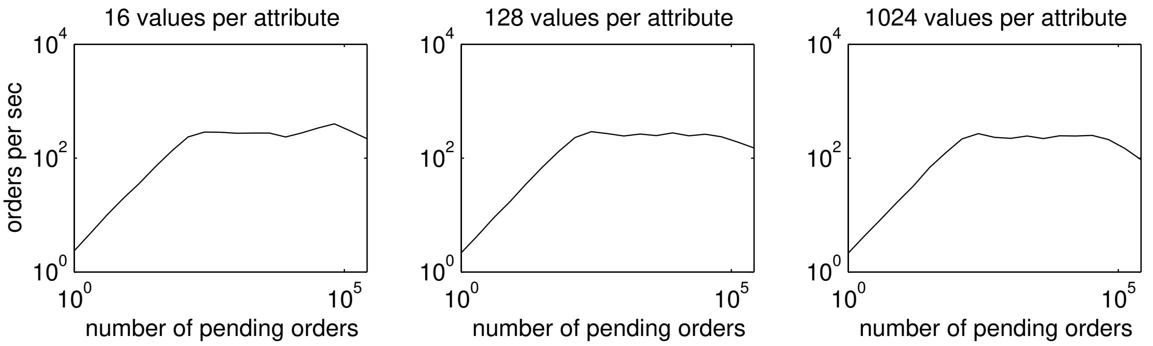
Figure A.37: Maximal number of orders per second, for *matching density of 0.1.* We consider markets with two values per attribute (left), four values per attribute (middle), and eight values per attribute (right).



(a) Market with one attribute.

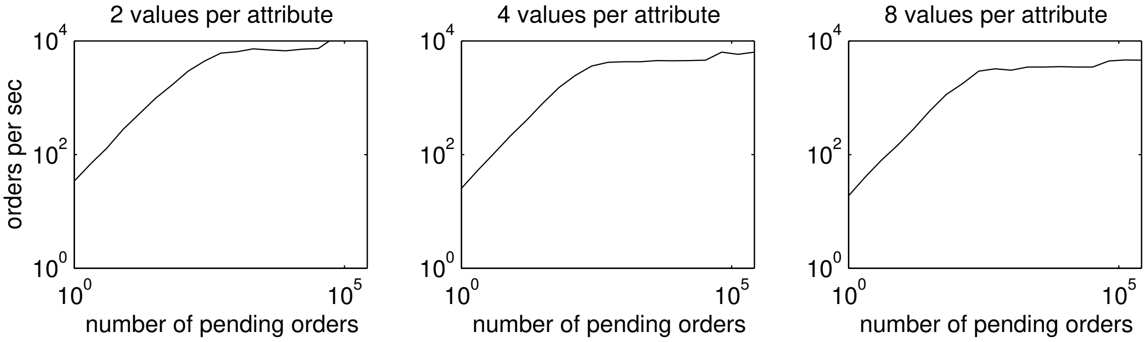


(b) Market with three attributes.

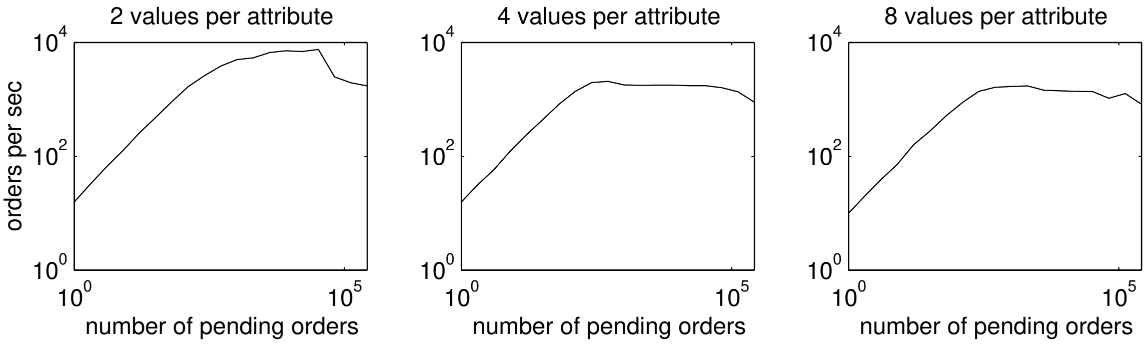


(c) Market with ten attributes.

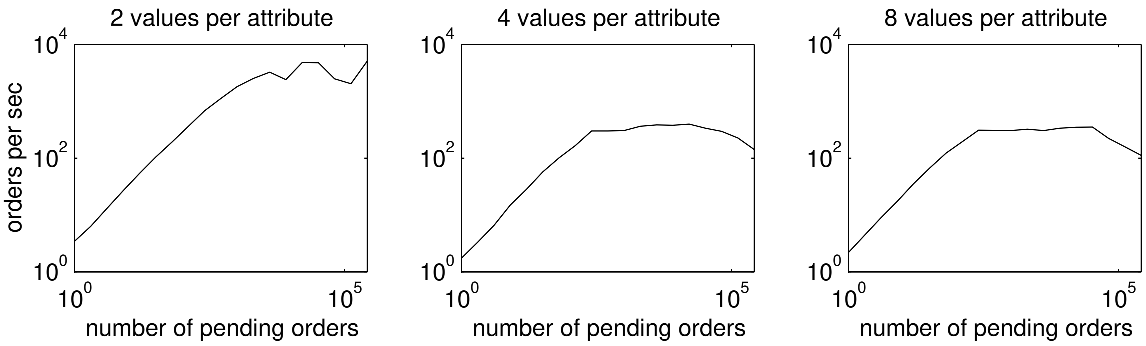
Figure A.38: Maximal number of orders per second, for *matching density of 0.1.* We consider markets with 16 values per attribute (left), 128 values per attribute (middle), and 1,024 values per attribute (right).



(a) Market with one attribute.

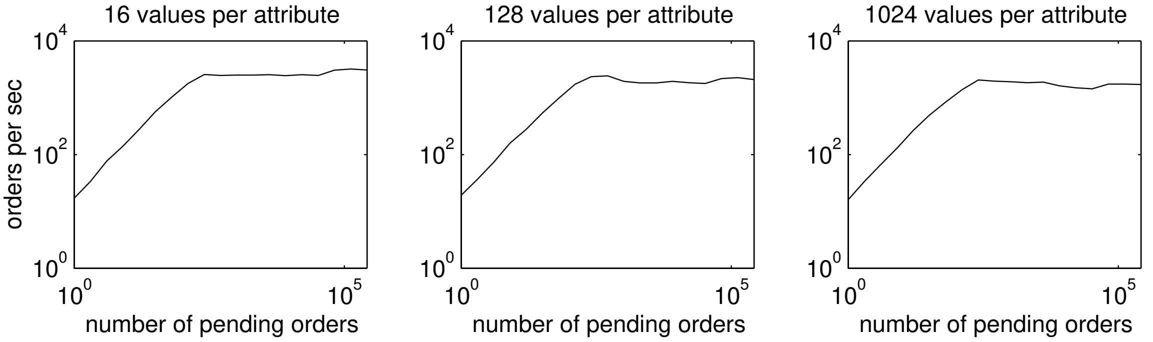


(b) Market with three attributes.

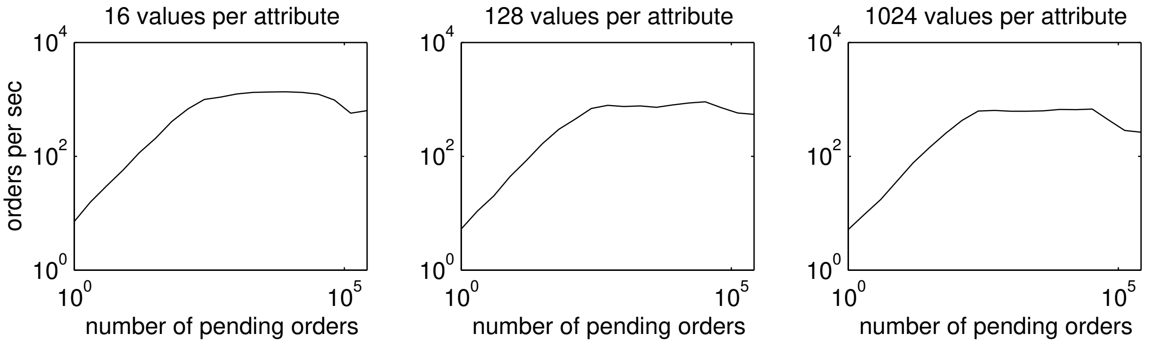


(c) Market with ten attributes.

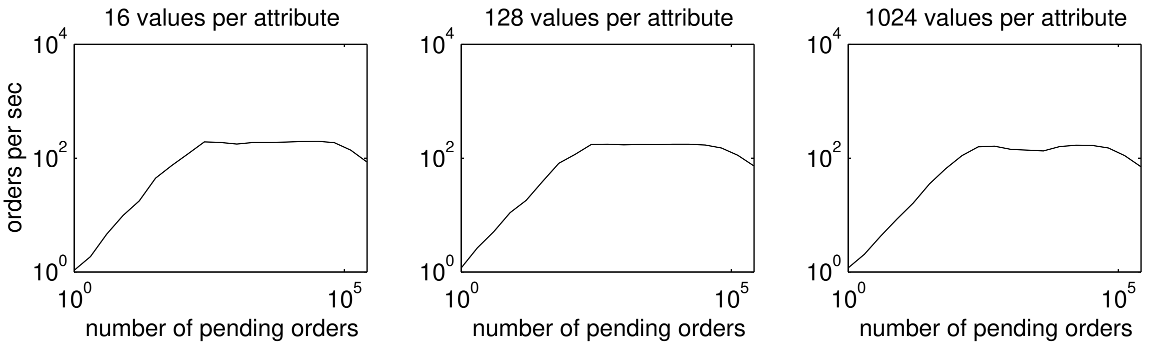
Figure A.39: Maximal number of orders per second, for *matching density of 1.* We consider markets with two values per attribute (left), four values per attribute (middle), and eight values per attribute (right).



(a) Market with one attribute.



(b) Market with three attributes.



(c) Market with ten attributes.

Figure A.40: Maximal number of orders per second, for *matching density of 1.* We consider markets with 16 values per attribute (left), 128 values per attribute (middle), and 1,024 values per attribute (right).

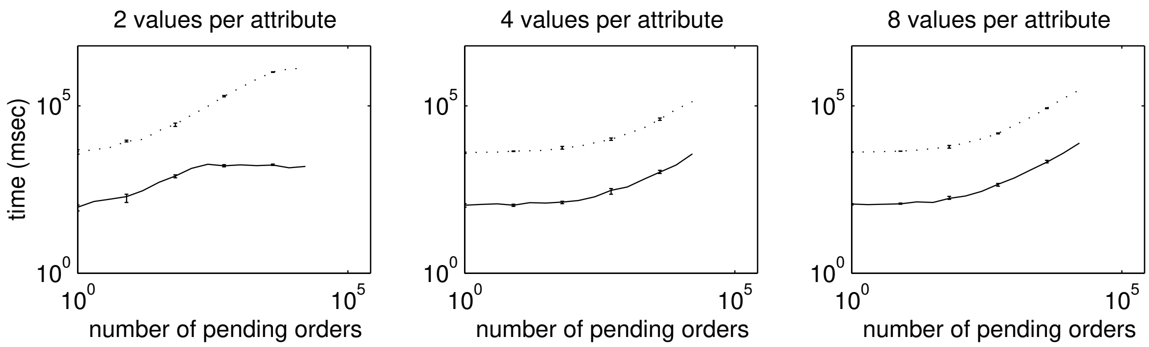
Appendix B

Experiments with a Large Number of Attributes

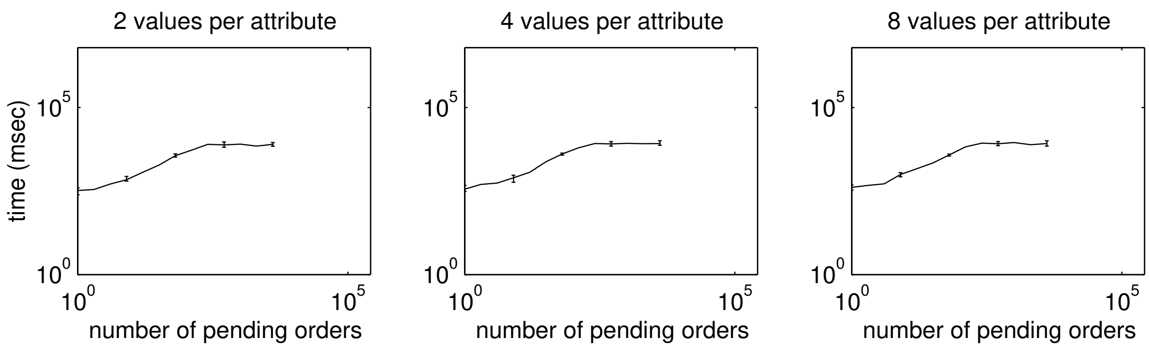
We give results for markets with a large number of attributes; specifically, we experiment with thirty and one hundred attributes. We have explained the setup for these experiments in Section 4.1.

B.1 Processing Time

We show the mean time of processing new orders in the first half of the system's top-level loop (see Figure 3.2a), for various settings of control variables. We also mark the minimal and maximal values of the time measurements.

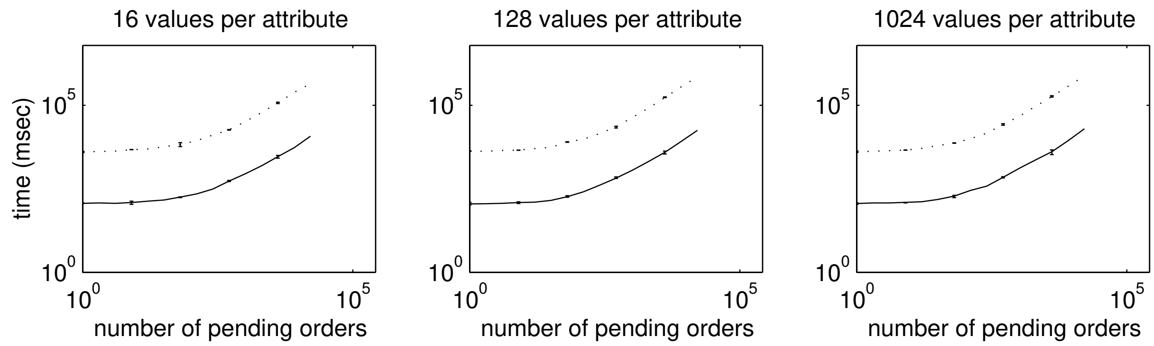


(a) Market with thirty attributes.

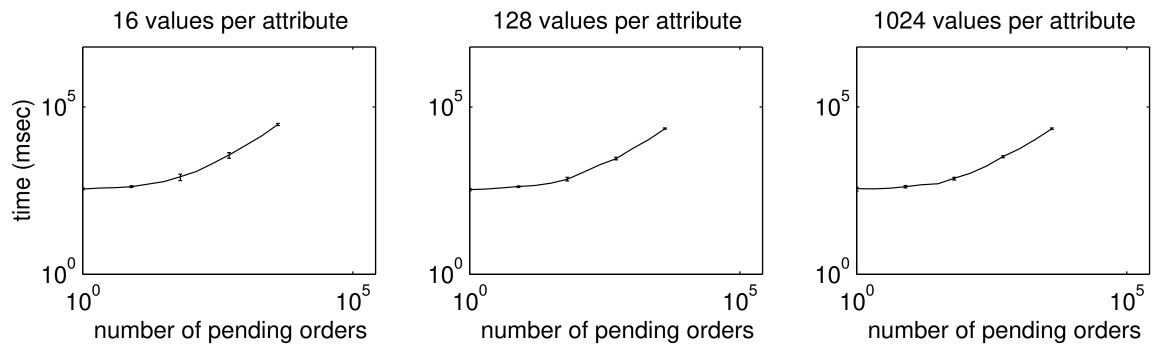


(b) Market with one hundred attributes.

Figure B.1: Time of processing the new orders, for *matching density of 0.0001.* We consider markets with two values per attribute (left), four values per attribute (middle), and eight values per attribute (right). We show the dependency of the processing time on the number of pending orders, for 256 new orders (solid lines) and 8,192 new orders (dotted lines). Both horizontal and vertical scales are logarithmic, and the vertical bars mark the minimal and maximal values of the time measurements.

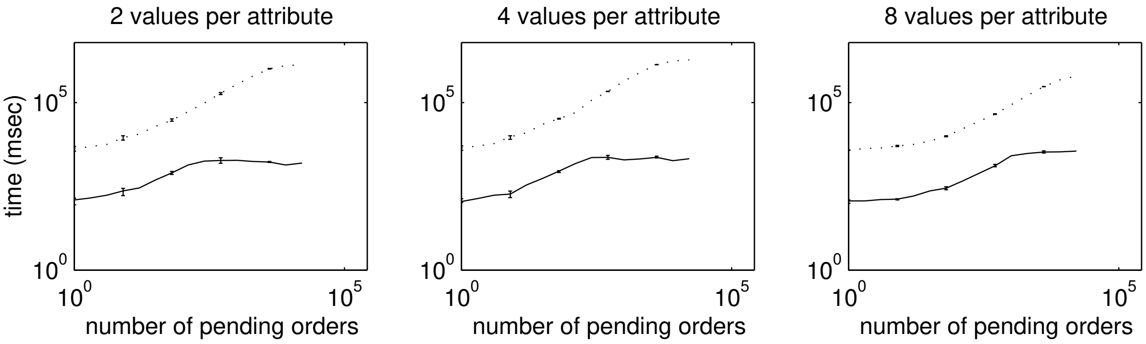


(a) Market with thirty attributes.

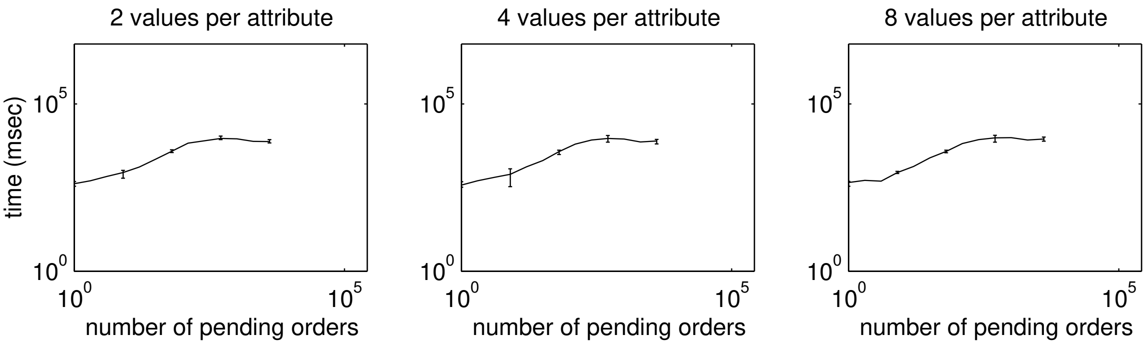


(b) Market with one hundred attributes.

Figure B.2: Time of processing the new orders, for *matching density of 0.0001.* We consider markets with 16 values per attribute (left), 128 values per attribute (middle), and 1,024 values per attribute (right); the legend is the same as in Figure B.1.

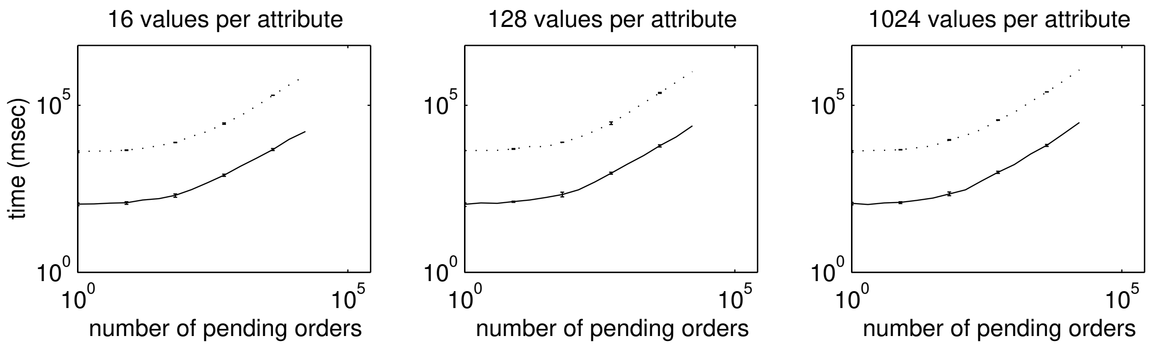


(a) Market with thirty attributes.

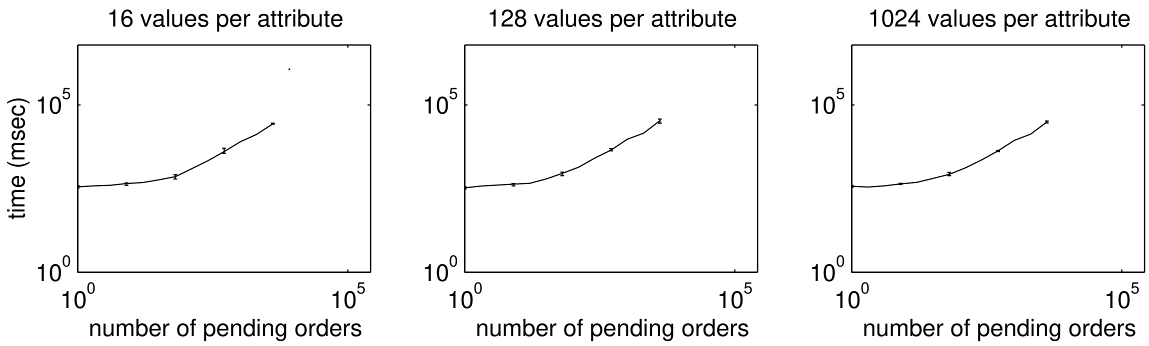


(b) Market with one hundred attributes.

Figure B.3: Time of processing the new orders, for *matching density of 0.001.* We consider markets with two values per attribute (left), four values per attribute (middle), and eight values per attribute (right); the legend is the same as in Figure B.1.

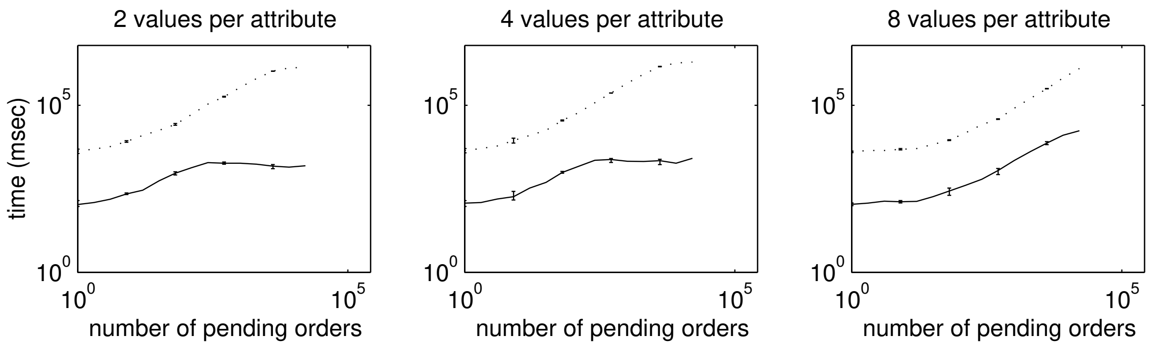


(a) Market with thirty attributes.

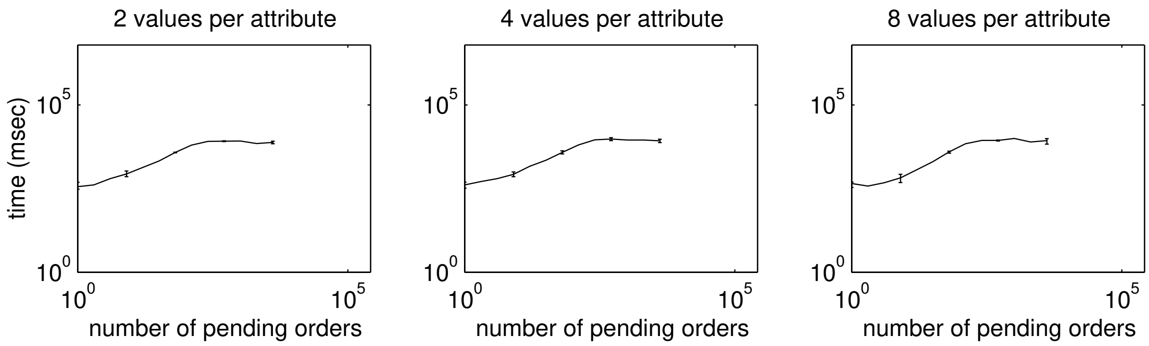


(b) Market with one hundred attributes.

Figure B.4: Time of processing the new orders, for *matching density of 0.001.* We consider markets with 16 values per attribute (left), 128 values per attribute (middle), and 1,024 values per attribute (right); the legend is the same as in Figure B.1.

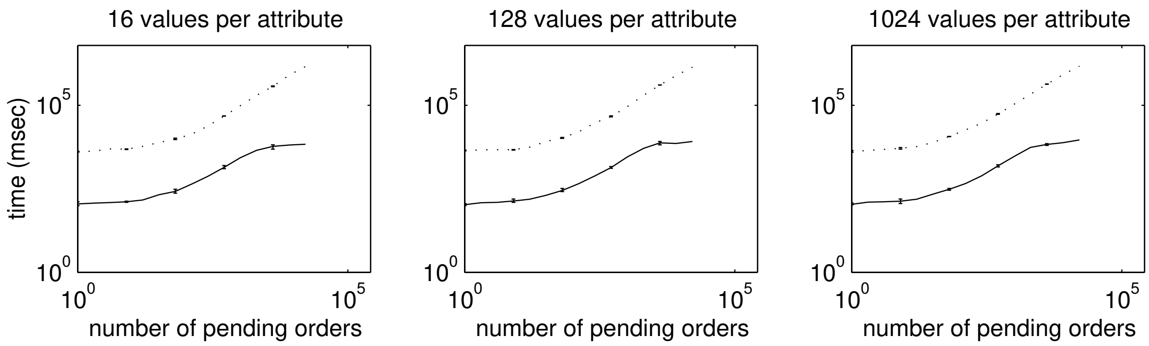


(a) Market with thirty attributes.

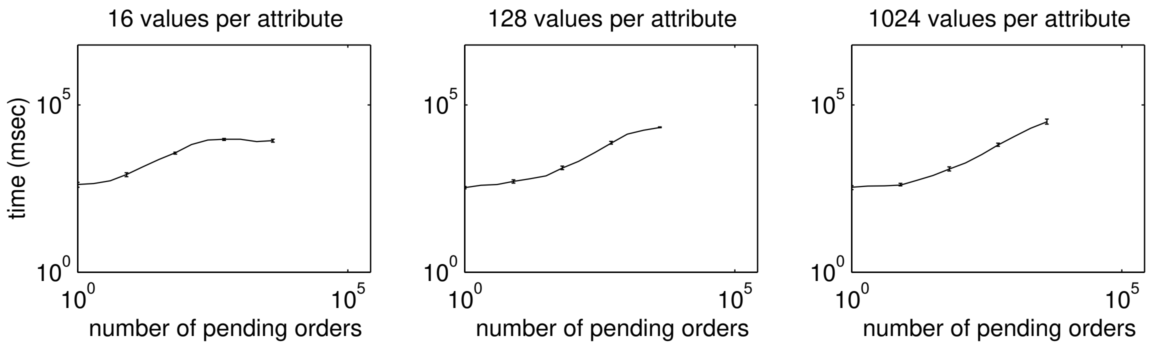


(b) Market with one hundred attributes.

Figure B.5: Time of processing the new orders, for *matching density of 0.01.* We consider markets with two values per attribute (left), four values per attribute (middle), and eight values per attribute (right); the legend is the same as in Figure B.1.

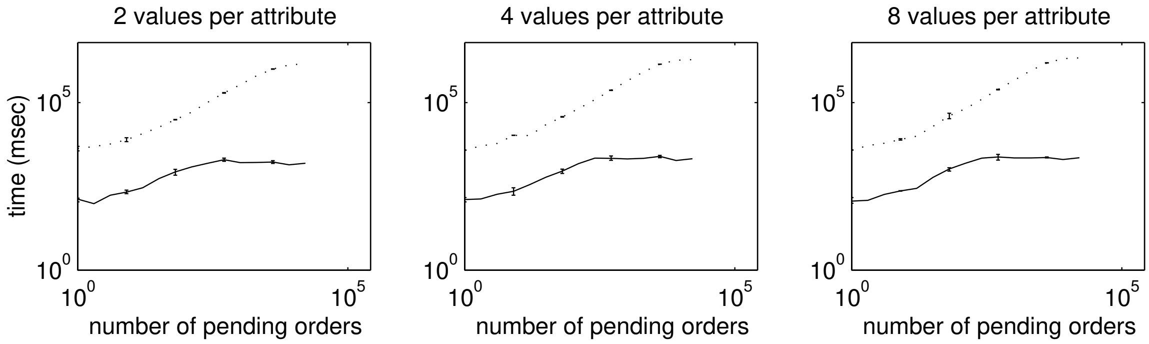


(a) Market with thirty attributes.

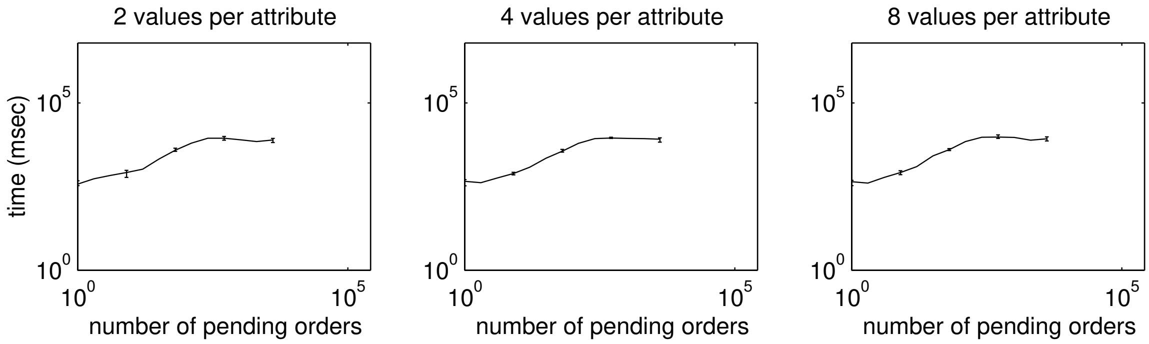


(b) Market with one hundred attributes.

Figure B.6: Time of processing the new orders, for *matching density of 0.01.* We consider markets with 16 values per attribute (left), 128 values per attribute (middle), and 1,024 values per attribute (right); the legend is the same as in Figure B.1.

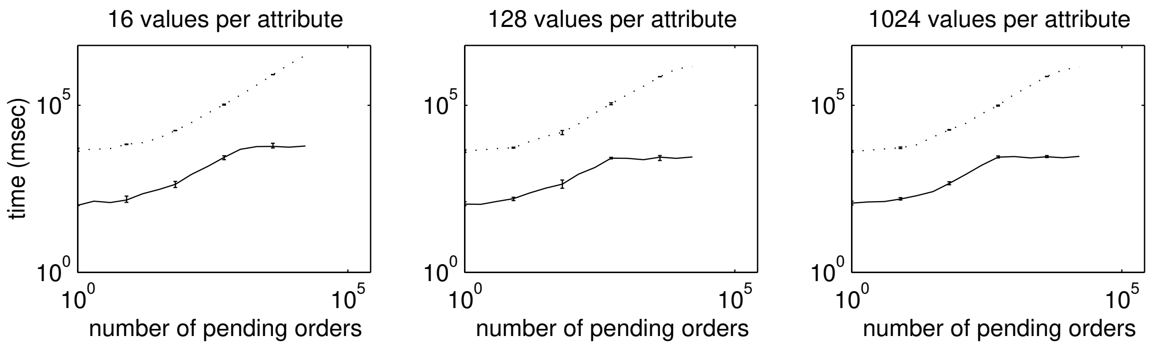


(a) Market with thirty attributes.

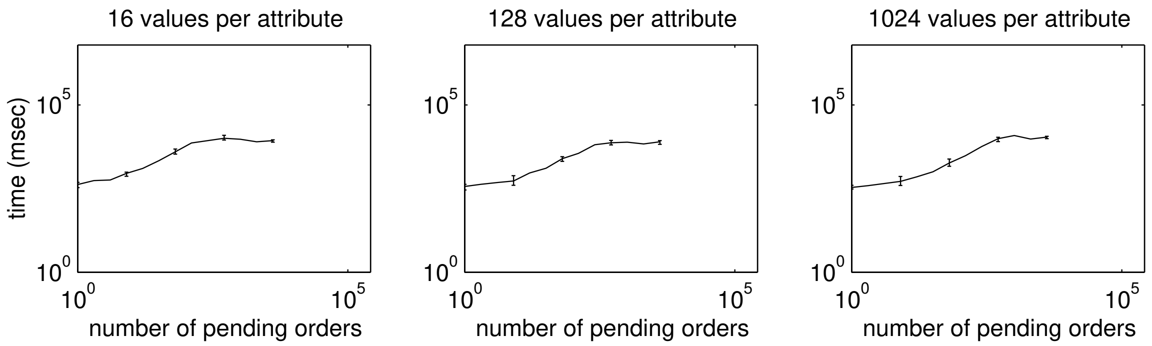


(b) Market with one hundred attributes.

Figure B.7: Time of processing the new orders, for *matching density of 0.1.* We consider markets with two values per attribute (left), four values per attribute (middle), and eight values per attribute (right); the legend is the same as in Figure B.1.

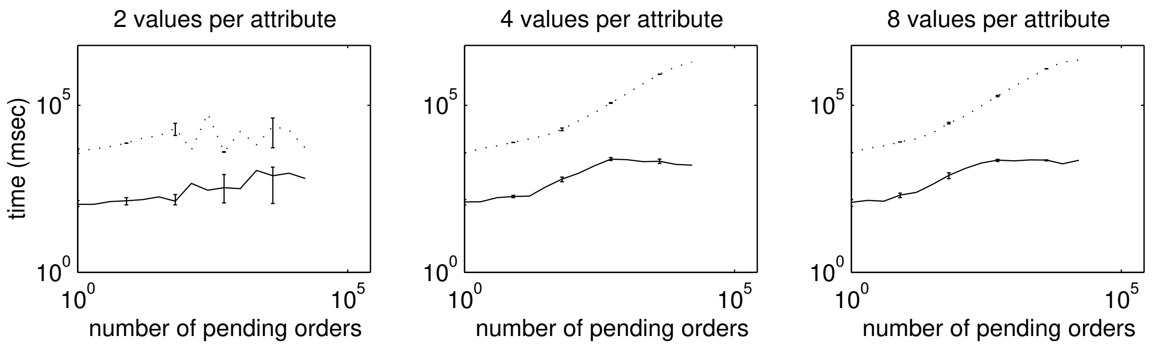


(a) Market with thirty attributes.

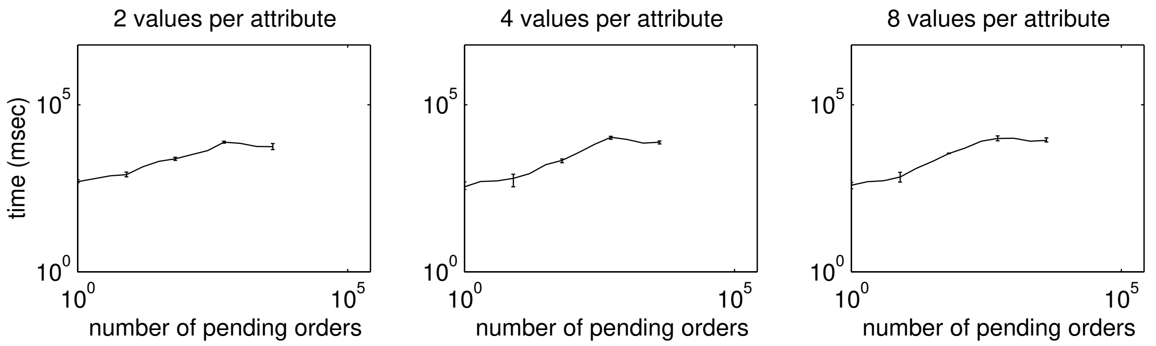


(b) Market with one hundred attributes.

Figure B.8: Time of processing the new orders, for *matching density of 0.1.* We consider markets with 16 values per attribute (left), 128 values per attribute (middle), and 1,024 values per attribute (right); the legend is the same as in Figure B.1.

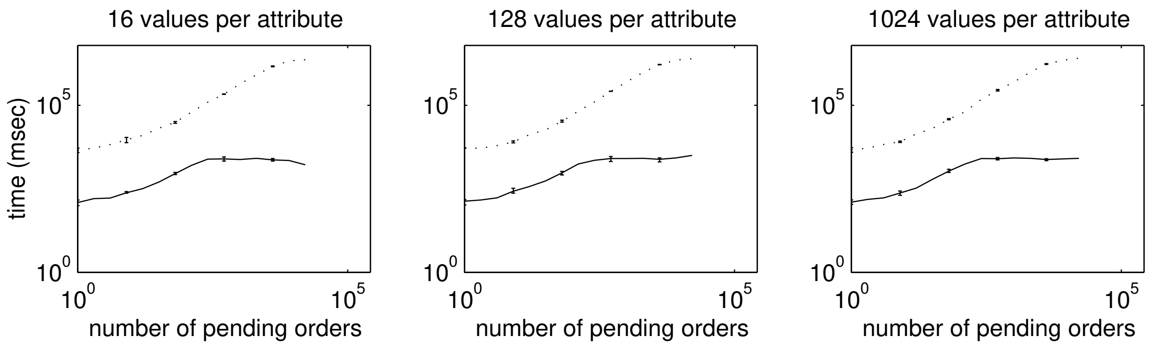


(a) Market with thirty attributes.

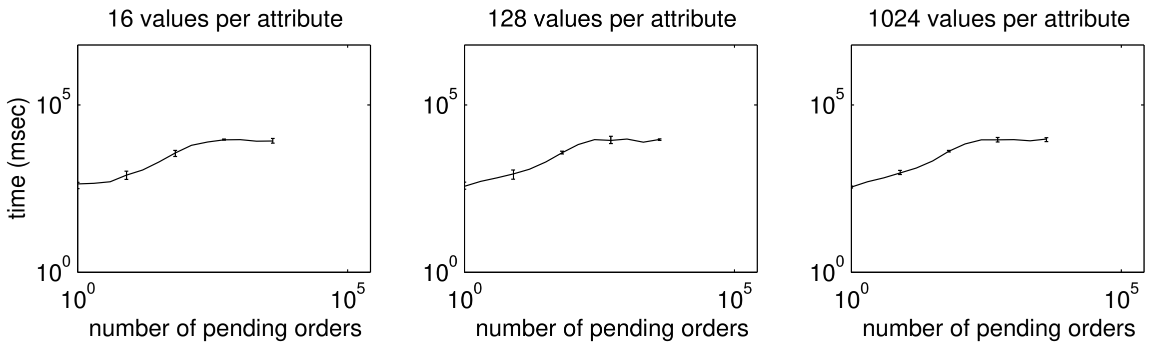


(b) Market with one hundred attributes.

Figure B.9: Time of processing the new orders, for *matching density of 1.* We consider markets with two values per attribute (left), four values per attribute (middle), and eight values per attribute (right); the legend is the same as in Figure B.1.



(a) Market with thirty attributes.

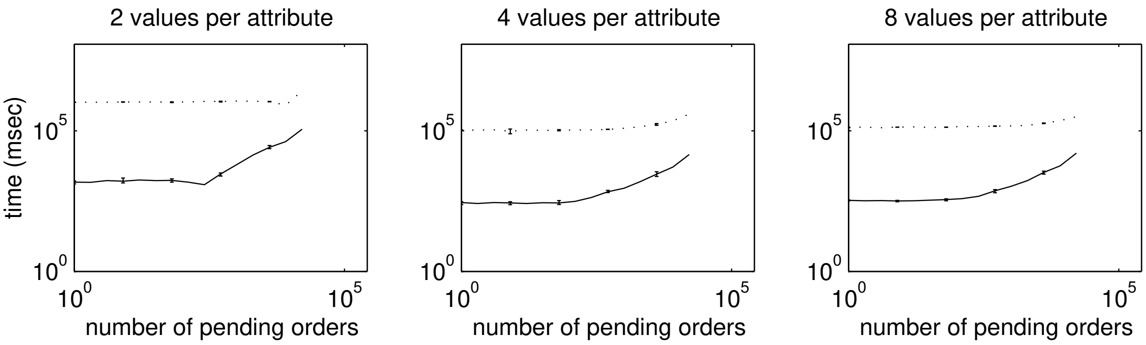


(b) Market with one hundred attributes.

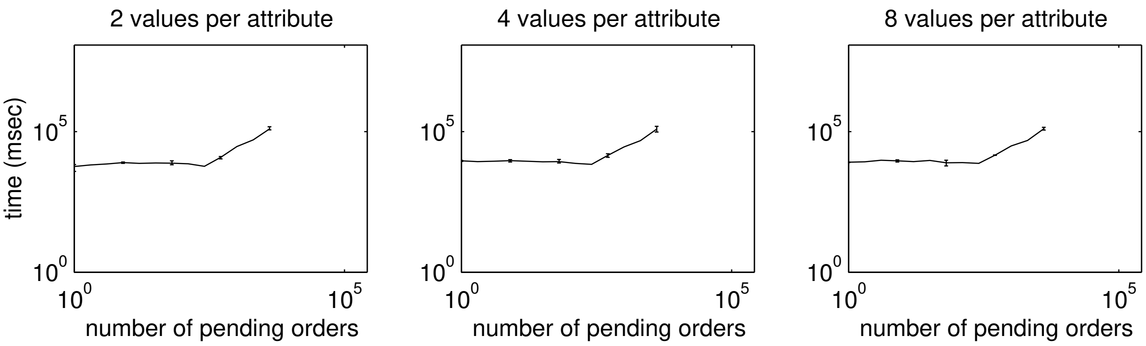
Figure B.10: Time of processing the new orders, for *matching density of 1.* We consider markets with 16 values per attribute (left), 128 values per attribute (middle), and 1,024 values per attribute (right); the legend is the same as in Figure B.1.

B.2 Matching Time

We give the mean time of matching all pending buy orders, in the second half of the system's top-level loop (see Figure 3.2a), along with the minimal and maximal values of the time measurements.

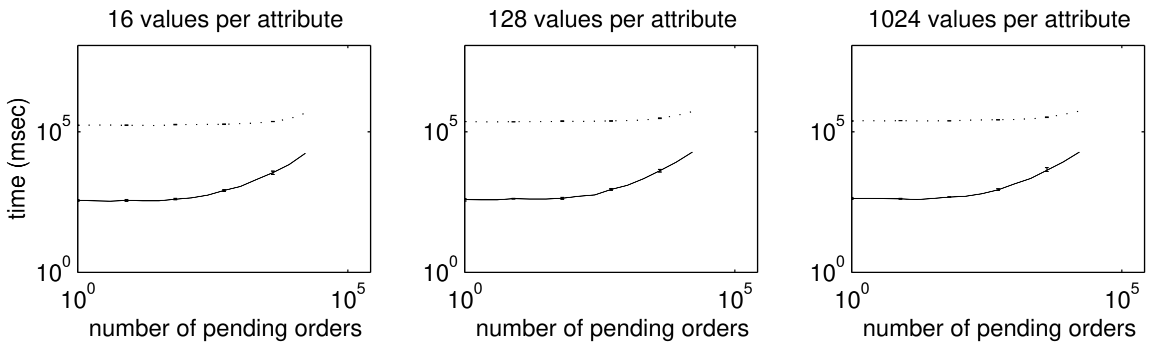


(a) Market with thirty attributes.

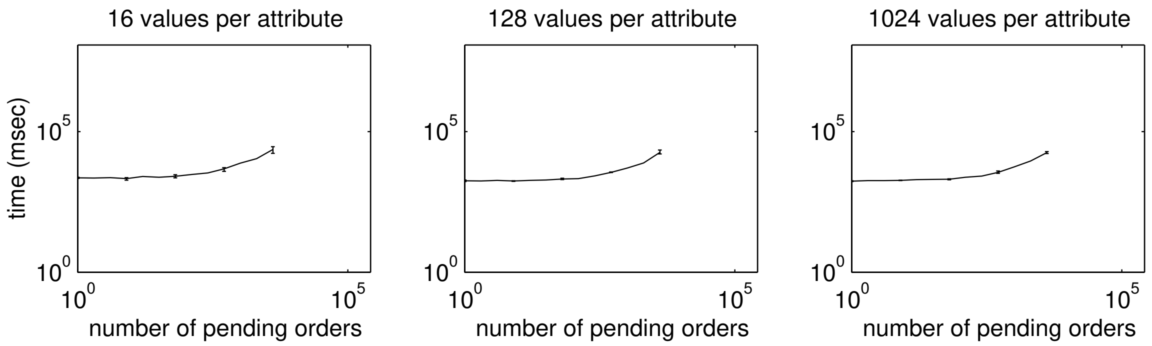


(b) Market with one hundred attributes.

Figure B.11: Time of matching the pending orders, for *matching density of 0.0001.* We consider markets with two values per attribute (left), four values per attribute (middle), and eight values per attribute (right). We show the dependency of the matching time on the number of pending orders, for 256 new orders (solid lines) and 8,192 new orders (dotted lines). Both horizontal and vertical scales are logarithmic, and the vertical bars mark the minimal and maximal values of the time measurements.

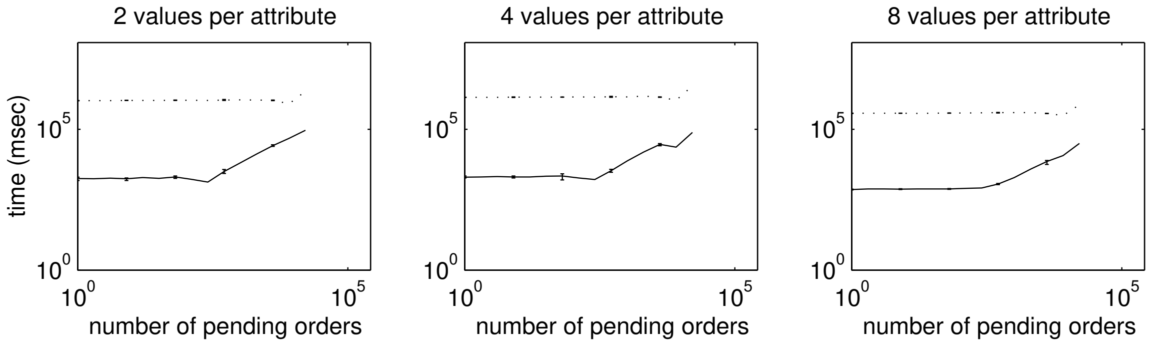


(a) Market with thirty attributes.

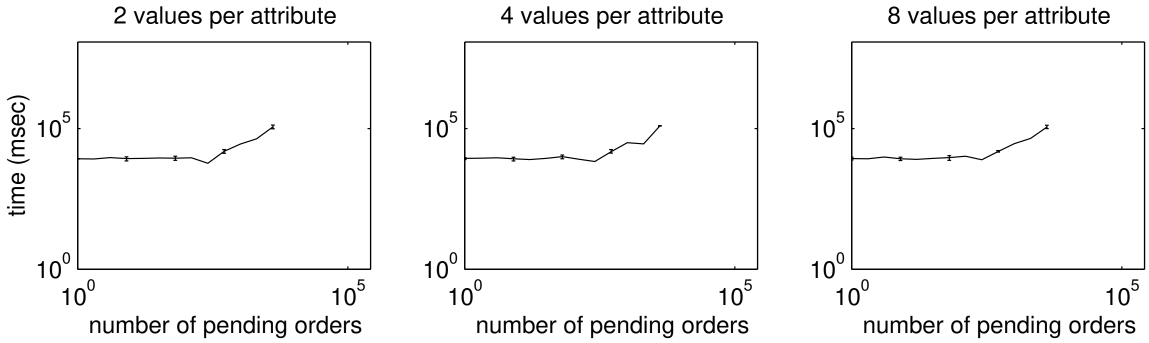


(b) Market with one hundred attributes.

Figure B.12: Time of matching the pending orders, for *matching density of 0.0001.* We consider markets with 16 values per attribute (left), 128 values per attribute (middle), and 1,024 values per attribute (right); the legend is the same as in Figure B.11.

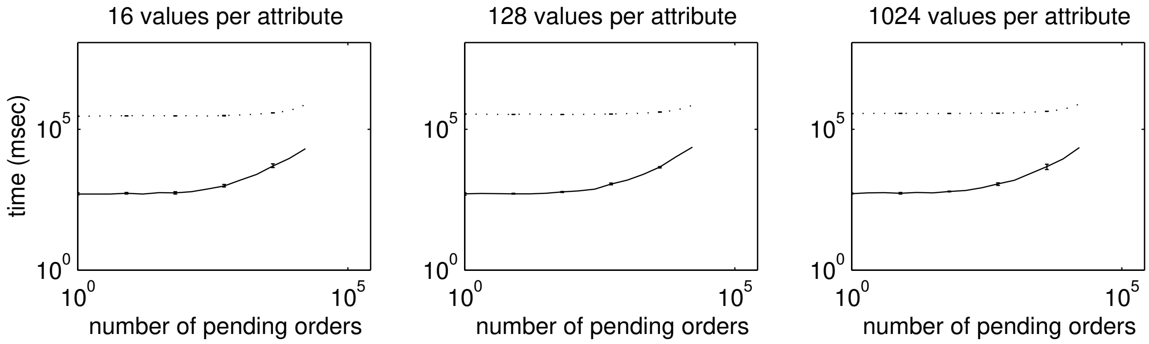


(a) Market with thirty attributes.

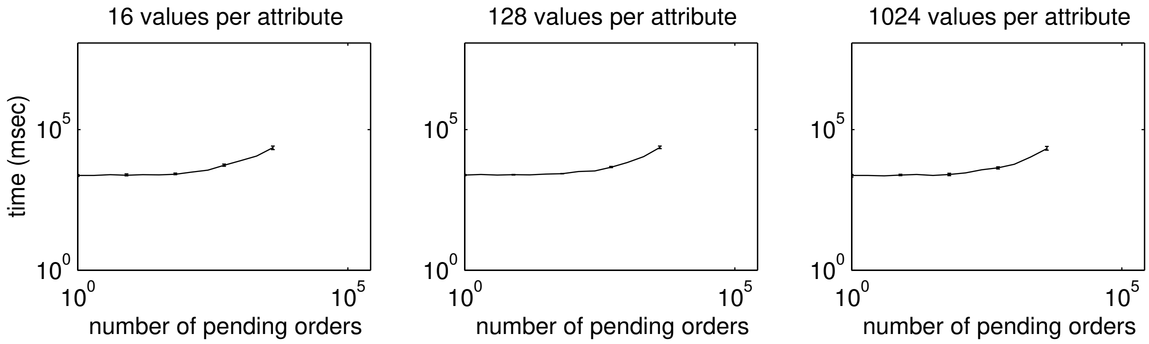


(b) Market with one hundred attributes.

Figure B.13: Time of matching the pending orders, for *matching density of 0.001.* We consider markets with two values per attribute (left), four values per attribute (middle), and eight values per attribute (right); the legend is the same as in Figure B.11.

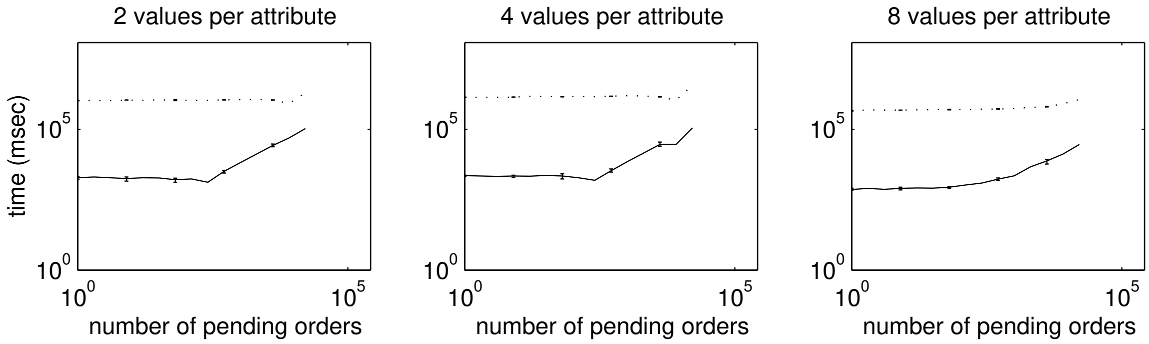


(a) Market with thirty attributes.

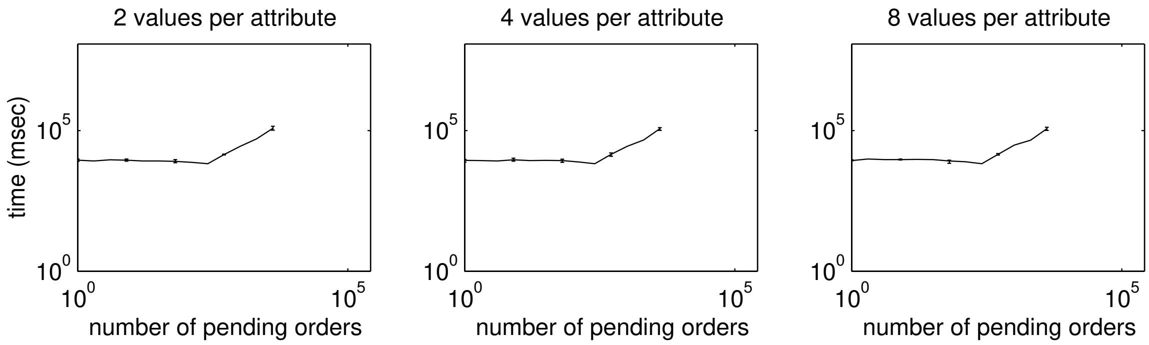


(b) Market with one hundred attributes.

Figure B.14: Time of matching the pending orders, for *matching density of 0.001.* We consider markets with 16 values per attribute (left), 128 values per attribute (middle), and 1,024 values per attribute (right); the legend is the same as in Figure B.11.

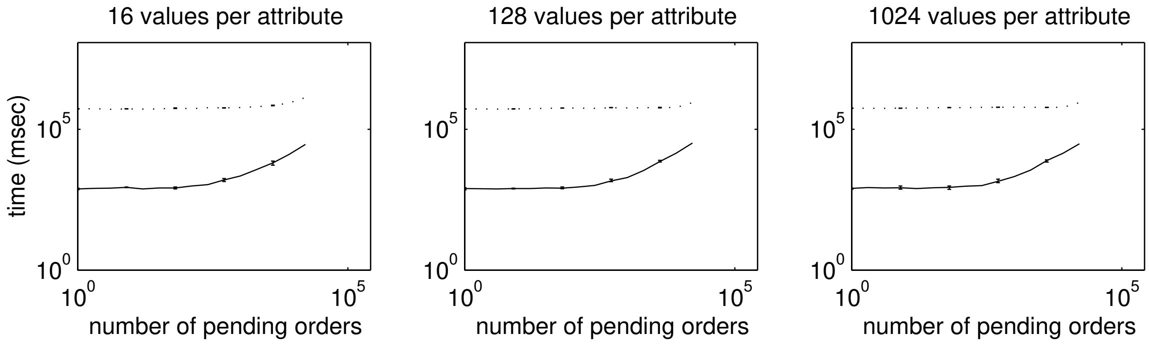


(a) Market with thirty attributes.

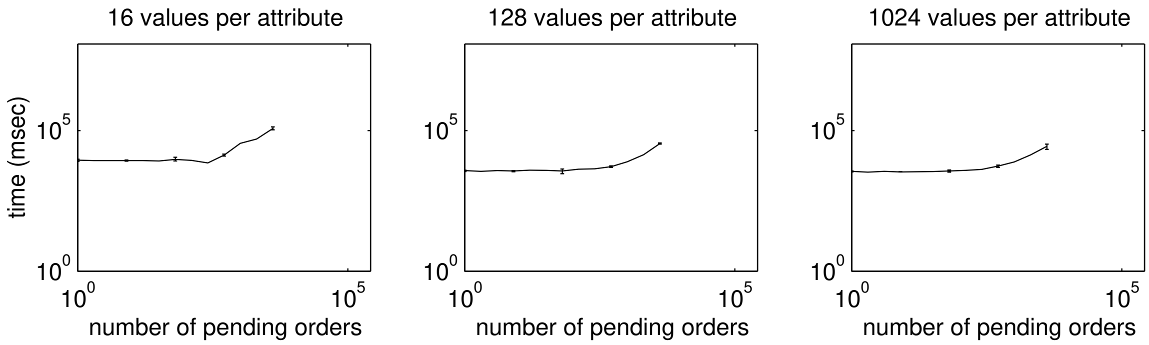


(b) Market with one hundred attributes.

Figure B.15: Time of matching the pending orders, for *matching density of 0.01.* We consider markets with two values per attribute (left), four values per attribute (middle), and eight values per attribute (right); the legend is the same as in Figure B.11.

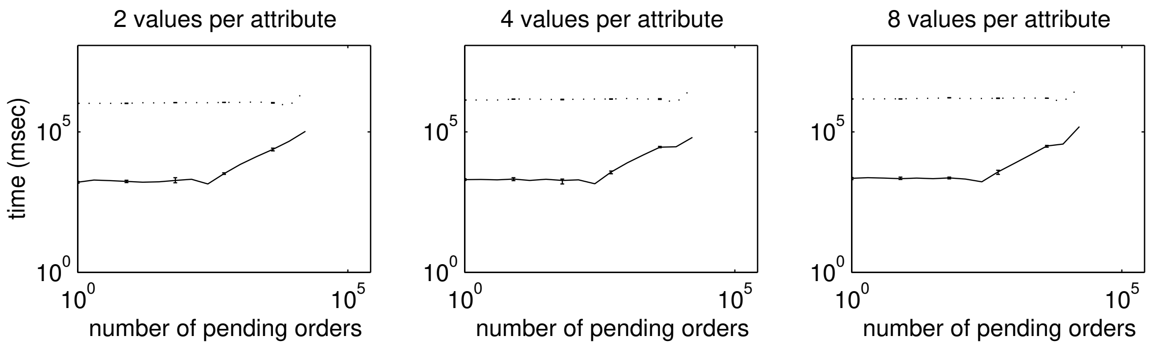


(a) Market with thirty attributes.

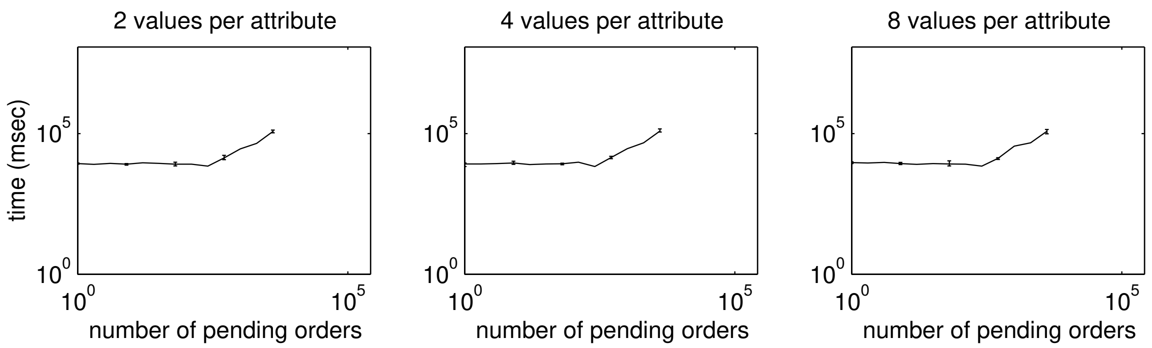


(b) Market with one hundred attributes.

Figure B.16: Time of matching the pending orders, for *matching density of 0.01.* We consider markets with 16 values per attribute (left), 128 values per attribute (middle), and 1,024 values per attribute (right); the legend is the same as in Figure B.11.

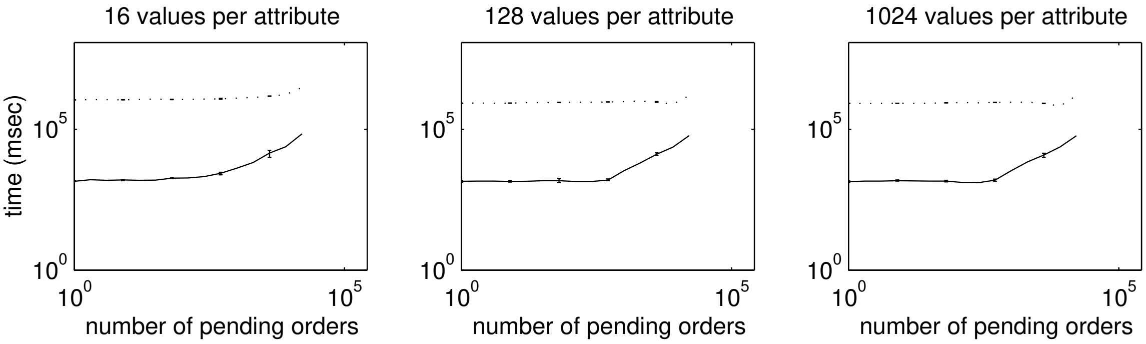


(a) Market with thirty attributes.

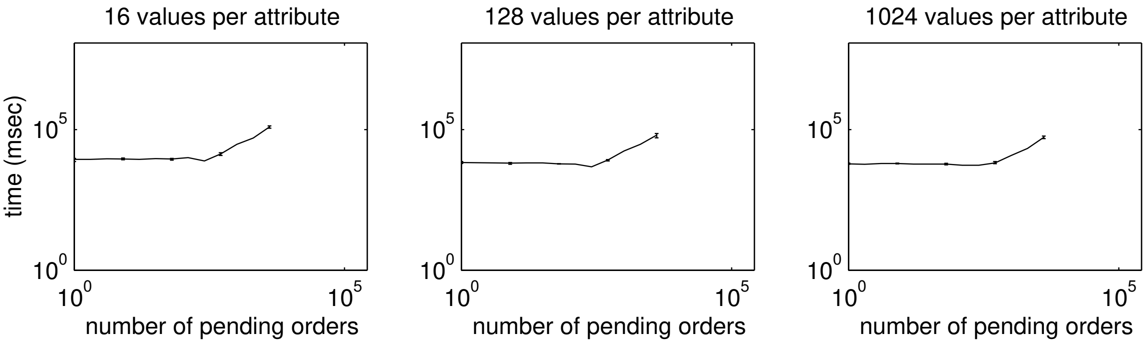


(b) Market with one hundred attributes.

Figure B.17: Time of matching the pending orders, for *matching density of 0.1.* We consider markets with two values per attribute (left), four values per attribute (middle), and eight values per attribute (right); the legend is the same as in Figure B.11.

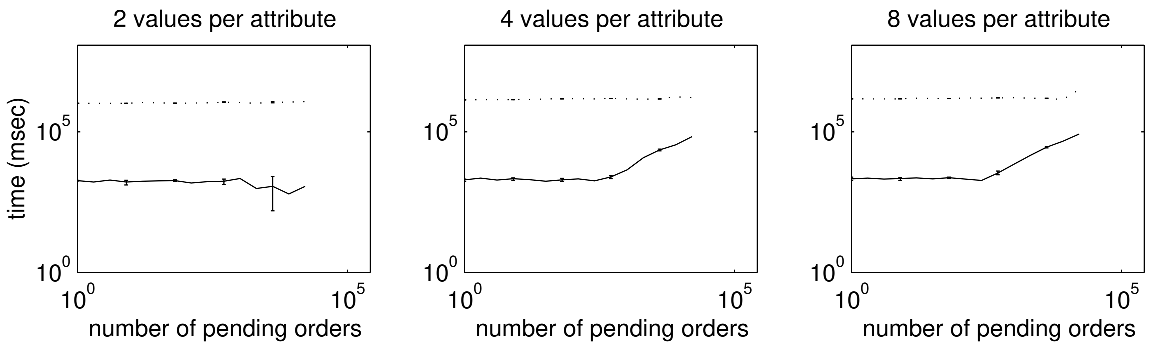


(a) Market with thirty attributes.

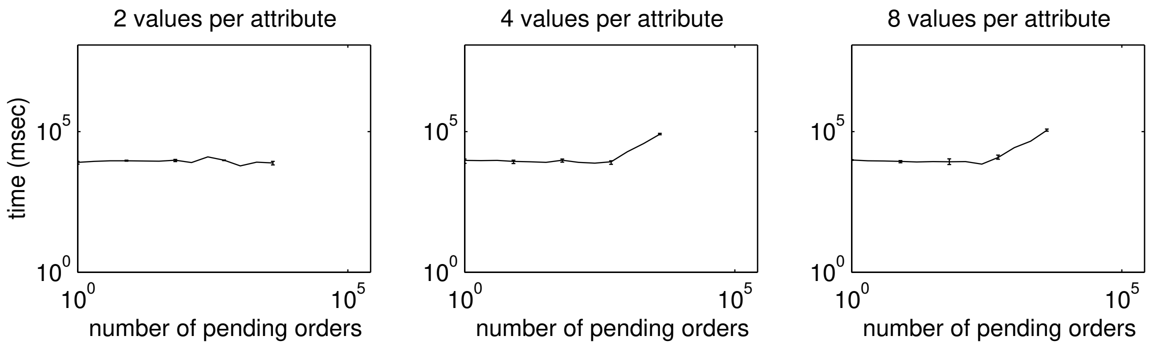


(b) Market with one hundred attributes.

Figure B.18: Time of matching the pending orders, for *matching density of 0.1.* We consider markets with 16 values per attribute (left), 128 values per attribute (middle), and 1,024 values per attribute (right); the legend is the same as in Figure B.11.

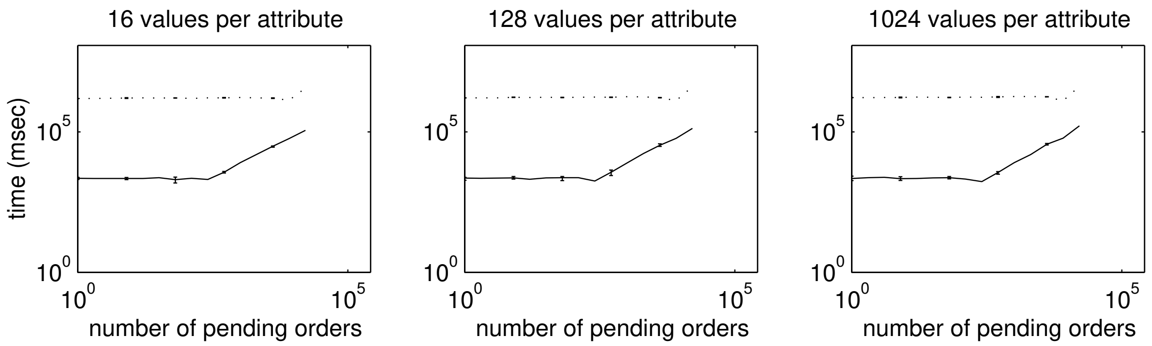


(a) Market with thirty attributes.

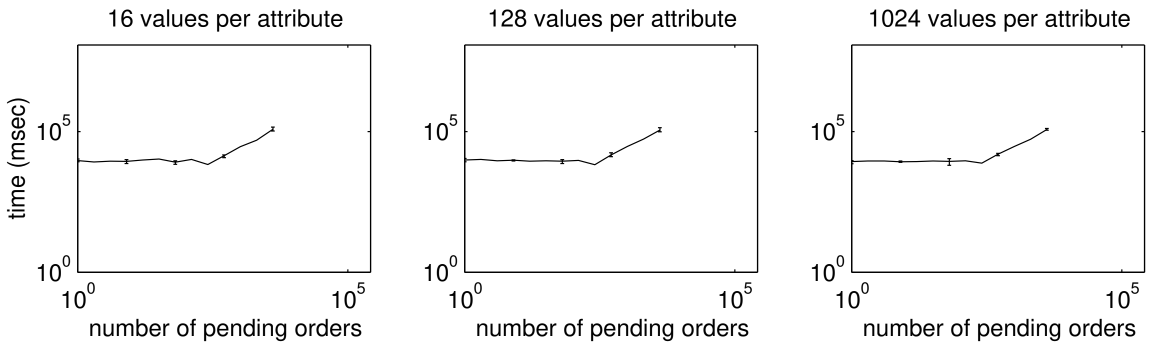


(b) Market with one hundred attributes.

Figure B.19: Time of matching the pending orders, for *matching density of 1.* We consider markets with two values per attribute (left), four values per attribute (middle), and eight values per attribute (right); the legend is the same as in Figure B.11.



(a) Market with thirty attributes.

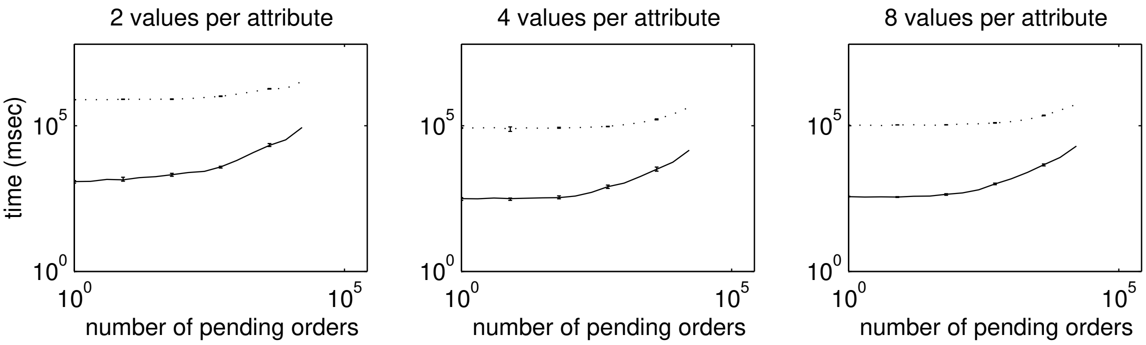


(b) Market with one hundred attributes.

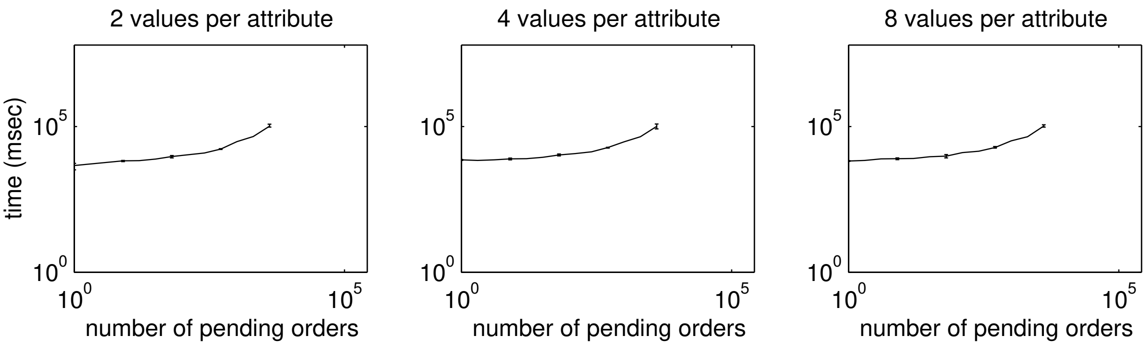
Figure B.20: Time of matching the pending orders, for *matching density of 1.* We consider markets with 16 values per attribute (left), 128 values per attribute (middle), and 1,024 values per attribute (right); the legend is the same as in Figure B.11.

B.3 Response Time

We plot the average time between placing a new order and receiving a response from the system. Recall that the response may not include a fill for the order; if the matcher does not find an immediate fill, it responds with a confirmation of the new order.

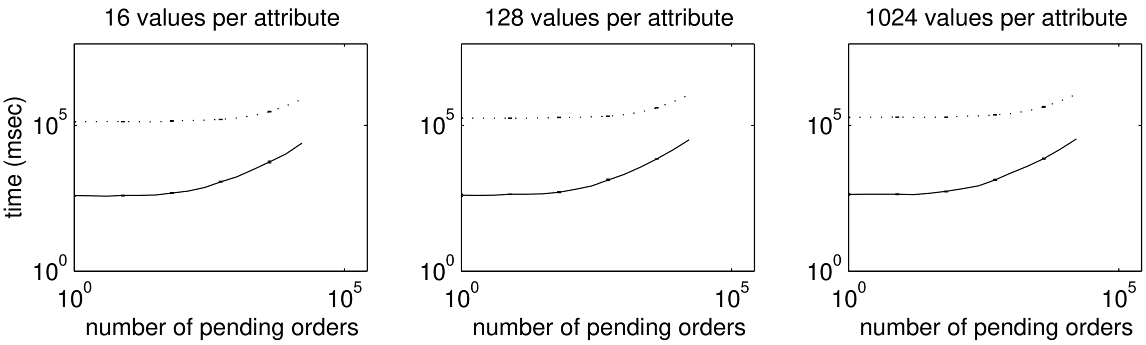


(a) Market with thirty attributes.

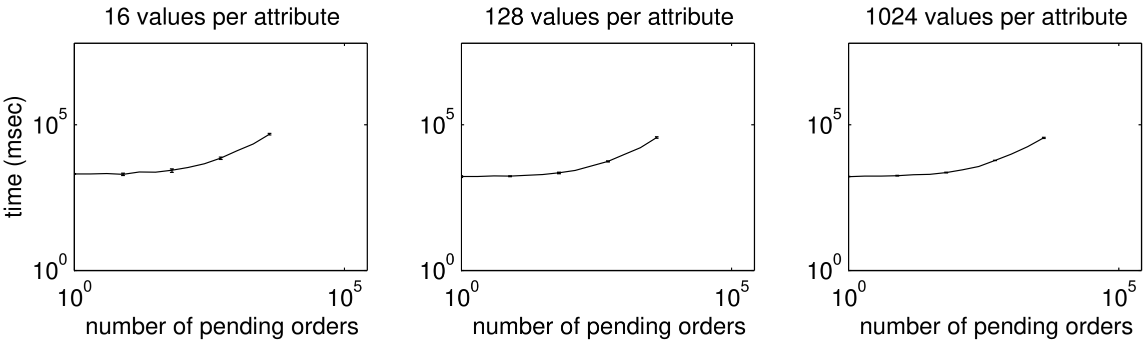


(b) Market with one hundred attributes.

Figure B.21: Time between placing a new order and getting a response, for *matching density of 0.0001.* We consider markets with two values per attribute (left), four values per attribute (middle), and eight values per attribute (right). We show the dependency of the response time on the number of pending orders, for 256 new orders (solid lines) and 8,192 new orders (dotted lines). Both horizontal and vertical scales are logarithmic, and the vertical bars mark the minimal and maximal values of the time measurements.

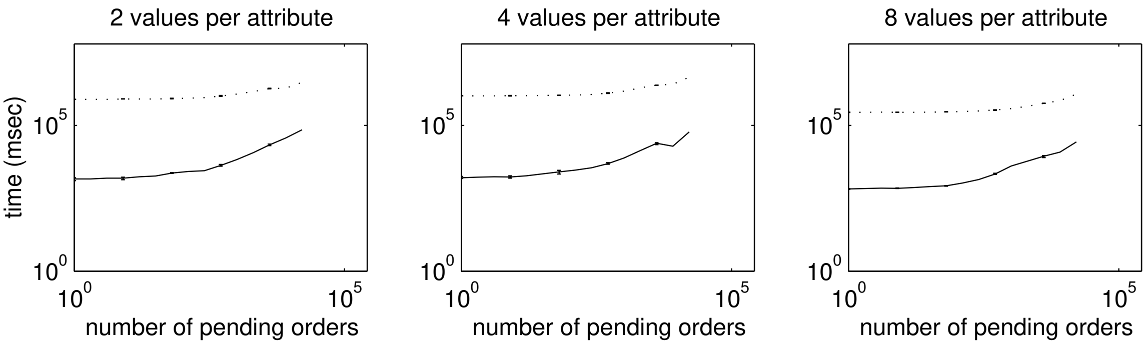


(a) Market with thirty attributes.

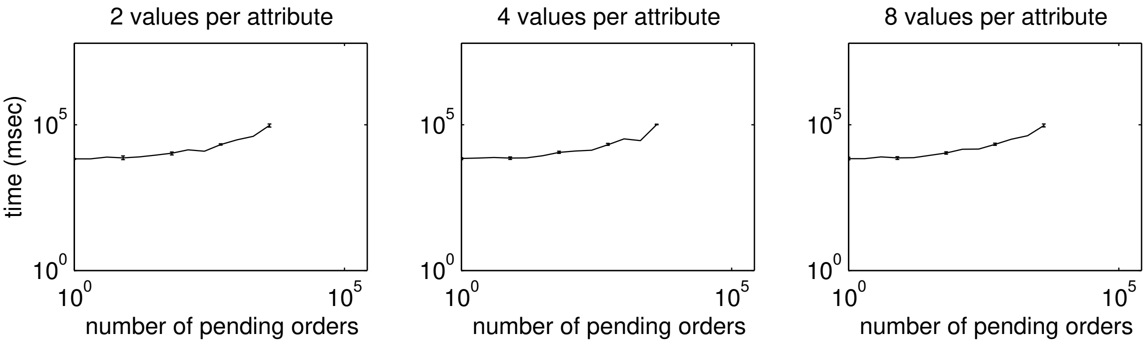


(b) Market with one hundred attributes.

Figure B.22: Time between placing a new order and getting a response, for *matching density of 0.0001.* We consider markets with 16 values per attribute (left), 128 values per attribute (middle), and 1,024 values per attribute (right); the legend is the same as in Figure B.21.

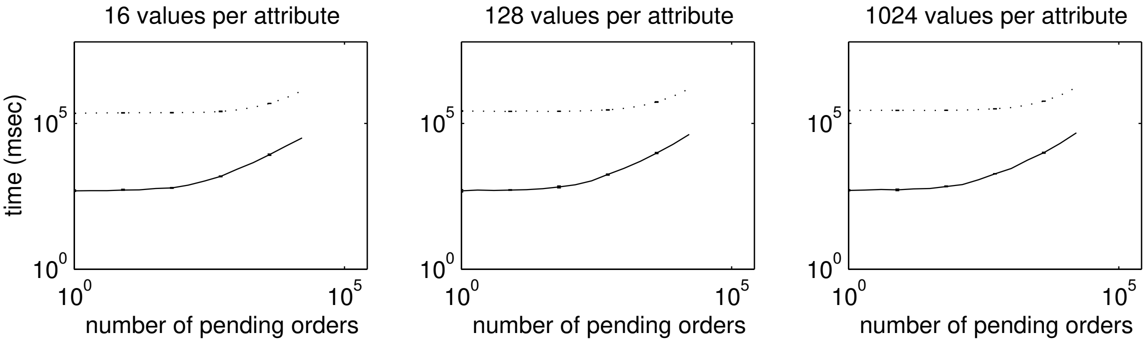


(a) Market with thirty attributes.

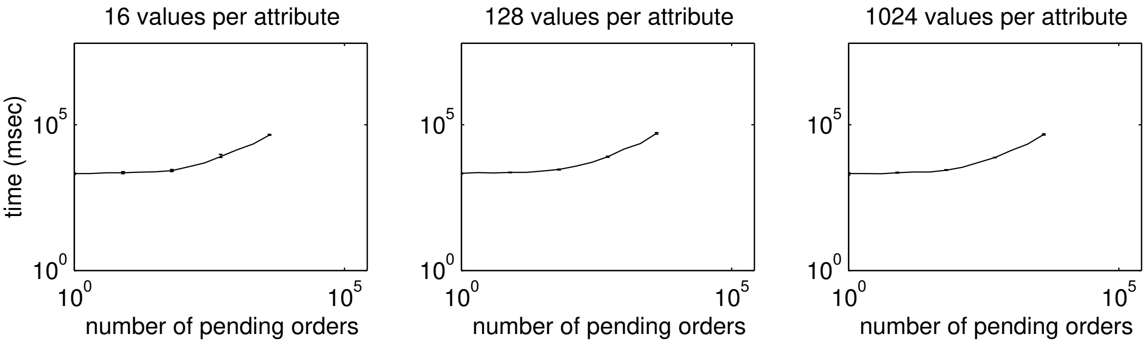


(b) Market with one hundred attributes.

Figure B.23: Time between placing a new order and getting a response, for *matching density of 0.001.* We consider markets with two values per attribute (left), four values per attribute (middle), and eight values per attribute (right); the legend is the same as in Figure B.21.

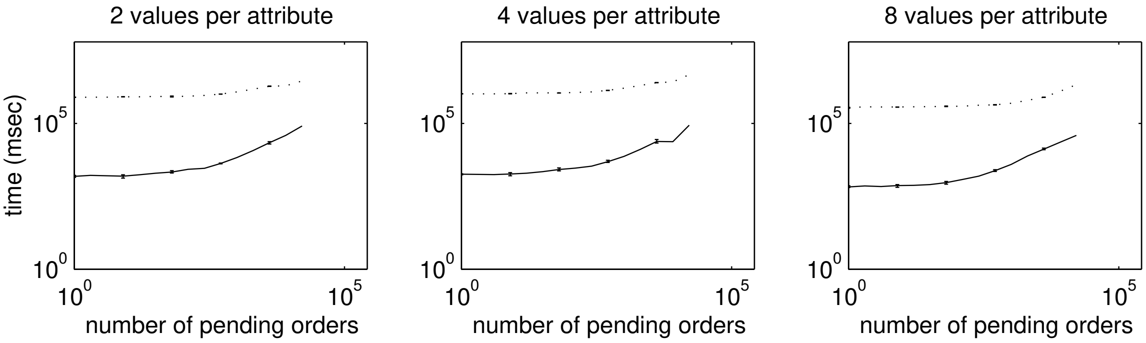


(a) Market with thirty attributes.

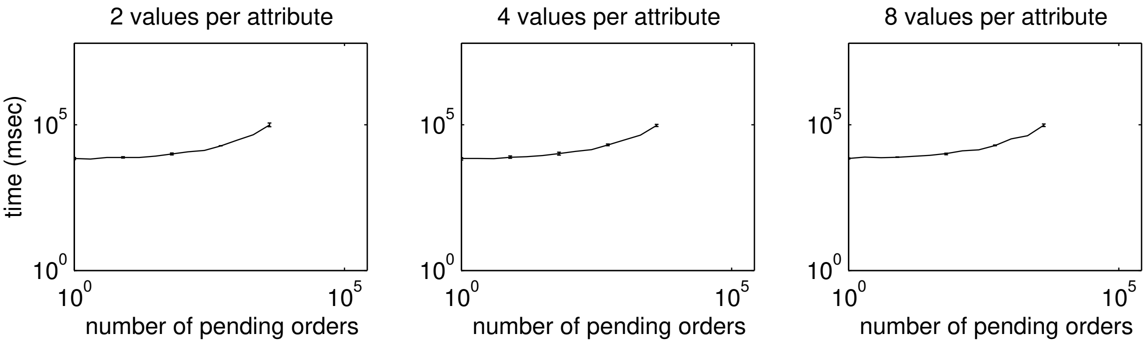


(b) Market with one hundred attributes.

Figure B.24: Time between placing a new order and getting a response, for *matching density of 0.001.* We consider markets with 16 values per attribute (left), 128 values per attribute (middle), and 1,024 values per attribute (right); the legend is the same as in Figure B.21.

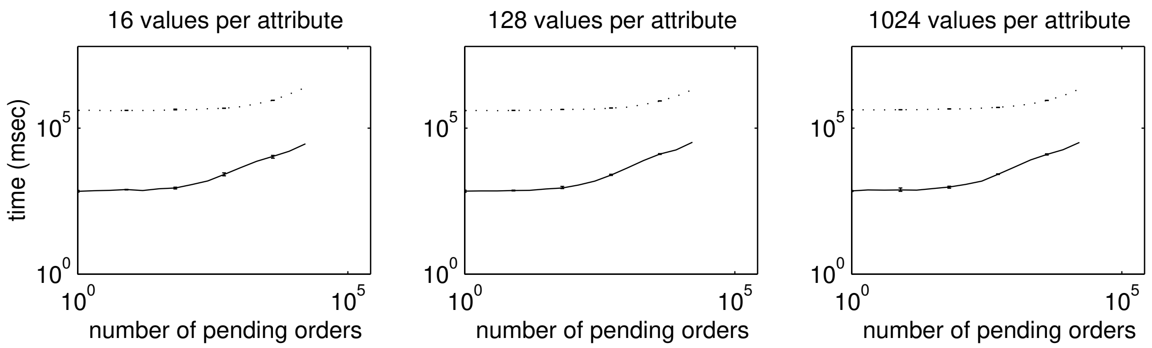


(a) Market with thirty attributes.

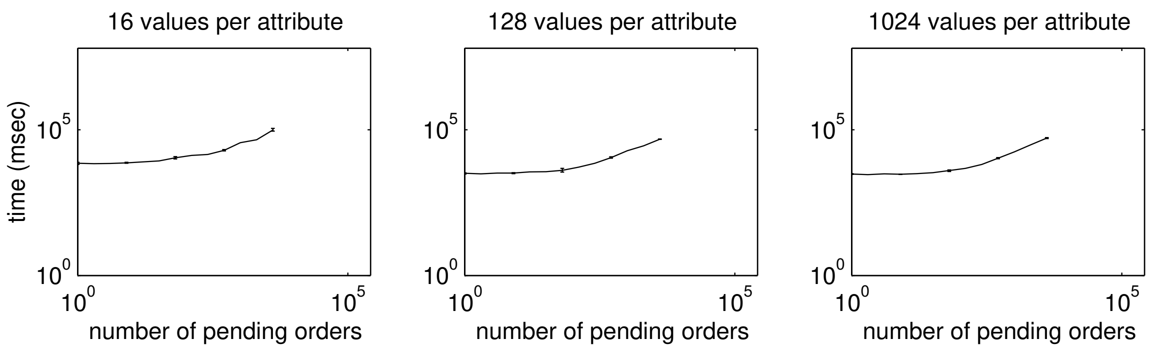


(b) Market with one hundred attributes.

Figure B.25: Time between placing a new order and getting a response, for *matching density of 0.01.* We consider markets with two values per attribute (left), four values per attribute (middle), and eight values per attribute (right); the legend is the same as in Figure B.21.

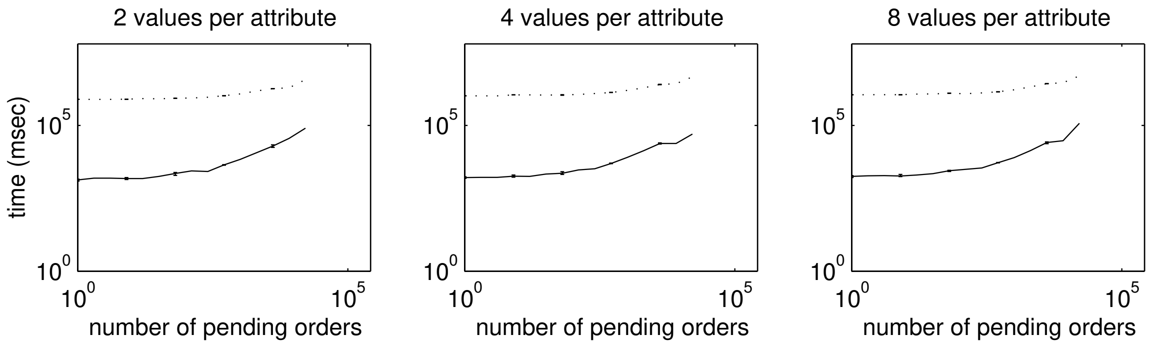


(a) Market with thirty attributes.

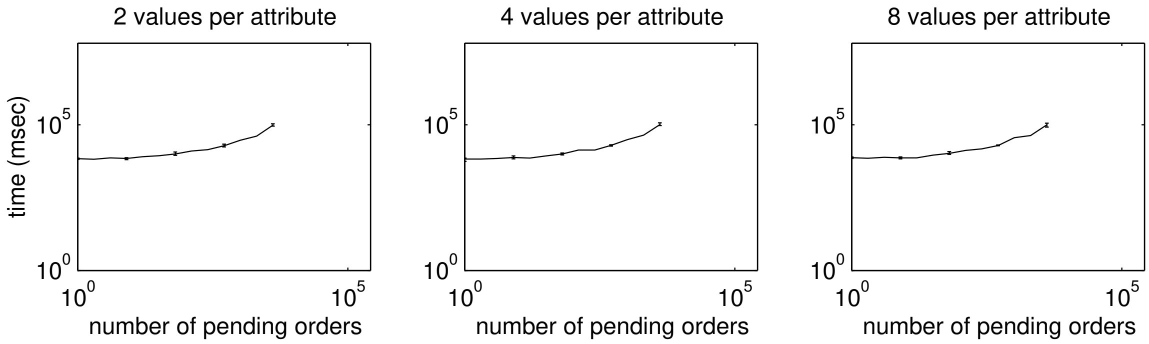


(b) Market with one hundred attributes.

Figure B.26: Time between placing a new order and getting a response, for *matching density of 0.01.* We consider markets with 16 values per attribute (left), 128 values per attribute (middle), and 1,024 values per attribute (right); the legend is the same as in Figure B.21.

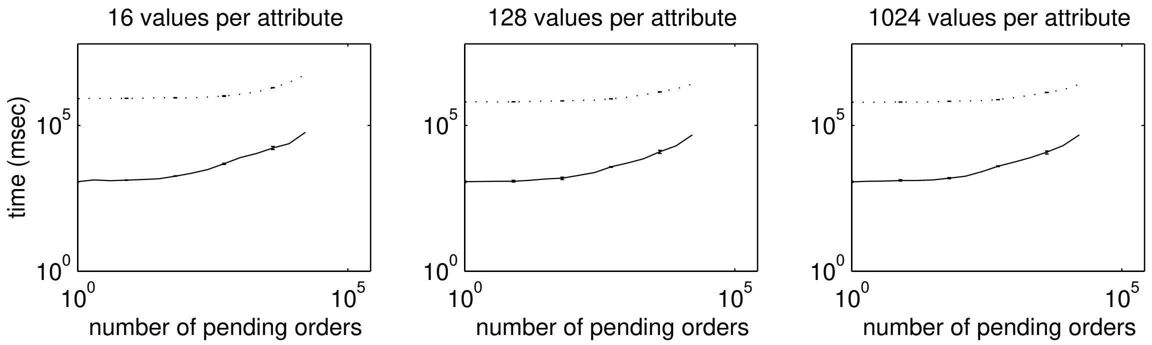


(a) Market with thirty attributes.

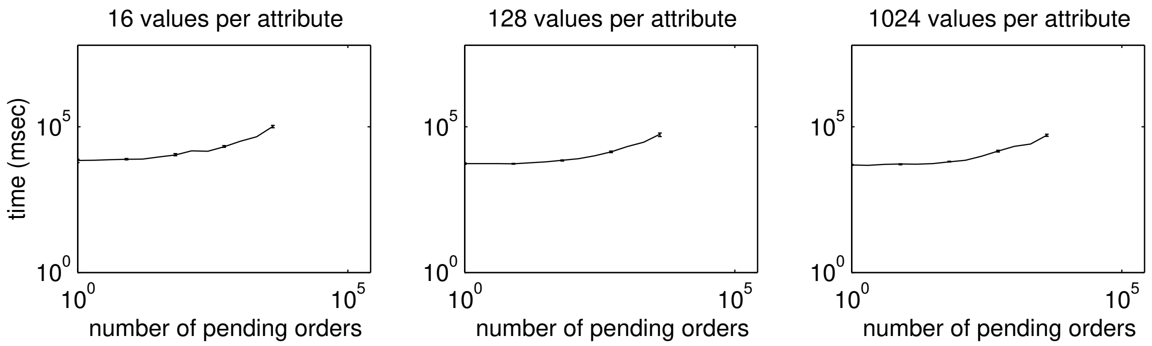


(b) Market with one hundred attributes.

Figure B.27: Time between placing a new order and getting a response, for *matching density of 0.1.* We consider markets with two values per attribute (left), four values per attribute (middle), and eight values per attribute (right); the legend is the same as in Figure B.21.

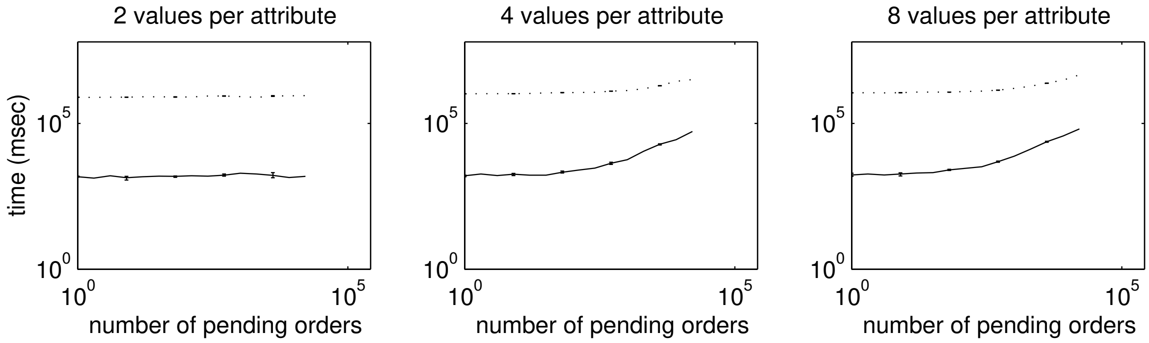


(a) Market with thirty attributes.

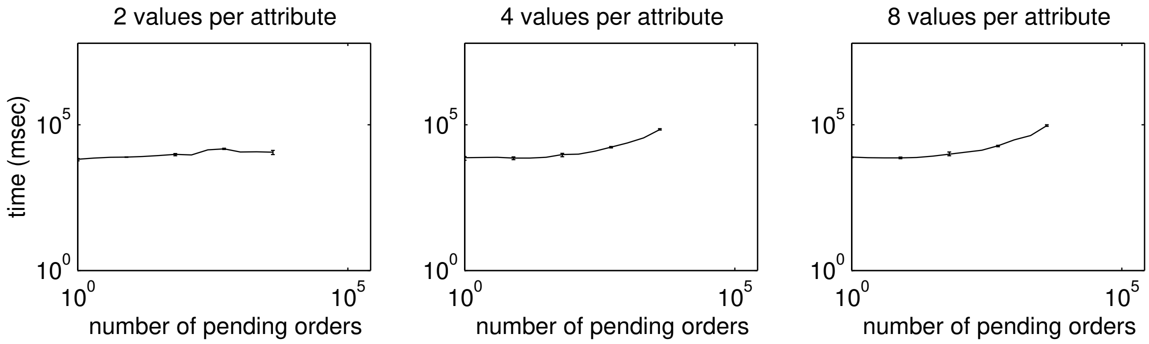


(b) Market with one hundred attributes.

Figure B.28: Time between placing a new order and getting a response, for *matching density of 0.1.* We consider markets with 16 values per attribute (left), 128 values per attribute (middle), and 1,024 values per attribute (right); the legend is the same as in Figure B.21.

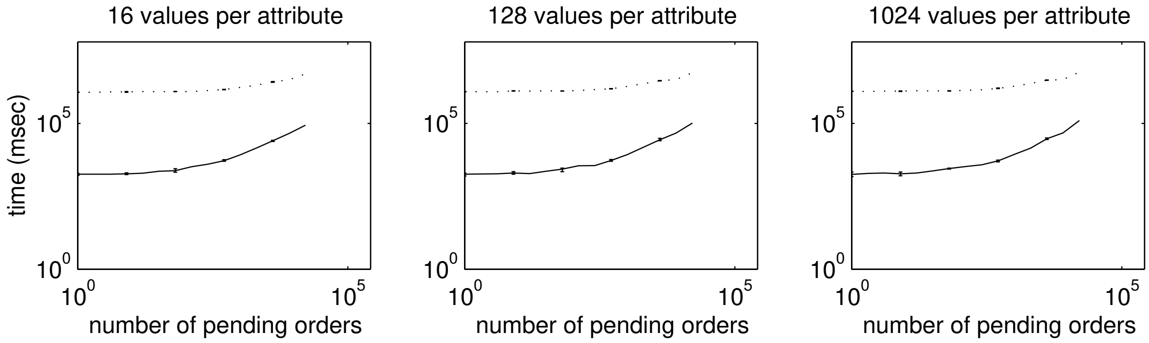


(a) Market with thirty attributes.

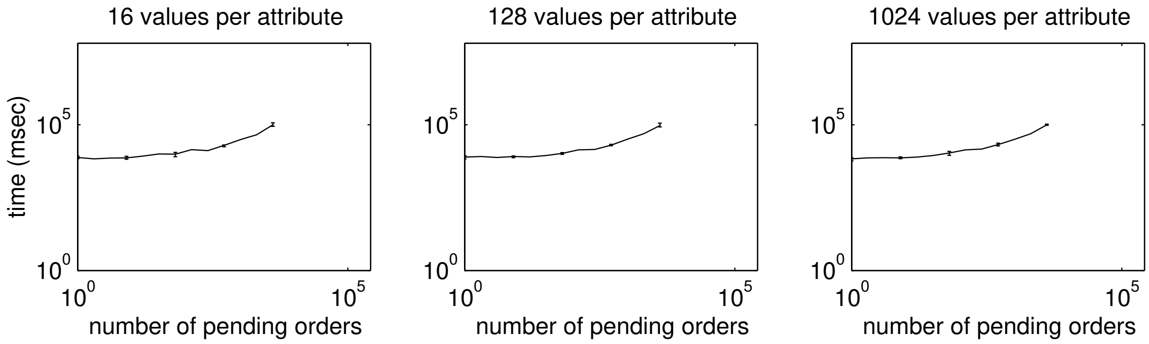


(b) Market with one hundred attributes.

Figure B.29: Time between placing a new order and getting a response, for *matching density of 1.* We consider markets with two values per attribute (left), four values per attribute (middle), and eight values per attribute (right); the legend is the same as in Figure B.21



(a) Market with thirty attributes.

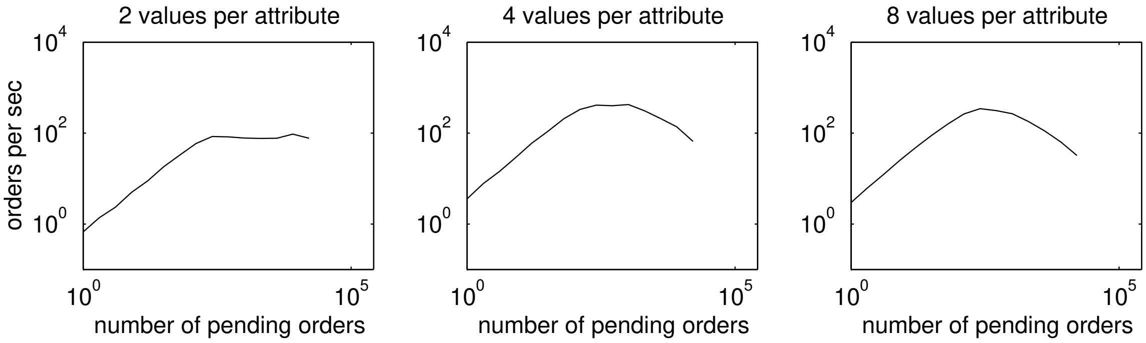


(b) Market with one hundred attributes.

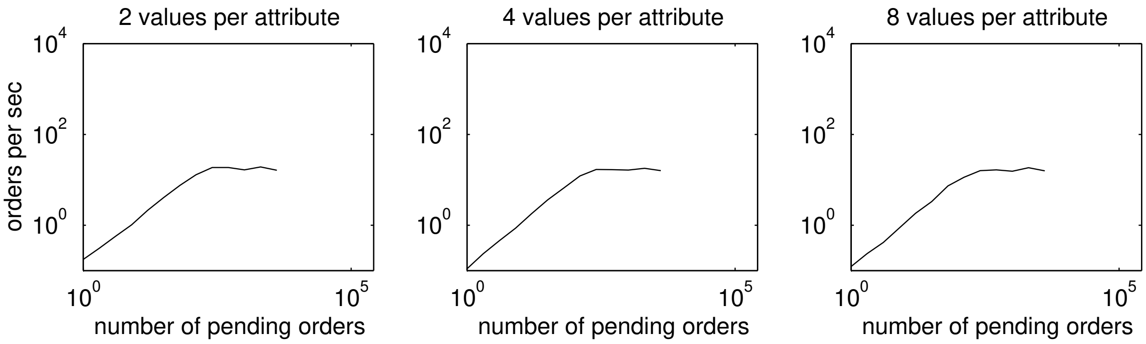
Figure B.30: Time between placing a new order and getting a response, for *matching density of 1.* We consider markets with 16 values per attribute (left), 128 values per attribute (middle), and 1,024 values per attribute (right); the legend is the same as in Figure B.21.

B.4 Maximal Throughput

We show the limit on the number of new orders per second. If the matcher gets more orders, it has to reject some of them, to prevent overflow of the message queue.

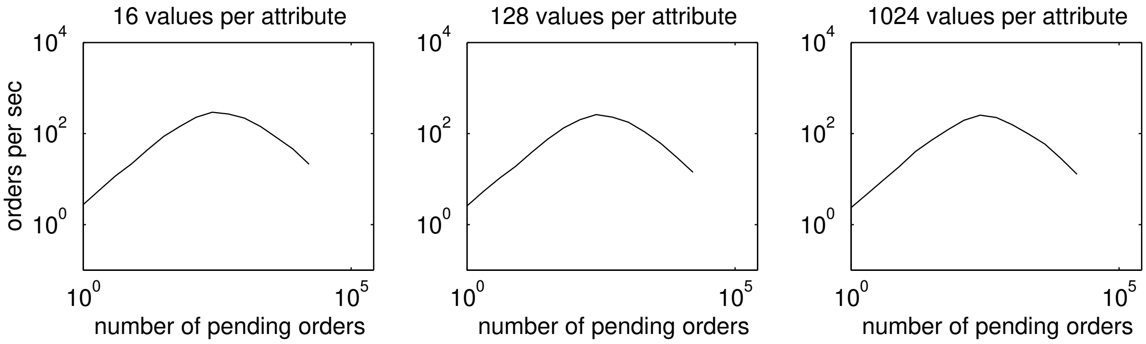


(a) Market with thirty attributes.

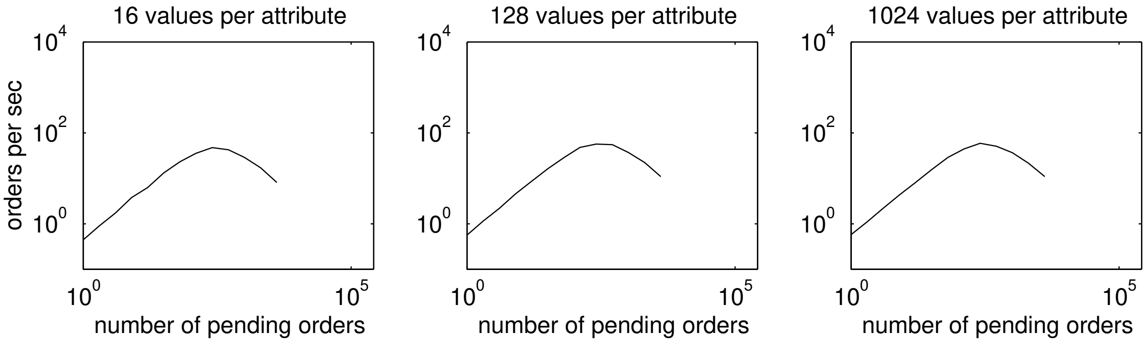


(b) Market with one hundred attributes.

Figure B.31: Maximal number of orders per second, for *matching density of 0.0001.* We consider markets with two values per attribute (left), four values per attribute (middle), and eight values per attribute (right). We show the dependency of the maximal number of orders per second on the number of pending orders. Both horizontal and vertical scales are logarithmic.

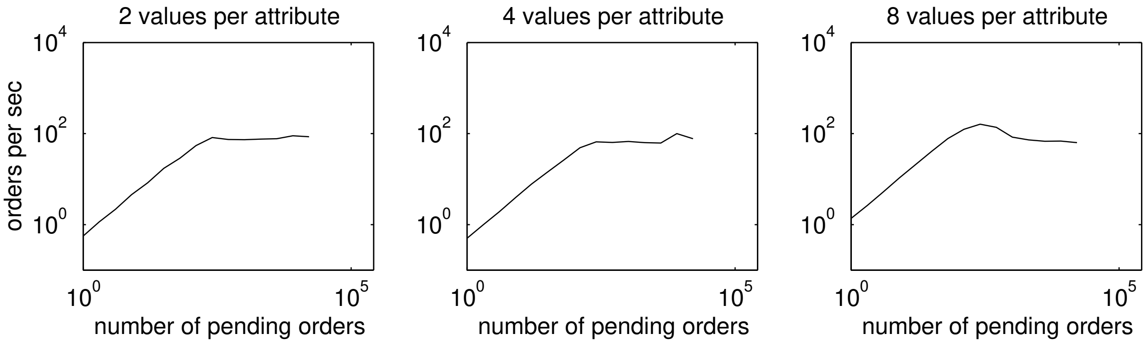


(a) Market with thirty attributes.

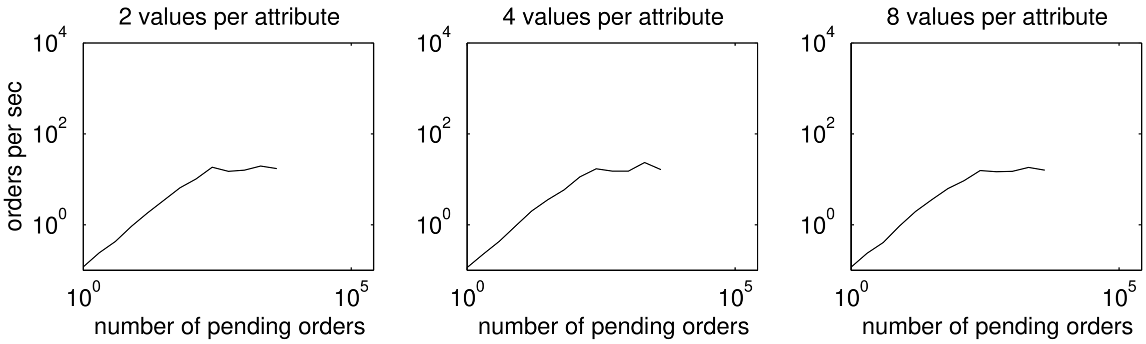


(b) Market with one hundred attributes.

Figure B.32: Maximal number of orders per second, for *matching density of 0.0001.* We consider markets with 16 values per attribute (left), 128 values per attribute (middle), and 1,024 values per attribute (right).

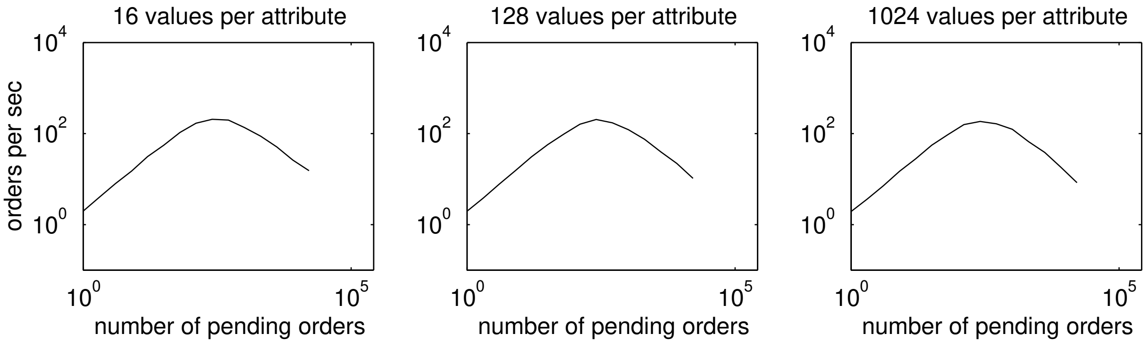


(a) Market with thirty attributes.

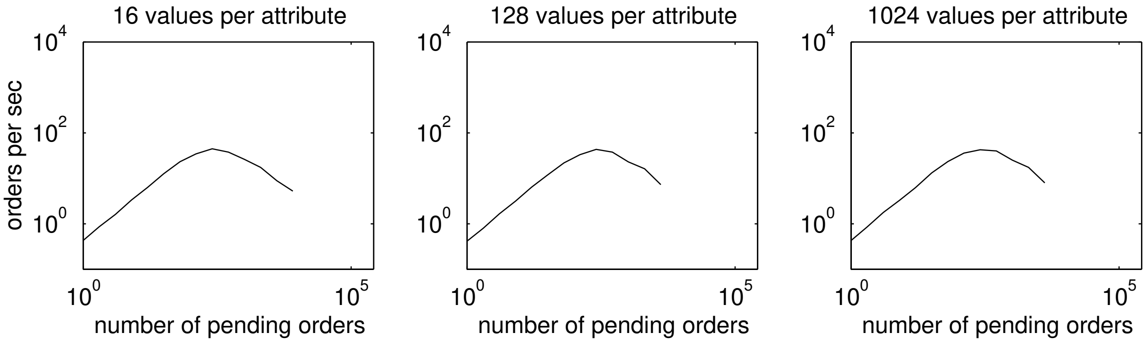


(b) Market with one hundred attributes.

Figure B.33: Maximal number of orders per second, for *matching density of 0.001.* We consider markets with two values per attribute (left), four values per attribute (middle), and eight values per attribute (right).

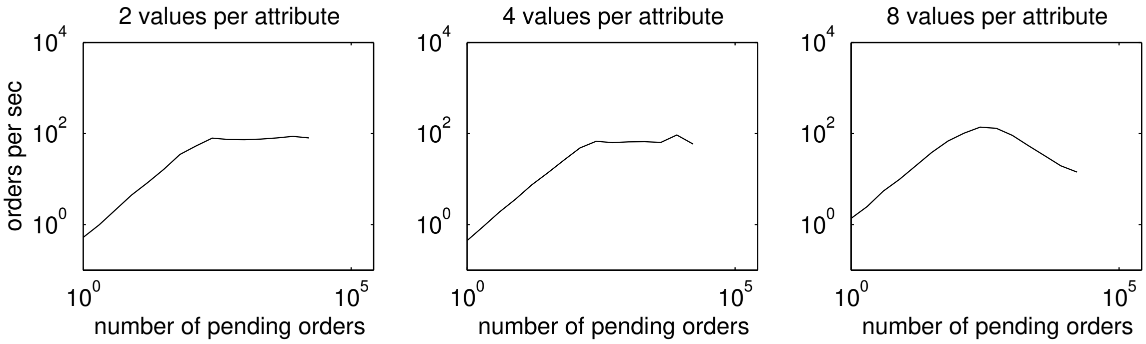


(a) Market with thirty attributes.

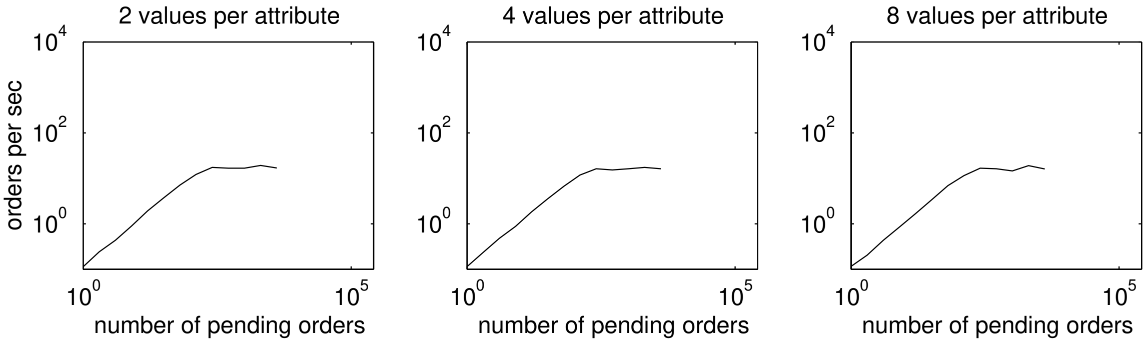


(b) Market with one hundred attributes.

Figure B.34: Maximal number of orders per second, for *matching density of 0.001.* We consider markets with 16 values per attribute (left), 128 values per attribute (middle), and 1,024 values per attribute (right).

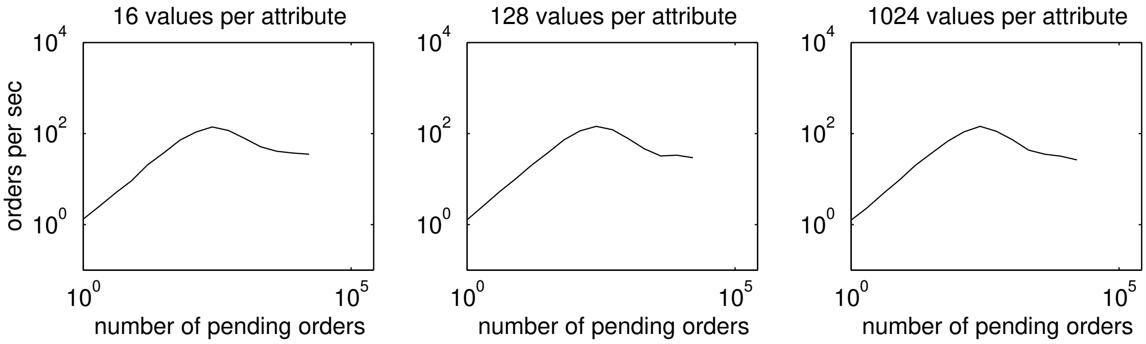


(a) Market with thirty attributes.

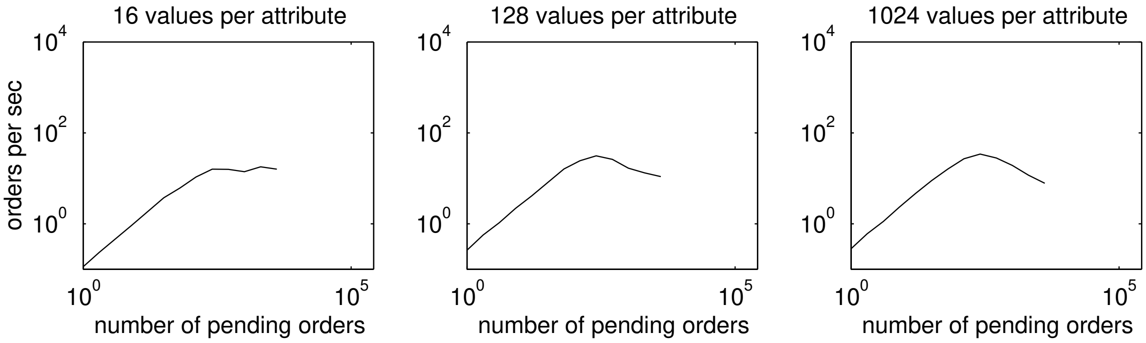


(b) Market with one hundred attributes.

Figure B.35: Maximal number of orders per second, for *matching density of 0.01.* We consider markets with two values per attribute (left), four values per attribute (middle), and eight values per attribute (right).

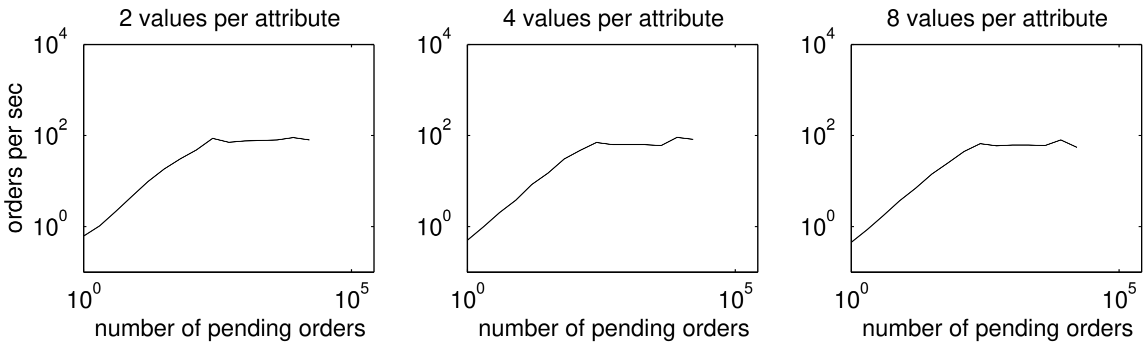


(a) Market with thirty attributes.

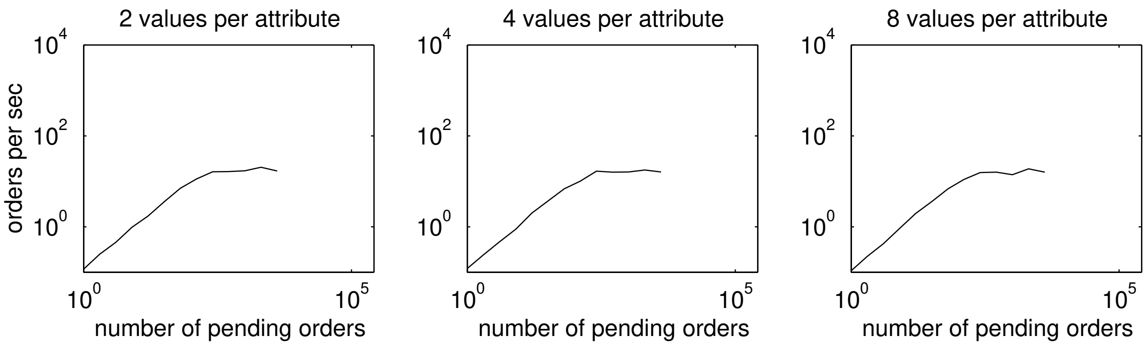


(b) Market with one hundred attributes.

Figure B.36: Maximal number of orders per second, for *matching density of 0.01.* We consider markets with 16 values per attribute (left), 128 values per attribute (middle), and 1,024 values per attribute (right).

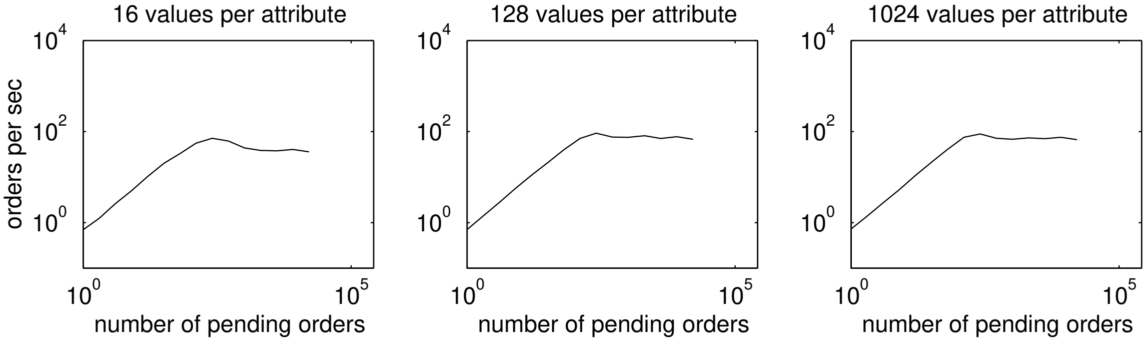


(a) Market with thirty attributes.

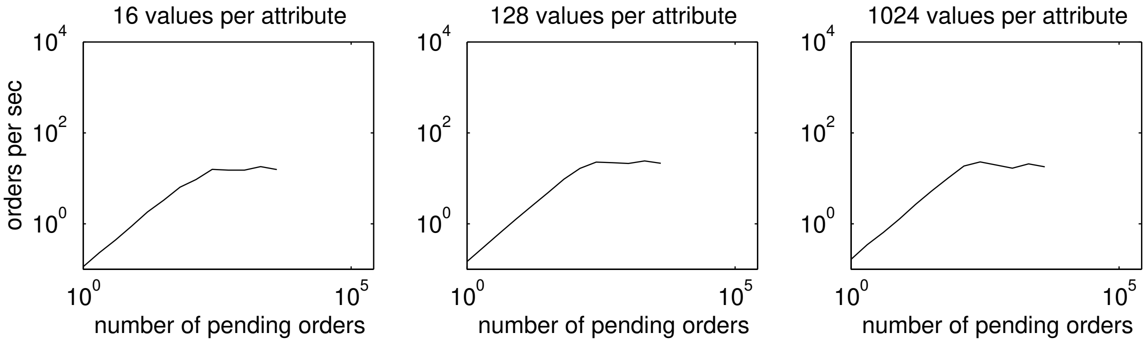


(b) Market with one hundred attributes.

Figure B.37: Maximal number of orders per second, for *matching density of 0.1.* We consider markets with two values per attribute (left), four values per attribute (middle), and eight values per attribute (right).

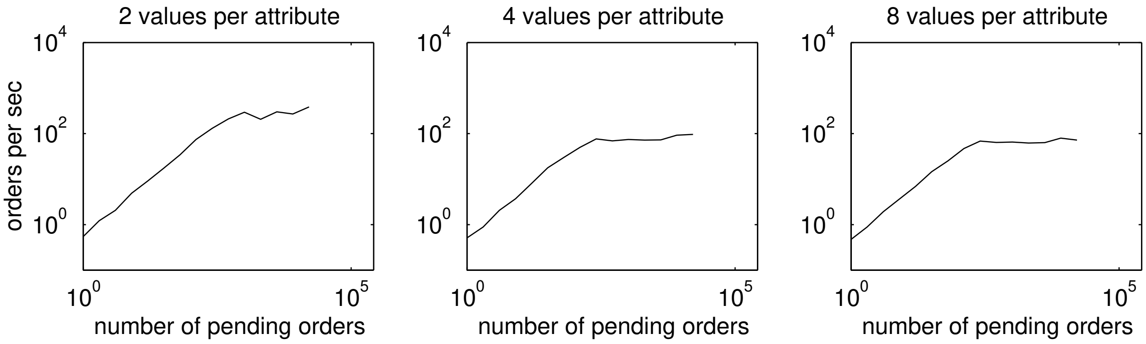


(a) Market with thirty attributes.

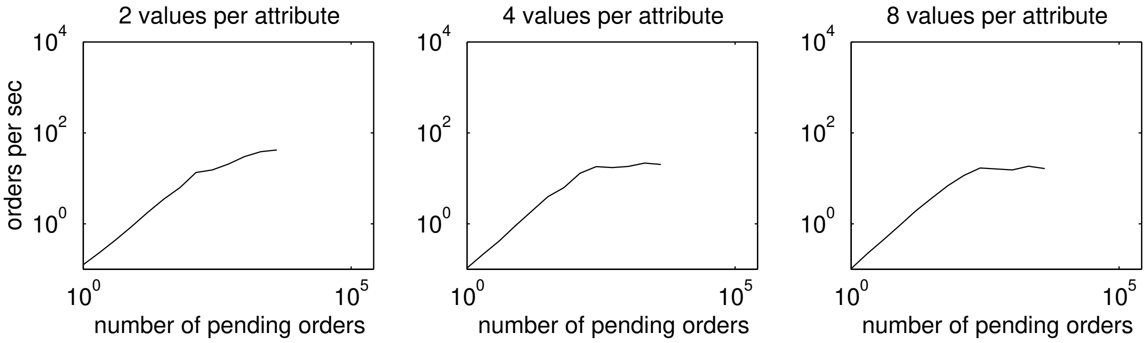


(b) Market with one hundred attributes.

Figure B.38: Maximal number of orders per second, for *matching density of 0.1.* We consider markets with 16 values per attribute (left), 128 values per attribute (middle), and 1,024 values per attribute (right).

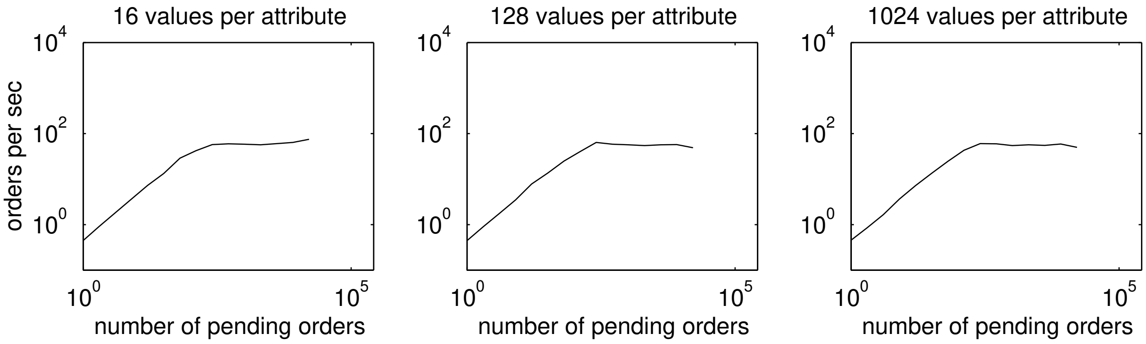


(a) Market with thirty attributes.

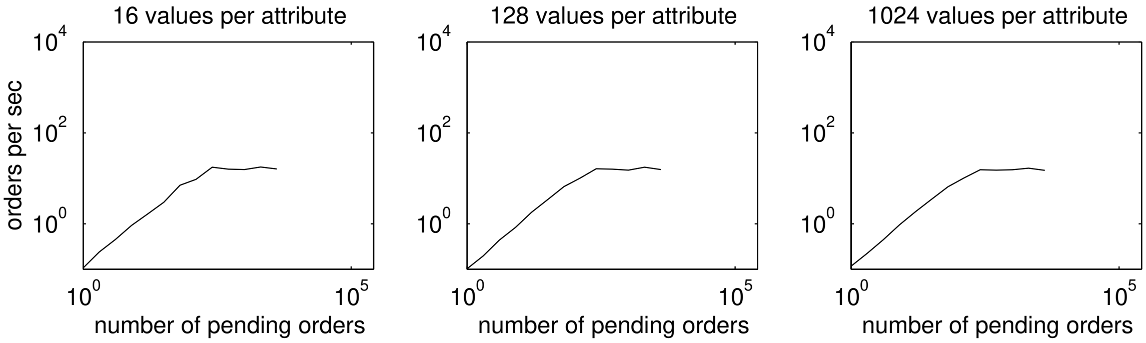


(b) Market with one hundred attributes.

Figure B.39: Maximal number of orders per second, for *matching density of 1.* We consider markets with two values per attribute (left), four values per attribute (middle), and eight values per attribute (right).



(a) Market with thirty attributes.



(b) Market with one hundred attributes.

Figure B.40: Maximal number of orders per second, for *matching density of 1.* We consider markets with 16 values per attribute (left), 128 values per attribute (middle), and 1,024 values per attribute (right).