Multiattribute Call Markets

by

Kevin M. Lochner

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
2008

Doctoral Committee:
Professor Michael P. Wellman, Chair
Professor Jeffrey K. MacKie-Mason
Professor Scott E. Page
Associate Professor Satinder Singh Baveja

To my parents, Lesly Williams Racine and Kenneth Murray Lochner.

Acknowledgments

This thesis could not have been completed without the support of a number of individuals. I'd first like to thank my thesis committee—Michael Wellman, Jeffrey MacKie-Mason, Scott Page, and Satinder Singh Baveja—for contributing their exceptional talents in ensuring the quality of my dissertation.

Certainly the biggest influence on my academic career came from my research advisor, Michael Wellman. He adeptly filled all the formal roles of an advisor, bringing me into existing research projects and guiding me through my own research, while teaching me critical analysis and instilling in me a healthy skepticism. It is perhaps in less expected ways that Mike stands apart as a truly exceptional academic advisor. Through example, Mike teaches qualities that are often grouped together and given the broad label of "professionalism," including respect, patience, fairness, diligence, and integrity. It has been an honor and a privilege to work with Mike while I've been at Michigan.

I'd like to thank Cindy Watts and Kelly Cormier for tackling all the administrative headaches I managed to introduce over the years.

A number of people contributed to the development of AB3D, including Kevin O'Malley, Shih-Fen Cheng, Julian Schvartzman, and Thede Loder.

I owe a big thanks to Yagil Engel. Although he tries to avoid taking credit, many contributions of my thesis derive from the bid expressiveness paper on which he took the lead. Daniel Reeves also deserves special mention, as he has been an indispensable resource throughout my graduate career, playing the role of student mentor from the time that I joined Mike's group.

Many friends and family members have helped me along the way, including Lisa, Sarah, and Eric Lochner, Eliza Dick, Tom Racine, David Dolby, Patrick Jordan, Rahul Suri, Chris Kiekintveld, Yevgeniy Vorobeychik, Yaacov Rubin, and Andrew McHenry. I'd like to specifically thank my brother, Eric Lochner, as I doubt I would have made it to graduate school if not for his support during my undergraduate studies.

Finally, I'd like to thank my mother, Lesly Racine, for eight years of home-cooked Sunday meals and matinee horror movies.

Table of Contents

Dedication ii

Acknowledgments iii

List of Tables vii

List of Figures viii

Abstract x

Chapter 1 Introduction 1
1.1 Auction Notation 5
1.2 Auction Efficiency 7
1.3 Valuations 9
1.4 Auction Design 10
1.5 Auction Specification and Implementation 11
1.6 Summary and Motivation 12
1.7 Overview of Thesis 13

Chapter 2 AB3D: A Market Game Platform based on Flexible Specification of Auction Mechanisms 15
2.1 Architecture 16
2.2 Auction Specification Framework 18
2.3 The AB3D Scripting Language 19
2.3.1 Parameters 19
2.3.2 Rules 20
2.3.3 User-Defined Variables 22
2.3.4 Bid Rules 23
2.4 Auction Engine Architecture 23
2.5 Bidding Languages 26
2.5.1 Divisible Price-Quantity Bids 26
2.5.2 Indivisible Bids 26
2.5.3 Multiattribute Bids 27
2.6 Market Game Specification 29

2.6.1 AB3D Game Description Language . . . 30
2.6.2 GDL Operation . . . 31
2.6.3 Variable Substitution . . . 31
2.6.4 Value Generation . . . 32
2.6.5 Iterative Content Generation . . . 33
2.6.6 Dynamic Game Information . . . 34
2.6.7 Externally Generated Content . . . 34
2.7 AB3D-Supported Research . . . 35
2.8 Conclusion . . . 36

Chapter 3 Iterative Multiattribute Call Market Algorithms . . . 37
3.1 Bidding Language . . . 38
3.2 Clearing . . . 39
3.3 Information Feedback . . . 41
3.3.1 Example . . . 42
3.3.2 Information Feedback under the *MMP-GMAP* Decomposition . . . 43
3.3.3 Link Quote Algorithms for the *MMP-GMAP* Decomposition . . . 46
3.4 Conclusion . . . 55

Chapter 4 Complement-Free Valuations in Multiattribute Auctions . . . 57
4.1 Bidder Valuations . . . 58
4.1.1 Complement-Free Valuations . . . 59
4.1.2 Submodular Valuations . . . 60
4.1.3 Gross Substitutes . . . 60
4.1.4 Syntactic Valuation Classes . . . 61
4.1.5 Valuation Hierarchy . . . 63
4.1.6 Valuations for Multiattribute Auctions . . . 64
4.2 Allocation with Complement-Free Valuations . . . 65
4.2.1 Hardness Results . . . 65
4.2.2 Market-Based Algorithms . . . 66
4.3 A New Valuation Metric . . . 67
4.3.1 Gross Substitutes Revisited . . . 68
4.3.2 Gross Substitutes Violations . . . 68
4.4 Testing the *EGSV*-Efficiency Relationship . . . 70
4.4.1 Valuation Generation . . . 70
4.4.2 Market Simulation . . . 71
4.5 Simulation Results . . . 72
4.6 Conclusion . . . 74

Chapter 5 Multiattribute Supply Chain Simulation . . . 76
5.1 The Trading Agent Competition Supply Chain Management Game . . . 78
5.1.1 The Goods . . . 79
5.1.2 Component Market . . . 80
5.1.3 PC Market . . . 81
5.2 Gross Substitutes Violations . . . 82

5.3 TAC SCM Market Efficiency 83
5.4 Multiattribute Simulation Implementation 86
5.4.1 Scenario Modifications 86
5.4.2 AB3D Auction Implementation 87
5.5 Baseline Manufacturing Agent 90
5.5.1 Component Procurement 90
5.5.2 Manufacturing Scheduling and Shipping 90
5.5.3 Customer RFQ Bidding 91
5.6 Bidding Strategies 92
5.6.1 GoBlueOval Random 93
5.6.2 GoBlueOval Cost 93
5.6.3 Multiattribute Direct 93
5.6.4 Iterative Multiattribute Bidding 94
5.7 Simulation Results 94
5.8 Market-Based Algorithm Convergence 99
5.8.1 Tatonnement Processes 100
5.8.2 Bidding Optimization Quantity Limits 103
5.9 Conclusion 105

Chapter 6 Summary of Contributions 108

Bibliography 111

List of Tables

Table

4.1 Known efficiency approximation factors for *m* goods achievable in polynomial time, from Dobzinski et al. (2005). 66

5.1 List of TAC SCM components. 80
5.2 TAC SCM bill of materials. 80
5.3 Average market efficiency for quarterfinals, semifinals, and finals of TAC SCM competition, years 2003–2005. 85

List of Figures

Figure

2.1 AB3D functional architecture. 17
2.2 A two-phase auction: ascending followed by CDA aftermarket. 22
2.3 Auction engine architecture. 24
2.4 The Message Queue sequences bids and potential auction trigger events. . . 25

3.1 *GMAP* network flow formulation for *OXR* bids. 41
3.2 Network flow clear time as a function of the number of *XR* units using CPLEX solver. 42
3.3 *GMAP* formulation with 3 buy offers and 3 sell offers. The optimal solution is indicated in bold. 44
3.4 *GMAP* formulation of Figure 3.3 with dummy node added for computing a link quote. 45
3.5 Link quotes computed for a bid quote given the *GMAP* formulation of Figure 3.3. 46
3.6 *GMAP* solution for Figure 3.3 with a new sell offer at the quoted price. . . . 46
3.7 Network for Example 2. Adding a unit of flow from XR_D violates the node balance constraint of XR_2. 49
3.8 Network for Example 3. Adding a unit of flow from XR_D does not induce any node balance constraints. 49
3.9 Network for Example 4. Adding a unit of flow from XR_D violates the node balance constraint of XR_3. XR_1 and XR_2 are both absorbing nodes, with the minimum-cost path terminating at XR_2. 50
3.10 *GMAP* formulation for Example 5. 51
3.11 Optimal flows for Example 5. 52
3.12 Residual network for Example 5, bid quote formulation. Absorbing nodes are colored black. 53
3.13 Residual network for Example 5, ask quote formulation. Absorbing nodes are colored black. 53
3.14 CPLEX clear time and shortest path quote time for all link quotes. 54
3.15 Clear and quote times, all methods. 55

4.1 Relationship of mean efficiency and aGSV for different mechanisms. 72

4.2 Relationship of mean efficiency and expected aGSV for different mechanisms. 74

5.1 Diagram of TAC SCM supply chain, with suppliers shown at left, manufacturing agents in the middle, and customer at right. 77
5.2 CDF of *EGSV* values for the TAC SCM bill of materials, using 400 valuations generated with random inventory and costs. 83
5.3 Relationship between efficiency and *EGSV* for the TAC SCM bill of materials with 95% confidence intervals. Each problem instance used 10 contiguous valuations from those depicted in Figure 5.2. 84
5.4 Customer RFQ translated into a multiattribute offer. 88
5.5 Multiattribute manufacturer offer. 88
5.6 Example trigger message to clear the AB3D auction. 89
5.7 CDF of realized efficiencies of `GoBlueOval` strategies for baseline (gbo), randomized (gbo_r), and cost-bidding (gbo_c) implementations. 95
5.8 Mean realized efficiencies with 95% confidence intervals of `GoBlueOval` strategies for baseline (gbo), randomized (gbo_r), and cost-bidding (gbo_c) implementations. Average efficiency achieved in the final round of the 2005 TAC SCM competition is depicted by the dashed horizontal line. . . . 96
5.9 CDF of realized efficiencies for randomized `GoBlueOval` (gbo_r), sealed-bid multiattribute (ma_d), and iterative multiattribute (ma_i) implementations. 97
5.10 Mean realized efficiencies with 95% confidence intervals for randomized `GoBlueOval` (gbo_r), sealed-bid multiattribute (ma_d), and iterative multiattribute (ma_i) implementations. Average efficiency achieved in the final round of 2005 TAC SCM competition is depicted by the dashed horizontal line. 98
5.11 Mean efficiency as a function of bidding iteration for various maximum quantity restrictions on the iterative multiattribute agent bidding optimization. 105
5.12 CDF of realized efficiencies for randomized `GoBlueOval` (gbo_r), iterative multiattribute (ma_i), and optimal subproblem strategy (ma_i^{opt}). 106
5.13 Mean realized efficiencies with 95% confidence intervals for randomized `GoBlueOval` (gbo_r), iterative multiattribute (ma_i), and optimal subproblem strategy (ma_i^{opt}). Average efficiency for final round of 2005 TAC SCM competition is depicted by the dashed horizontal line. 107

Abstract

Multiattribute auctions support automated negotiation in settings where buyers and sellers have valuations for alternate *configurations* of a good, as defined by configuration *attributes*. Bidders express offers to buy or sell alternate configurations by specifying configuration-dependent reserve prices, and the auction determines both the traded goods and transaction prices based on these offers.

While multiattribute auctions have been deployed in single-buyer procurement settings, the development of double-sided multiattribute auctions—allowing the free participation of both buyers and sellers—has received little attention from academia or industry. In this work I develop a multiattribute *call market*, a specific type of double auction in which bids accumulate over an extended period of time, before the auction determines trades based on the aggregate collection of bids. Building on a polynomial-time clearing algorithm, I contribute an efficient algorithm for information feedback. Supporting the implementation of market-based algorithms, information feedback support extends the range of settings for which multiattribute call markets achieve efficiency.

Multiattribute auctions are only one of many auction variants introduced in recent years. The rapidly growing space of alternative auctions and trading scenarios calls for both a standardized language with which to specify auctions, as well as a computational test environment in which to evaluate alternate designs. I present a novel auction description language and deployment environment that supports the specification of a broad class of auctions, improving on prior approaches through a scripting language that employs both static parameter settings and rule-based behavior invocation. The market game platform, *AB3D*, can execute these auction scripts to implement multi-auction and multi-agent trading scenarios.

The efficiency of multiattribute call markets depends crucially on the underlying valuations of participants. I analyze the expected performance limitations of multiattribute call markets, using existing analytical results where applicable. Addressing a lack of theoretical guidance in many natural settings, I introduce a computational metric on bidder valuations, and show a correlation between this metric and the expected efficiency of multiattribute

call markets. As further validation, I integrate multiattribute markets into an existing supply chain simulation, demonstrating efficiency gains over a more conventional negotiation procedure.

Chapter 1
Introduction

From the millions of daily eBay auctions, promoted with colorful television advertisements, to the multi-million dollar art auctions, terminating with the oft-heard phrase "going once, going twice, ... sold!", most people have been exposed at one time or another to the English ascending auction. In this variety of auction, participants place *bids* to buy a good, sequentially offering to buy a good at a price above the currently highest bid. A *price quote* is provided in the form of an *ask price*, which guides bidding by informing participants of the minimum offer at which they would become the highest bidder. The auction terminates when no participants wish to submit additional bids, at which point the ultimate highest bidder wins the good, paying his offered price. One can understand from this process a primary purpose of auctions: to dynamically negotiate the transaction price of a good, resulting in a *trade* between buyer and seller.

This simple auction is only one of myriad varieties found both in practice and in academic literature. One such variety, which helps to motivate this work, is a *reverse* auction. In a reverse auction, a single buyer uses an auction to negotiate the *purchase* of a good from among a group of interested sellers. For example, a buyer interested in purchasing a new computer could hold an auction, letting participants place bids to *sell* to him.[1] The "winner" of such an auction would be the seller with the lowest price, who would provide the buyer with a computer at a price determined by the auction process.

Now consider the case of a buyer who does not have one specific model in mind, but instead has *preferences* as to which models she favors. In this case, an auction could allow sellers to bid both a price as well as a *configuration* of the model they could provide at that price. A single seller may offer several such configurations, each at a configuration-dependent reserve price. For example, a computer defined by processor speed, memory, and hard disk capacity would have configurations such as:

$$\{\text{100MHz Processor, 1GB of Memory, 100GB Hard Disk Drive}\},$$

[1] I attempted to do this in an ad hoc manner on eBay, with mixed results.

or

$$\{\text{100MHz Processor, 2GB of Memory, 200GB Hard Disk Drive}\}.$$

The buyer would select the winning seller based on who offers the combination of both price and configuration that best suits her preferences. Auctions allowing bidders to specify configuration-dependent prices are designated *multiattribute auctions*, reflecting the fact that configurations are defined by the underlying *attributes* of the goods. The goods traded in such auctions are therefore designated *multiattribute goods*, in contrast to the *homogeneous goods* traded in *standard* auctions. Multiattribute reverse auctions, as in this computer example, have been studied in academia (Che, 1997; Branco, 1997; Parkes and Kalagnanam, 2005; Bichler, 2001; Sunderam and Parkes, 2003; Shachat and Swarthout, 2003; Engel and Wellman, 2007), and have been deployed in industry and government for procurement purposes.

For goods that are traded in large quantity, *double auctions* present a natural progression in market structure. Double auctions provide a centralized location (although usually no longer a single *physical* location) where buyers and sellers can meet to trade. Traders may submit buy or sell bids indicating their desired prices and quantities, while the auction rules mediate trades among all participants. As in the English auction, double auctions typically provide an *ask price*, indicating the minimum price of a new successful buy offer given the current set of submitted bids. To aid in the submission of successful sell bids, double auctions additionally provide a *bid price*, designating the maximum price for a successful sell offer. Organizations hosting a group of such auctions are referred to as *exchanges*. The stock market presents the most salient example of a double auction, where investors can offer to buy or sell shares during market hours, or obtain a price quote indicating the prices at which buy and sell trades could be executed.

In contrast to *single-sided auctions* (i.e., one seller per auction), double auctions present traders with a simplified bidding process, as they no longer have to manage a set of bids across independent auctions. Double auctions also offer efficiency advantages by aggregating trade into a single marketplace, determining an optimal set of matches given the the bids of all traders. Additionally, the price information provided by centralized markets provides market participants with useful signals to direct purchasing and production decisions. The prevalence of such organized markets for the exchange of securities and commodities attests to their accompanying benefits, including the provision of trade *liquidity*, i.e., consistent access to a ready group of buyers and sellers during market operation. If a trader needs to quickly buy or sell a stock in large quantity, she can simply go to the centralized exchange and be fairly certain of finding a willing trade partner.

In the spirit of financial exchanges, double auction markets for multiattribute goods

offer the opportunity for enhanced efficiency, price dissemination, and trade liquidity. Taking our computer model as an example, it is easy to see why exchanges are not employed for the trade of multiattribute goods: given that the standard double auction arbitrates only based on price and quantity, the deployment of an exchange for our simple computer model would require a separate market for each possible combination of (cpu, memory, hard disk drive). Assuming only 10 options for each component type, we would need 1000 active auctions to support trade in all computer configurations. Lacking sufficient trade volume to support a double auction for each configuration, trade in multiattribute goods is currently conducted using one-sided auctions and posted-price markets.

Alternatively, a double auction could admit bids that specify configuration-dependent reserve prices, i.e., *multiattribute bids*, thus unifying all trade into a single auction. Previous research on such auctions includes work by Fink et al. (2004), Gimpel et al. (2005), and Gong (2002), all of whom consider the matching problem for *continuous* double auctions (CDAs), where deals are struck whenever a pair of compatible bids is identified. Clearing a multiattribute CDA is much like clearing a one-sided multiattribute auction. Since any transacting bids are immediately matched and removed from the auction, the problem is to match a given new bid (say, an offer to buy) with the existing bids on the other (sell) side.

In a *call market*, in contrast, bids accumulate until designated times (e.g., on a periodic or scheduled basis) at which the auction clears, determining a comprehensive match over the entire set of bids that best balances buy and sell offers. Because the optimization is performed over an aggregated scope, call markets often enjoy liquidity and efficiency advantages over CDAs (Economides and Schwartz, 1995).

For homogeneous goods, a commonly employed and computationally simple type of call market is the *uniform price* double auction with *divisible bids*, meaning that a single price is applied to all transactions generated by a market clearing, and bidders must accept any quantity up to their offered quantity (at a unit price not exceeding their offered price). For this type of auction, there will exist a range of prices that best balance supply with demand. Specifically, there will exist a range of prices for which there may be excess supply (i.e., non-transacting sell bids offered at a price weakly greater than the clearing price), or excess demand (i.e., non-transacting buy bids offered at a price weakly less than the clearing price), but not both simultaneously.

For this type of call market the market-balancing price range, $[\underline{p}, \overline{p}]$, can be identified with little more effort than simply sorting the bids based on price (Wurman et al., 2001), after which the market may be cleared in constant time (or as much time as required to provide transaction information to bidders). Information feedback in this case is typically provided as $[\underline{p}, \overline{p}]$, as these prices constitute the limiting prices for the sale/purchase of a

single unit of the good.

The combination of bid aggregation with multiattribute bidding introduces an algorithmic challenge in designing an auction. With the introduction of multiattribute goods and bids, simply determining the most efficient trade between any two market participants becomes a task requiring non-trivial effort, while computing the most efficient trades in clearing a call market requires solving NP-hard optimization problems for certain classes of bidding languages (Kalagnanam et al., 2001).

Work on multiattribute call markets with Engel and Wellman (Engel et al., 2006) developed a polynomial clearing algorithm for a restricted class of multiattribute bidding languages. Our algorithm supports market operation in a *sealed bid* regime, where agents submit their bids without receiving any information as to the expected transaction prices of different configurations. The constraints we impose on the bidding language restrict the form of expressible multi-unit offers. For example, a participant may have value for up to 10 computers, either Macintosh or PC, but require that they all be of the same type. We allow participants to bid for 10 computers of a specific type, e.g., 10 Macintoshes, and we allow bids specifying 10 computers of any type, where the mix of computers is dictated by the auction, but do not allow bids to express alternative sets, e.g., "10 Macintoshes, $500 each or 10 PCs, $600 each."

Constraints imposed on a bidding language generally restrict the range of bidder *valuations* for which an auction can yield efficiency. In a sealed-bid auction, the hypothetical bidder in the example above would likely end up choosing between bidding on Macintoshes or bidding on PCs. Lacking any information as to the market price of computers (given the current set of bids), the bidder may bid on a computer variety that does not best suit her combined performance and cost preferences.

Given a bidding language that is insufficiently expressive to accurately reflect underlying agent valuations, information feedback may improve the overall efficiency of an auction. Again, in our simple example, price quotes on Macintosh and PC computer models could allow our bidder to submit a bid which yields an outcome as efficient as if she had been able to express her full preferences to the auction.

In this thesis, I extend the aforementioned work on clearing algorithms with polynomial-time information feedback algorithms, presenting an implemented market design supporting clearing and information feedback operations for multiattribute bids. Using both known theoretical results and simulation-based analysis, I characterize the efficiency limitations of my market design with respect to bidder valuation classes, exploring the extent to which information feedback is able to overcome expressive deficiencies of the bidding language.

I also present the *AB3D market game platform*, developed in support of my empiri-

cal evaluation. While motivated to develop this system to support my own research, this system makes a valuable contribution in its own right, supporting the description, implementation, and evaluation of a wide variety of auction types through a novel rule-based *auction description language*.

Before defining the contributions of this thesis more precisely, in the following sections I present formal definitions from auction theory that will be used throughout this thesis, and I frame the auction design and evaluation problem that is central to this work.

1.1 Auction Notation

Auctions mediate the trade of goods among a set of self-interested participants, or *agents*, as a function of agent messages, or *bids*. The goods can be anything from used cars to service contracts, and often an auction will mediate the trade of multiple goods simultaneously. I use x to denote a unique *type* of good, and X to denote the set of all types of goods. An *allocation*, $g \in G$, is a multiset of such goods, i.e., a set possibly containing more than one of each type. A multiset of goods can be formally defined as a pairing of an underlying *set* of goods, and a *function* mapping that set to the positive integers:

$$g = (A, Q) | A \subseteq X \wedge Q : A \mapsto \mathbb{Z}^+,$$

where for any $x \in A$, $Q(x)$ designates the *quantity* of x in g. I use $Q_g(x)$ to denote the quantity of good x in allocation g. For example, given multiset $g_1 = \{x_1, x_1, x_1, x_2\}$, $Q_{g_1}(x_1) = 3$ and $Q_{g_1}(x_2) = 1$.

Bids define one or more *offers* to buy or sell goods. An offer pairs an allocation and a *reserve price*, (g, p), where $g \in G$ and $p \in \Re^+$. For a buy offer, the reserve price indicates the maximum payment a buyer is willing to make in exchange for the set of goods comprising allocation g. Similarly, the reserve price of a sell offer defines the minimum payment a seller is willing to receive to provide allocation g.

A *bid*, $b \in B$, defines a set of offers (often implicitly) which collectively define an agent's reserve price over the space of allocations. Since bids define reserve prices over the entire space of allocations, it is without loss of generality to assume a single bid for each bidder. I use the term *valuation* to designate any mapping from the space of allocations to the nonnegative real numbers: $v : G \mapsto \Re^+$, hence a bid defines a valuation. For ease of explication, I use the function $r : G \times B \mapsto \Re^+$ to indicate the reserve price of a bid for a given allocation. The *bidding language* of an auction defines the syntax of allowable bids, thereby defining the space B of expressible bids.

The majority of formal auction analysis assumes a fixed set of agents. With only slight loss of generality, I further divide agents into buyers $C = \{1, \ldots, i, \ldots, c\}$ and sellers $S = \{c+1, \ldots, j, \ldots, c+s\}$.[2]

Each bidder has a single bid, b_i for buyer i and b_j for seller j. Upon receiving a new or revised bid, the auction determines whether the bid is *admissible* given the current auction state. If admissible, the bid is added to the *order book* of the auction, Ω, comprising the collection of all active buy and sell bids:

$$\Omega = \{\Omega^b, \Omega^s\} = \{\{b_1, b_2, \ldots, b_c\}, \{b_{c+1}, \ldots, b_{c+s}\}\}.$$

When an auction determines the allocations and payments of participants, the process is referred to as *clearing*. In a clear operation, the auction computes a *global allocation* $\{\Theta^b, \Theta^s\}$ comprising an assignment of individual allocations and associated payments:

$$\{\{\theta_1^b, \theta_2^b, \ldots, \theta_c^b\}, \{\theta_{c+1}^s, \ldots, \theta_{c+s}^s\}\},$$

where $\theta_i^b = (g_i, p_i)$ defines an allocation g_i supplied to buyer i in exchange for payment p_i, and $\theta_j^s = (g_j, p_j)$ defines an allocation of g_j supplied by seller j, who receives payment p_j.

A global allocation is *feasible* if the set of goods allocated to buyers is contained in the set of goods supplied by sellers, and the net payments are non-negative.

$$\text{feasible}(\Theta^b, \Theta^s) \iff \begin{cases} \forall x \in X, \quad \sum_{i \in C} Q_{g_i}(x) \leq \sum_{j \in S} Q_{g_j}(x), \\ \sum_{(g_i, p_i) \in \Theta^b} p_i - \sum_{(g_j, p_j) \in \Theta^s} p_j \ \geq 0. \end{cases}$$

A global allocation is *acceptable* if individual payments meet the reserve price constraints expressed in the bids of buyers and sellers.

$$\text{acceptable}(\Theta^b, \Theta^s | \Omega) \iff \begin{cases} \forall (g_i, p_i) \in \Theta^b, r(g_i, b_i) \geq p_i, \\ \forall (g_j, p_j) \in \Theta^s, r(g_j, b_j) \leq p_j. \end{cases}$$

We can now formalize the clear operation as computing a feasible and acceptable global allocation based on the order book. There typically will be multiple global allocations that are both feasible and acceptable. The auction selects one such allocation based on its *clearing policy*, which defines the timing and implementation of the clear operation as a function of the auction state.

[2]This assumption precludes settings in which agents wish to simultaneously buy and sell goods. Chapter 2 is agnostic in this respect, allowing for agents to both buy and sell, while other chapters hold to this assumption. This does allow for agents to both buy and sell if modeled as a separate "buyer" and "seller," but does not allow agents to express a reserve price that is simultaneously contingent on both selling and buying.

In a *direct revelation* mechanism, each agent submits at most a single bid, in the form of a valuation, without receiving any information about the bids of other agents. In contrast, *iterative auctions*, such as the English ascending variety, allow agents to revise their bids over time based on summary information provided by the auction about the current auction state. The *information revelation policy* of an auction defines how and when summary information is provided to agents, possibly depending on the auction state, the sequence of bid submissions, and the agent identities. Summary information is typically derived from the clearing algorithm given the current auction state, informing agents of their current hypothetical allocations as well as *price quotes* indicating the minimum or maximum prices to buy or sell allocations (Wurman et al., 2001).

1.2 Auction Efficiency

In selecting an auction type, one typically has a set of specific objectives in mind. For example, a private individual may try to generate the greatest possible revenue in selling an item. Alternatively, in allocating public resources, the government may want to put goods in the hands of those who most value them. Before delving into the issues of auction design, we first need to formally define the problem.

In formulating the auction design problem, I assume agents have preferences over alternative allocation and payment outcomes which can be represented with *quasilinear* utility functions, meaning that utility is linear in payments. Buyer i then has quasilinear utility function $u_i(g,p) = v_i(g) + m$, where valuation v_i defines the net change in buyer utility when supplied with a given allocation, and m defines the net payments made to the buyer. Similarly, seller j has utility function $u_j(g,p) = -v_j(g) + m$, where valuation v_j is interpreted as a cost function for supplying allocations.

Given agent bids, the auction determines who gets what (the *allocations*), and who pays what (the *payments*). The allocations and payments determine the realized utilities of all agents, and the goal of the auction designer is often to maximize some function of these utilities. In the case of our private seller looking to maximize revenue, the objective function is the sum of all payments made to the seller. For the government trying to "put goods in the hands of those who most value them," the objective function is the sum of all realized agent utilities, also called *global surplus*. Maximization of global surplus is a common goal in auction design, which in many settings has the added benefit of simultaneously maximizing payments. If an auction terminates with maximal global surplus, we call the auction *efficient*.

Formally, the global surplus is the sum over all realized buyer and seller utilities. Since utilities are quasilinear in money, any transfer payments leave global surplus unaffected. The global surplus can therefore be computed directly from buyer and seller valuations, as the sum of realized buyer valuations, less the sum of seller costs:

$$\sigma^*(\Theta^b, \Theta^s) = \sum_{(g_i, p_i) \in \Theta^b} v_i(g_i) - \sum_{(g_j, p_j) \in \Theta^s} v_j(g_j).$$

One obstacle in determining an efficient global allocation is that the auction must compute agent allocations without direct observation of the agent valuations. An intermediate goal of the auction process is therefore to elicit agent preference information. The auction elicits agent preference information through bids. In signifying willing deals, bids place bounds on the agent valuations. To the extent that bids accurately reflect valuations, an auction can use bids as proxies for underlying valuations, and maximize the objective function for the valuations expressed through bids. In the case of global surplus maximization, the auction would instead maximize the *trade surplus* of a global allocation, which is the sum of the buyer reserve prices (expressed through agent bids) for all buyer allocations, less the sum of the seller reserve prices for all seller allocations:

$$\sigma(\Theta^b, \Theta^s | \Omega) = \sum_{(g_i, p_i) \in \Theta^b} r(g_i, b_i) - \sum_{(g_j, p_j) \in \Theta^s} r(g_j, b_j).$$

An auction intended to maximize global surplus will therefore be efficient if the maximization of trade surplus simultaneously maximizes the global surplus. The most simple way to guarantee this equivalence is if agent bids accurately reflect underlying agent valuations. In an *incentive compatible* auction, the auction rules are designed explicitly such that agents find it in their best interests to truthfully and fully reveal their valuations through bids. In the case of quasilinear utility functions, monetary transfers leave global utility unchanged, so the allocation of goods is chosen to optimize the desired objective function, while monetary transfers are used as incentives to induce truthful valuation revelation from agents.

Incentive compatibility is not prerequisite for an efficient outcome. In some settings, for example, despite inducing bids strictly lower than agent valuations, the first-price sealed-bid auction produces an efficient outcome equivalent to the incentive-compatible *Vickrey* auction (Myerson, 1981; Riley and Samuelson, 1981). Furthermore, iterative auctions are often implemented such that a minimal amount of preference information is conveyed through bids to achieve efficiency. An analog of incentive compatibility often applied to iterative mechanisms is *ex-post incentive compatibility* with *straightforward bidding* (Parkes

and Ungar, 2000; Ausubel and Milgrom, 2002). This translates into ensuring that agents behave optimally by bidding truthfully at each iteration on the set of goods which maximize their own utility given the current price quotes.

1.3 Valuations

An important consideration in selecting an auction mechanism is the class of valuations from which participants' preferences are drawn. If bidders have valuations defined over *combinations* of different goods, mechanisms that allocate efficiently with respect to such *combinatorial* valuations must be considered.

Combinatorial valuations include preferences exhibiting *complementarity* and *substitutability*. For example, a bidder with complementary preferences may have value for owning a unit of good *a* in conjunction with one unit of good *b*, but have no value for owning a single unit of either good in isolation. The most common example used for complementary preferences is that of a left shoe and a right shoe—a person typically has significantly higher value for the pair than for either shoe individually. Substitute preferences dictate that raising the price of one good does not lower demand for other goods. For example, consider two different brands of shoes: raising the price of Nike shoes should not reduce a person's desired quantity of Adidas shoes, indicating that most people have substitute preferences for these shoe brands.

The exponentially sized offer specifications induced by combinatorial valuations present a particularly hard allocation problem, both in the expression of agent valuations (Segal, 2005) and in the algorithmic problem of computing optimal allocations (Sandholm, 2005; Sandholm et al., 2002). For certain subclasses of multi-unit valuations, however, the computation of efficient outcomes is made tractable. Notably, for valuations satisfying the *gross substitutes* condition, it is well known that a Walrasian equilibrium exists, and a market-based algorithm admitting offers only on individual goods, in response to dynamically updated price quotes, can provide a fully polynomial approximation scheme for the computation of efficient allocations (Lehmann et al., 2006).

Since a configuration in multiattribute negotiation corresponds to a unique type of good, the class of multi-unit valuations for multiattribute goods is equivalent to the class of combinatorial valuations. The problem of multi-unit multiattribute allocation therefore inherits the hardness results derived for combinatorial auctions, but moreover applied to a cardinality of goods that is itself exponential in the number of attributes. The adaptation of combinatorial auction algorithms to multiattribute domains thus presents a new and chal-

lenging problem, as such algorithms typically assume (at least for practical purposes) a predefined and modest-sized set of goods.

1.4 Auction Design

In its earliest incarnation, auction design was practiced exclusively through analytical means, while laboratory testing with human subjects verified (or possibly refuted) analytical results. Early work in auction design produced many important theoretical results, perhaps most notably the *revelation principle* (Myerson, 1981, 1979) and the family of *Vickrey-Clark-Groves* (VCG) mechanisms (Vickrey, 1961; Clarke, 1971; Groves, 1973).

The revelation principle states that the outcome of any mechanism can be implemented using an incentive compatible direct revelation mechanism. This result equivalently states that in designing an auction for a specific setting, one can restrict attention to direct revelation mechanisms in which agents are truthful. VCG mechanisms operate on the principle that agents can be induced to truthfully reveal their valuations if their payments are independent of their reported values. This powerful result has led to a large class of auction mechanisms which can be shown to terminate with what is designated the *VCG outcome*.

As the complexity of goods and trading situations has increased, the powerful game-theoretic tools used in the design of auction mechanisms have increasingly been supplemented with empirical and often agent-based simulation when applied to complex mechanisms (Rust et al., 1994; Neumann et al., 2002; Walsh et al., 2002; Phelps et al., 2002; Fasli and Michalakopoulos, 2008; Jordan et al., 2007). Where human testing has historically been used to empirically validate analytical results, the increasing complexity of auctions has rendered human experimentation impossible in many settings.

Recently, VCG-based approaches have drawn criticism in academic literature. For example, VCG mechanisms have been challenged by questions of realism in agent modeling assumptions (Rothkopf, 2007), as well as by questions of robustness to bidder collusion or deception (Yokoo et al., 2004; Conitzer and Sandholm, 2004a). Additionally, the revelation principle has drawn criticism in settings where agents are unable to easily determine and reveal preference information (Larson and Sandholm, 2001), and in settings where the clearing task is intractable given direct preference revelation (Conitzer and Sandholm, 2004b).

Citing the importance of implementation details, problem complexity, and context, recent work by Reeves (2005) and MacKie-Mason and Wellman (2006) advocates a computational approach to mechanism analysis, whereby a game is induced by a restricted (and

possibly parametrized) set of agent strategies, after which the game may be approximated through simulation and subsequently solved. This approach has been applied to settings in which the complexity of the mechanism precludes purely analytical results (Wellman et al., 2005), and has been developed into an algorithmic process for mechanism design (Vorobeychik et al., 2006).

1.5 Auction Specification and Implementation

Prerequisite for computational testing are both an implemented candidate mechanism and implemented models of agent behavior, often consisting of a class of agent valuations and beliefs (Porter et al., 2003). A mechanism may then be tested on various problem instances to derive performance results. Many practitioners of computational economics design what amount to single-scenario simulation environments tailored specifically to their research needs (Reeves et al., 2005; Jordan et al., 2007; Rust and Miller, 1993; Arunachalam and Sadeh, 2005). If able to meet the needs of a diverse group of practitioners and research agendas, a general-purpose modeling environment would mitigate a significant barrier to computational approaches by reducing software development time. Several such economic modeling platforms have been introduced in literature, typically employing parameter-driven auction implementations in achieving varying levels of generality (Collins et al., 2002; Wurman et al., 2001; Sandholm, 2004).

With the growing complexity of auction mechanisms, formally expressing the rules of an auction has become a challenging task. Detailed bidder eligibility rules, time-dependent auction procedures, as well as highly complex clearing rules have generated auction specifications of great length and complexity. For example, the most recent documentation of procedures for FCC spectrum auctions comprises over 100 pages (FCC, 2007). Special-purpose auction description languages may facilitate the communication of complex mechanisms (Lomuscio et al., 2000; Reeves et al., 2002; Rodriguez-Aguilar et al., 1998; Wurman et al., 2001). Specifying an auction mechanism in a high-level, special-purpose language promotes a standardization of interfaces and transparency of encoding. As Rolli et al. (2006) point out, a machine-readable auction language could ultimately enable automated trading agents to reason about their strategies. If executable, auction description languages facilitate the rapid deployment and testing of mechanisms, while also enabling automated search and analysis of the design space (Cliff, 2003; Phelps et al., 2002).

1.6 Summary and Motivation

The overarching goal of this thesis is largely practical: to design and implement multi-attribute call markets which have performance characteristics making them suitable for general use. In approaching this goal, I make several contributions to the more general field of auction design.

Underlying any iterative auction are algorithms for computing allocations and providing quote information. For an auction to be practical, these algorithms must be computationally efficient on problem sizes likely to be encountered in practice. I present a polynomial-time clearing algorithm from joint work with Engel and Wellman, and provide empirical complexity results from randomly generated problem instances. I additionally present a novel information feedback algorithm that can produce price quotes with polynomial complexity, again presenting empirical complexity from randomly generated problem instances. This pairing of clearing and information feedback algorithms is able to support the first known polynomial-time multiattribute call market.

I have implemented these call market algorithms into a functional auction server, along with support for two distinct methods of multiattribute bidding. This work therefore presents the first *implemented* multiattribute call market. The auction server on which I implemented these algorithms, the *AB3D market game platform*, also serves as a contribution to the field of auction design. Designed to support game-theoretic auction research, the AB3D market game platform is capable of implementing a broad range of auctions, while additionally providing facilities for the description and implementation of multi-auction market games. This includes facilities for distributed execution of multi-auction markets, invocation of user-defined agents for simulation, dynamic agent valuation generation, and evaluation of market outcomes for game-theoretic mechanism analysis.

The AB3D market game platform encompasses another contribution of this thesis: an executable auction description language, the *AB3D scripting language* (AB3DSL). Scripts written in AB3DSL define auctions through a combination of static parameter settings and dynamic rule-based behavior invocation. For example, a parameter may specify the bidding language for an auction, while a rule may dictate that the auction will clear after any bid submission. This novel approach supports a compactness of representation that is superior to most specification efforts of comparable scope.

In addition to designing and implementing multiattribute call markets, I make contributions in assessing the applicability of this auction design. As noted, auctions are typically chosen based on expected bidder valuations, with performance degradation often resulting when the bidding language is insufficiently expressive to fully convey agent valuations. I

investigate the extent to which the information feedback algorithms I develop are able to compensate for the lack of expressive power of our multiattribute bidding language. In the course of my analysis, I present a new metric on bidder valuations, based on the degree to which valuations violate technical conditions for efficiency. I present evidence that this metric is indeed correlated with the expected efficiency loss of multiattribute markets, and that this efficiency loss is successfully mitigated through the use of information feedback.

Finally I present results from integrating multiattribute markets into an existing supply chain simulation. These results provide evidence for the applicability of multiattribute markets in real-world domains, while also serving as a demonstration of the flexibility of the AB3D system. After presenting mediocre performance using the myopic bidding strategies of standard market-based algorithms, I explore how my iterative implementation deviates from provably convergent market-based algorithmic approaches. I use a stylized subproblem to find bidding strategies yielding improved efficiency given my iterative implementation. I then use one such strategy in the full supply chain simulation in demonstrating an efficiency advantage of multiattribute markets over a more conventional negotiation procedure.

1.7 Overview of Thesis

In Chapter 2, I begin with a presentation of the AB3D market game platform and AB3D scripting language. In presenting AB3D, I describe the more general auction design space, which will provide perspective for ensuing chapters. I discuss limitations of strict parameter or strict rule-based approaches, giving examples of how my novel combination of parameters and rule-based behavior allows for the specification and implementation of a very broad range of auction mechanisms.

In Chapter 3, I define the algorithms supporting multiattribute call markets in AB3D. I outline work with Engel et al. (2006) that developed polynomial-time network flow clearing algorithms for call markets by placing restrictions on the form of expressible offers. I present those results in the context of the specific form of bidding language incorporated into AB3D, describing the clearing algorithm and presenting empirically derived complexity results. I then build on that work with a framework for information feedback algorithms based on the same network flow structure of the clearing algorithm. I present a procedural example demonstrating how quote information is computed given a solution to the clearing algorithm, and give both theoretical and empirically measured complexity results.

Chapter 4 takes the call markets developed in Chapter 3 and quantifies their efficiency

limitations with respect to bidder valuations. Since multi-unit multiattribute markets are a generalization of both standard double auctions and combinatorial auctions, I first review existing analytically derived efficiency limitations for combinatorial settings. I then focus on the use of information feedback for the implementation of market-based algorithms in reaching favorable allocations. I review the existing literature on market-based algorithms, including the technical condition of gross substitutes under which such algorithms have been shown to obtain efficient outcomes.

I next present results from an agent-based simulation, using a model of valuations inspired by an existing supply chain simulation, the Trading Agent Competition Supply Chain Management game (TAC SCM) (Arunachalam and Sadeh, 2005). I show that in TAC SCM, the induced agent valuations naturally violate the substitutes condition, rendering inapplicable the efficiency results for market-based algorithms. Lacking analytical efficiency results given violations of the substitutes condition, I present a new metric for bidder valuations based on the severity with which agent valuations violate this condition. I provide experimental evidence from an agent-based simulation suggesting a correlation between this metric and expected efficiency loss, and also demonstrate that this efficiency loss is mitigated through the provision of information feedback.

In Chapter 5, I evaluate multiattribute markets in the context of the full TAC SCM game. This analysis is intended to demonstrate the benefit of a multiattribute market over an existing alternative protocol, given bidder valuations derived from a grounded simulation environment. Whereas the goal of participants in TAC SCM is to maximize profit over the course of each game, one can measure the success of the market itself in efficiently allocating resources in meeting final demand. I first describe my methods in adding AB3D to the TAC SCM simulation environment, replacing the existing customer negotiation protocol with a multiattribute call market. I describe the baseline agent used for simulations, `GoBlueOval`, my entry in the 2005 TAC SCM competition. Using a tool developed to measure market efficiency for the TAC SCM 2005 tournament (Jordan et al., 2006), I show that the introduction of a multiattribute auction into TAC SCM increases market efficiency both over that achieved by my baseline agents, and over that achieved in the 2005 TAC SCM competition.

Chapter 2

AB3D: A Market Game Platform based on Flexible Specification of Auction Mechanisms

This chapter presents the AB3D market game platform and AB3D scripting language. The design of AB3D builds on lessons learned from the Michigan Internet AuctionBot (Wurman et al., 1998b), which supported a large variety of auction types through a *parametrization* of the auction design space (Wurman et al., 2001). In attempting to extend this parametric specification to a wider variety of designs, I experienced a decreasing orthogonality as I introduced parameters. In other words, it became increasingly necessary to introduce parameters switching on and off whole areas of the design space, thus diluting the benefit of previous specification effort. Parameters defining the auction *control structure*—the temporal pattern of auction events—seem particularly ill-suited to parametric description. In general, the timing of a particular action within an auction may depend on arbitrary features of the auction history. For specification of such functional dependence, control structures reminiscent of programming languages may be more effective than simple parameter settings.

To support the wider design space, I introduce a simple and extensible rule-based language for the specification of auction mechanisms. The language combines parameter specification with rule-based invocation of auction behaviors, providing sufficient flexibility to capture a wide range of known and conceivable auctions, using a natural and transparent encoding.

Given the ability to express increasingly complex market mechanisms, the need remains to evaluate alternative designs. Standard game-theoretic tools used in the evaluation of simple auction mechanisms have increasingly been supplemented with empirical and often agent-based simulation when applied to complex mechanisms (Rust et al., 1994; Neumann et al., 2002; Walsh et al., 2002; Phelps et al., 2002; Fasli and Michalakopoulos, 2008; Jordan et al., 2007). Agent-based simulation requires embedding the markets in a particular

scenario or context. Operating markets in a specified scenario can achieve tasks such as testing, training, or actual trading.

To support these tasks, I have integrated my auction description language and execution system into a more general market game description and simulation system. I call this platform *AB3D*, in reference to the two previous generations of AuctionBot ("AB"), and a three-dimensional characterization of auction policy space ("3D") (Wurman et al., 2001).

A game description language (GDL) supports the simulation of complex Bayesian games with probabilistic and programmatic agent preference generation, as well as dynamic agent preference modification. Designed with a view toward extensibility, GDL interacts with my auction description language to describe and implement higher-level multi-market games, including information about objectives needed for the AB3D system to automate the scoring of game outcomes.

Following description of the high-level AB3D architecture, I describe my auction description language and execution environment, as well as the game description language. I provide some simple examples demonstrating the generation of diverse auction and game behaviors. I conclude with examples of current and prior AB3D-supported research.

2.1 Architecture

The high-level architecture of AB3D is shown in Figure 2.1. From the point of view of a trading agent, the AB3D system is a black box with two socket-based access points: the *Game Scheduler* and the *Agent Manager*; other components of the system are internal, and not visible to the agent. Using an XML-based protocol, users access the Game Scheduler to instantiate a new game, using the Agent Manager to communicate with AB3D during a game. AB3D supports the use of any trading entity which interacts with the system during a game (obtaining preference information, submitting bids, requesting quote and transactional data) via a socket-based XML protocol. For Java-based trading agents, moreover, the system supports automatic invocation based on market events. Independent graphical interfaces for human traders can similarly access the AB3D system via its socket-based protocol. Cheng has implemented one such interface, for a commodity-trading scenario discussed below (Cheng, 2007).

From an architectural point of view, the AB3D system is divided into a central *System Manager*, and a set of one or more *Auction Supervisors*, which instantiate and terminate individual auctions. Auction Supervisors run independently (and often on a separate host) from the System Manager, and may be distributed across multiple host computers. This

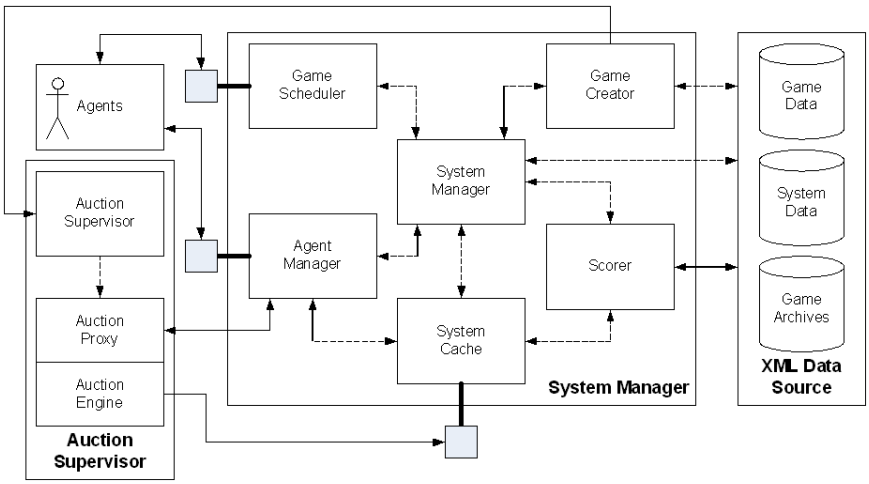

Figure 2.1: AB3D functional architecture.

distributed architecture supports multi-auction games that would be computationally infeasible on a single machine, while the Agent Manager provides a central communication point for agents to access all active auctions.

The System Manager is the main coordinator of the AB3D system. It is responsible for most of the components that handle agent interaction, game creation, storage of game information, and game scheduling. Within the System Manager, the *Game Creator* is responsible for all operations required to instantiate a game. This includes creating auctions, generating agent-specific information, running internal system agents (if any), and invoking the scorer at the end of the game. A *System Cache* provides a system-wide location for holding a game's runtime data, including bids, quotes, transactions, and agent preference information.

Auction Supervisors provide an environment for the management and execution of individual auctions. The System Manager (and its components) send communication messages to the Auction Supervisors to invoke and tear down individual auctions, which are themselves divided into an *Auction Engine* and an *Auction Proxy*. The purpose of the proxy is to read and forward the messages to the Auction Engine. This design enables auction developers to concentrate on auction-specific code, thus relieving them of some implementation details. I describe the Auction Engine further below in my presentation of the AB3D scripting language.

To compute post-game scoring information, the system solves each agent's *allocation problem* (Boyan and Greenwald, 2001): assigning final holdings of goods to their possible tasks (or other uses) in order to maximize its designated objective. The user specifies the objective in terms of a standard mathematical modeling language (e.g., AMPL (Fourer

et al., 2002)), and the AB3D system employs a generic scorer which only requires a defined mapping between task preferences and data used in the mathematical model.

2.2 Auction Specification Framework

The general class of *auctions* comprises all mediated mechanisms that determine market-based allocations (i.e., exchanges of goods and services for money) as a function of agent messages. These messages, or *bids*, are typically composed of *offers* specifying deals in which the agent is willing to engage. Although the form and content of bids, along with the auction's behavior given such bids, can vary widely among auction mechanisms, there are several common constructs that can be defined across the entire space (Wurman et al., 2001).

Upon receiving a new or revised bid, the auction determines whether the bid is *admissible* given its current state. If so, the bid is admitted to the *order book*, a repository representing the current collection of active offers. At some point (depending on the auction rules, of course), the market *clears*, producing a set of exchanges matching compatible offers in the order book according to the auction's *clearing policy*. Along the way, the auction may send messages to participants providing information about auction state (often in highly summarized form), according to its *information revelation policy*. Since this information often—though not invariably—includes current *price quotes* (i.e., indications of what the hypothetical clearing prices would be in the current state), I refer to both the action and revealed information as a *quote*.

Following Wurman et al. (2001), I maintain that the substantial differences among auction mechanisms can be characterized by their policies for the three major activities described above: processing bids, clearing, and revealing information. I further decompose the specification of these policies into

1. their functional implementation (i.e., the ***how***), and
2. their timing (the ***when***).

My decomposition strikes a middle ground among approaches to auction specification, where the *how* is specified primarily through parameter settings, and the *when* through rule condition patterns. Alternative approaches have tended either toward strict use of parameters (Wurman et al., 1998b), or strict avoidance of parameters (Rolli et al., 2006). I have found that while the temporal characteristics of an auction can be characterized by a rule-based language without computational penalty, many clearing and information feedback policies are impractical without specialized algorithms. For example, implementing clear-

ing algorithms for the bidding languages described in Section 2.5 with a general auction description language would likely render the mechanisms unusable.

2.3 The AB3D Scripting Language

An AB3D auction script comprises a sequence of statements, of four types:

1. *initialization* of auction parameters,
2. *rules* that trigger auction events and parameter changes,
3. *declarations* of user variables, and
4. *bid rules* defining additional bid admissibility requirements.

The language also supports the organization of statements into distinct phases or *rounds*, denoting named regimes of auction control.

2.3.1 Parameters

Static auction policies are characterized by predefined *parameters*, conditioning the criteria for admitting bids, matching offers, and summarizing state information in quotes.

For example, one commonly adopted element of bidding policy is a "beat-the-quote" (btq) rule, which requires that any new or revised offer meet or exceed the standing offer (as represented by the quote), in some well-defined way (which might further depend on the bidding format). Wurman et al. (2001) describe this policy element, as well as many others straightforwardly encoded as AB3D parameters.

In the AB3D scripting language, parameter values are initialized and revised through *assignment* statements, expressed using the *set* keyword. The statement

`set` *param expr*

dictates that the parameter *param* be (re)assigned the value of expression *expr*. For example, the statement

```
set bid_btq 1
```

activates the beat-the-quote requirement, thus instructing the auction that bids must compare favorably with the price quote to be admitted to the order book.

Assignments may appear unconditionally as initializations or reassignments as part of a control structure, or conditionally as part of rules. The latter facility provides for qualitative modification of auction policy while the auction is active.

2.3.2 Rules

Whereas the static features of auction policy may be best characterized parametrically, such an approach is quite limited for specifying the dynamic control of auction events. For this purpose we employ a simple rule language, allowing that auction events be conditioned on arbitrary functions of auction state. These functions may also be parametrized, thus providing the benefits of both constructs.

An AB3D rule takes the form:

when | *while* (*conditions*) {*actions*}

In this rule, *conditions* is a conjunction of boolean-valued predicate expressions, written as a sequence of such expressions separated by `and` keywords, enclosed by parentheses. An individual predicate expression evaluates to *true* or *false* depending on the current auction state. If all are true, the rule is triggered, and *actions* are executed. Each action on the *actions* list (delimited by semicolons, enclosed in curly braces) corresponds to an auction activity, such as quoting or clearing, or an internal event such as assigning a parameter. The keyword *when* or *while* designates whether a rule is to be invoked only on becoming true (*when*), or continually until the condition no longer holds (*while*).

Multiple rules with the same actions essentially represent a disjunction of their corresponding conditions.[1] Arbitrary boolean combinations can thus be expressed in this manner.

Condition expressions reference *state variables*—either predefined auction state variables or user-defined script variables. Auction state variables represent summary measures significant to auction operation, designated by auction developers for exposure to the auction script interpreter. Examples of state variables routinely provided by AB3D auctions include the current time (`time`), the last time a clear was executed (`lastClearTime`), and the number of bidders currently eligible to bid (`numBuyers` and `numSellers`). A typical condition expression using such variables would be to compare the current time with some other state variable. For example, an auction can specify a predefined duration (say, 500 seconds) with a rule of the form:

```
when (time ≥ auctionStartTime + 500)
  {close}
```

[1] Multiple rules with the same action differ from a disjunction in that if they happen to hold simultaneously, the action would executed once for each rule.

Though the previous rule may be slightly more intuitive than a parametrized representation, it could certainly be captured under parametrization without difficulty. Consider an auction that requires a clear to be performed whenever a bid is admitted (causing the state variable `validBid` to become *true*), but only before a predefined period of time:

```
when (validBid and
      time ≤ auctionStartTime + 500)
  {clear}
```

If this pattern of conditions is sufficiently common, then it too could be captured in a dedicated parameter. However, as we consider further state variables, the number of boolean combinations grows exponentially, as would the number of parameters required to capture the policy possibilities. The rule language enables this expressiveness without the associated blowup in primitives.

Periodic events are a common feature of auction mechanisms. In the AB3D scripting language, internal state variables referencing the last execution time of any given action provide a natural way to specify periodic events. For example, an auction that performs a clear every 100 seconds would include the following rule:

```
when (time ≥ lastClearTime + 100)
  {clear}
```

Assignment within a rule action is a powerful construct, enabling dynamic modification of auction behavior. For example, the US Treasury auctions its "T-bills" in a uniform-price auction, after which winning bidders may trade the bills in a secondary market (Bikhchandani and Huang, 1993). The script listed in Figure 2.2 defines such a two-phase mechanism, comprised of an ascending auction for the first 1000 seconds, followed by a continuous double auction (CDA) (Friedman and Rust, 1993) for the second 1000.

This two-phase example employs several parameters controlling the clearing policy for a multi-unit auction. The `pricing_fn` parameter identifies the criterion to be used for determining prices. A value of *chronological* indicates that price is determined according to the relative submission times of the matching bids comprising an exchange. Let p_0 be the price of the earlier bid, and p_1 the price of the later (by the fact that they match, we know the buy price is at least as great as the sell, but either could be earlier). The value $k \in [0,1]$ of the `pricing_k` parameter dictates how the transaction price, p_*, is selected from the compatible range:

$$p_* = p_0 + k(p_1 - p_0).$$

For CDAs, a new bid transacts with a standing bid in the order book at the price specified by the standing bid. Thus, the CDA policy has chronological pricing with $k = 0$.

```
defAuction twoPhase {
  set bid_btq 1
  set pricing_fn uniform
  set pricing_k 0
  when (time = auctionStartTime + 1000)
    {clear; quote;
     set pricing_fn chronological;
     set bid_btq 0}
  when (time ≥ auctionStartTime + 1000
        and validBid)
    {clear; quote}
  when (time = auctionStartTime + 2000)
    {close}
  }
```

Figure 2.2: A two-phase auction: ascending followed by CDA aftermarket.

A *uniform* pricing function produces a single price governing a collection of simultaneous exchanges. In general (for a multi-unit auction with divisible offers) there will be a range of possible clearing prices (Wurman et al., 1998a), delimited by the *bid* and *ask* quotes. Designating the bid by p_0 and the ask by p_1, the *k*-double auction (Satterthwaite and Williams, 1989) selects within this range according to the linear interpolation above. Note that I economize on parameters by reusing the interpolation parameter *k* for both the uniform (for which it was named) and chronological cases.

2.3.3 User-Defined Variables

User-defined variables add a degree of flexibility to the language, allowing for more general forms of auction control structures such as counters or looping constructs. Such a construct can be used to specify a predefined number of periods comprising the auction. The following set of declarations and rules would generate seven rounds of 60 seconds.

```
declare userTime auctionStartTime
declare counter 0
when (time ≥ userTime + 60)
   { set userTime time;
    set counter counter + 1;}
when (counter=7) {close}
```

Often an auction will have variables that are defined for each bidder, for example a user-specific credit limit. For this special case of user-defined variables, AB3D provides a bidder-indexed array construct, created (and optionally initialized to a value) with a statement of the form:

```
bidderAttribute ATTRIBUTENAME [val]
```

Bidder attributes can be referenced either by the bidder ID:

```
ATTRIBUTENAME(ID),
```

or if the rule is associated with a bid submission by the keyword `bidder`:

```
ATTRIBUTENAME(bidder).
```

2.3.4 Bid Rules

In the same way that I found rule-based methods necessary to reduce the complexity of parametrization in the case of auction control logic, I have found instances where it makes sense to allow rule-based definition of bid admissibility requirements. In addition to the parametrized bid rules provided by the AB3D scripting language, users may use the `bidRule` statement as a functional method of defining bidding requirements. For example an auction may allow only a single bid per user in a given round. To achieve this, a script may include the bidder attribute `hasbid` which is set to 1 on valid bid submission, along with the following bid rule: `bidRule(hasbid(bidder)=0)`.

2.4 Auction Engine Architecture

Figure 2.3 depicts the main functional components of the Auction Engine, along with the information flows among them. The Auction Engine interprets the auction script, modifying the behavior of the Message Processor and Order Book according to specified parameters and rules. The Message Queue serves to synchronize bids and auction events (as well as information queries in some domains), ensuring that bids are processed in the order received, and in the correct temporal relation to scheduled actions such as price quotes and clears. Information passes to agents through the system cache, designed to conserve communication bandwidth at the primary message processor by routinely pushing commonly needed data. For domains where the pushing of all quote information would be prohibitive (either due to computational or bandwidth limitations), queries allow agents to request that some subset of the auction quote information be calculated and sent to the system cache.

On launch of an auction script, the Order Book and Message Processor are initialized

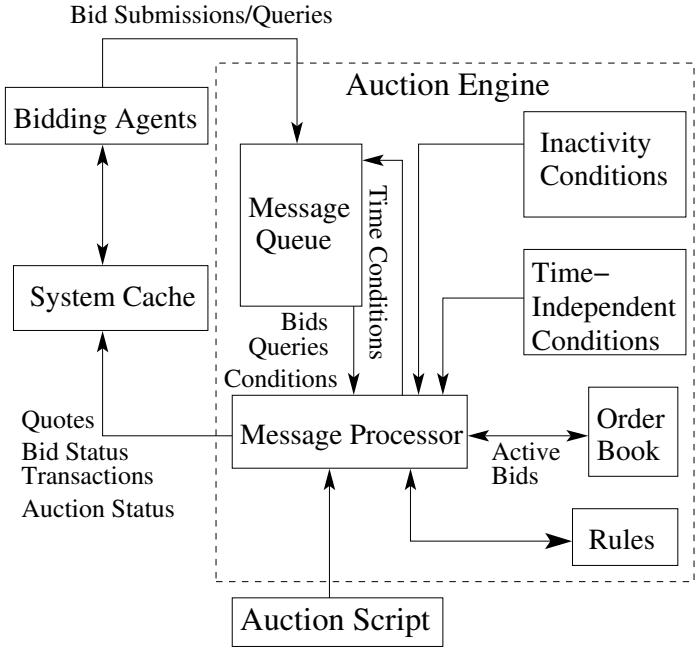

Figure 2.3: Auction engine architecture.

according to parameter settings. The interpreter parses the rules, generating data structures employed by the condition matching process. To ensure timely triggering of rules when their conditions become satisfied, I provide special recognition procedures for three categories of conditions:

1. *temporal* predicates,
2. *inactivity* properties, and
3. *nontemporal* conditions.

Each category requires its own methods for detection based on the ways in which its conditions can become satisfied.

Conditions involving temporal predicates are organized in the same queue employed to process bids and queries. On initial reading of the auction script, the interpreter calculates the earliest time that each such condition may become true. A corresponding message object timestamped with this time is inserted into the queue, as illustrated in Figure 2.4. The Message Processor continually monitors the queue, processing the earliest-timestamped message in turn, with ties broken arbitrarily, as long as this timestamp precedes the absolute clock time.

To process a bid message, the AB3D engine verifies that the agent's bid is admissible according to current policies (specified in parameters and bid rules), and if so, updates objects representing the agent's bid in the active order book. Outcomes of bid processing (i.e., admittance or rejection notices) are transmitted to the bidder through the system cache.

To process a rule-condition message, the engine evaluates whether the associated con-

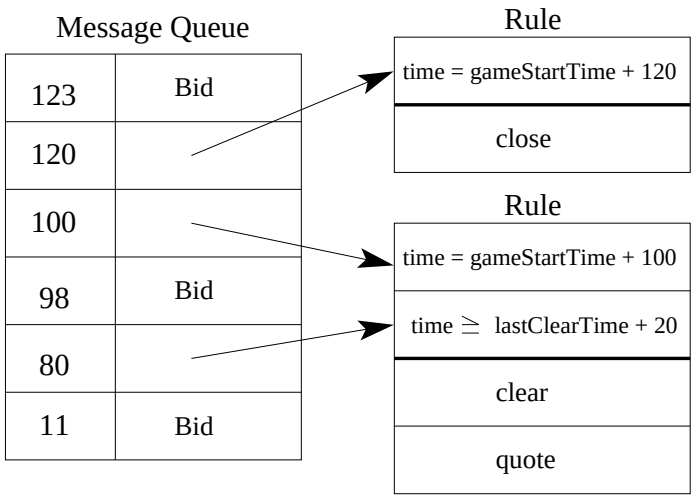

Figure 2.4: The Message Queue sequences bids and potential auction trigger events.

dition is currently true. If so, it checks the status of other conditions of the rule (if any), and if all hold, executes the associated rule actions. Otherwise, it recalculates the earliest time at which each unsatisfied condition may become true, and inserts corresponding messages with the respective timestamps.

Conditions dependent on inactivity (e.g., the length of time since the last admitted bid) are maintained in separate queues within the Message Queue, one for each inactivity property, with predicates ordered in ascending values of inactivity. A pointer in each queue tracks the condition with shortest inactivity that has not expired. The inactivity period of the next unexpired inactivity condition is added to the timestamp of the last activity time (e.g., the time of the last admitted bid), producing a scheduled processing time. The entire rule associated with an inactivity condition will be delivered to the Message Processor if this timestamp precedes the timstamp of the next message with absolute timestamp. For example, in Figure 2.4 the predicate

$$\texttt{time} \geq \texttt{lastClearTime} + 20$$

is scheduled to be processed at time 80, indicating that the last clear operation was performed at time 60.

To handle nontemporal conditions, the system maintains an index of referenced nontemporal state variables, and checks associated rule conditions whenever these are modified.

2.5 Bidding Languages

The extension of AB3D to support a new bidding language requires the implementation of new clearing and information feedback algorithms, bid processing rules, as well as a standardized XML representation for communication with the system. To date three such bidding languages have been implemented for the AB3D system.

2.5.1 Divisible Price-Quantity Bids

The first language allows offers expressing price-quantity correspondences for a single homogeneous good. The unit prices must be monotone in quantity, which is assumed divisible. For example, expressed as a sequence of (price, quantity) pairs, the buy offer

$$(2,2),(3,2)(5,1)$$

represents an offer to buy 1 unit at a price of 5, up to 3 units at a price of 3, and up to 5 units at a price of 2.

The information feedback and clearing functions supporting this language employ the *4-heap* algorithm originally developed for AuctionBot (Wurman et al., 1998a), which supports bid insertion and deletion operations with complexity that is logarithmic in the number of offers. Clearing and information feedback require constant time with this algorithm, as the computation is amortized over bid insertions.

2.5.2 Indivisible Bids

The second language supports *indivisible offers*, meaning that bidders only accept the trades of quantity equal to their offered quantity. Bids express a schedule of (quantity, payment) pairs, where the quantity is indivisible, and the payment signifies a total payment for the offer. For example, the buy offer

$$(4,2)(3,1)$$

represents an offer to buy exactly 1 unit for a total payment of 3, or exactly 2 units for a total payment of 4. This bidding language is fully expressive with respect to multi-unit valuations for a single homogenous good.

Clearing and information feedback for this bidding language is enabled by an incremental knapsack algorithm (Schvartzman and Wellman, 2007). If information feedback is

not enabled, bid insertion with this algorithm has complexity $\mathcal{O}(C^2 \log(N))$, where C is the maximum quantity an offer may express, and N is the number of bids. With information feedback enabled, bid insertion has complexity $\mathcal{O}(C^2 N \log(N))$. In both cases, clearing and quote computation take constant time, as the computation is amortized over bid insertions.

2.5.3 Multiattribute Bids

My multiattribute bidding language extension was designed around the polynomial clearing algorithm that I present in Section 3.2, which separates the clearing problem into bilateral matching and subsequent global optimization (Engel et al., 2006). The critical feature of this algorithm is that the form of the bids affect the complexity only of the bilateral matching. The global match assembles these bilateral matches into an overall optimal set of exchanges. I refer to the matching step applied to a single buyer-seller pair as the *Multiattribute Matching Problem* (*MMP*), the result of which is a *match*, designating the surplus and quantity of the best bilateral trade between these bidders.

More specifically, separation of the clearing process requires that bids comprise a bounded number of independent offers, each defining a mapping between attribute vectors (i.e., configurations) and reserve prices, with a single maximum quantity for each offer. Each offer then consists of a unit pricing function defined over configurations and a maximum quantity of units.

Information feedback for multiattribute auctions in AB3D is provided by the information feedback algorithm presented in Section 3.3.2. This algorithm requires an initial computation of complexity $\mathcal{O}(N^3)$, where N is the number of independent offers in the order book, and subsequently can compute a quote for any single configuration with complexity linear in the number of offers. This complexity result naturally precludes computing quotes for all configurations when goods have large or continuous attribute domains. AB3D therefore features a query-based information feedback system, by which the quotes for single configurations are computed on-demand in response to bidder requests.

Acknowledging the potential for many different forms of unit pricing functions within the context of this clearing algorithm, I designed the multiattribute order book to admit the general class of offers supporting the *MMP* operation. I refer to that implementation as an `MMPable` interface. To qualify as `MMPable`, I require that a single quantity be defined for each offer, and that calling the `mmp` operation of one offer on another will produce a match defining the surplus, quantity, and configuration of the maximum surplus bilateral trade. The multiattribute order book will thus support any new class of multiattribute offer implementing the `MMPable` interface. I have implemented two such classes of of-

fers, one associating each configuration with a unique ID, and one allowing for a factored multiattribute representation.

Enumerated Multiattribute Bid

The enumerated bidding language associates a unique ID with each configuration, making the assumption that goods are defined by a single integer-valued attribute. This formulation supports a fast bilateral matching procedure, as the reserve price for each configuration can be found with a hash table lookup. The optimal bilateral match can be performed in time linear in the number of configurations, by computing the bilateral surplus of matching on each configuration.

Factored Multiattribute Bid

The factored representation sacrifices matching simplicity for a richer multiattribute bidding language. In this bidding language, a configuration is specified by an XML element with tag `CFG`, having multiple child elements which define attribute values (each with tag `A`). Attributes must be defined as one of several types by setting a `T` parameter within the attribute element. The supported types are integer (`T = "I"`), string (`T = "S"`), and float (`T = "F"`).

In order to map each attribute back to some real feature of a good, each attribute element must define a unique attribute ID. These attribute IDs must be defined and communicated to bidders, so that their offers carry a shared semantics. Each attribute element may specify one or more acceptable values (where these values comprise an indifference set). Attribute values are specified in point form with a `V` tag:

```
<A ID="3" T="I">
  <V>2</V>
</A>.
```

An upper and lower value may be used to specify an indifference range when enclosed with an `R` tag:

```
<A ID="1" T="F">
   <R>
     <V>9.1</V>
     <V>21.0</V>
   </R>
</A>.
```

Values may also be enumerated if enclosed with an E tag:

```
<A ID="4" T="S">
  <E>
    <V>Red</V>
    <V>Green</V>
    <V>Blue</V>
  </E>
</A>
```

In the following XML fragment, I present an offer to buy used cars, taking a simplified example where the buyer is concerned only with the make and the year of the car. I assume that attribute ID 1 specifies the manufacturer, and attribute ID 2 specifies the production year. The following XML constitutes an offer to buy up to 2 cars, paying $1758 for Ford models manufactured between 1996 and 2007, and paying $2156 for Chevy models manufactured between 2002 and 2007:

```
<OS>
  <Q>2</Q>
  <O>
    <P>1758.0</P>
    <CFG>
      <A ID="1" T="2"><V>Ford</V></A>
      <A ID="2" T="1"><R><V>1996</V><V>2007</V></R></A>
    </CFG>
  </O>
  <O><P>2156.0 </P>
    <CFG>
      <A ID="1" T="2"><V>Chevy</V></A>
      <A ID="2" T="1"><R><V>2002</V><V>2007</V></R></A>
    </CFG>
  </O>
</OS>
```

The factored multiattribute bid representation induces a bilateral matching procedure that has complexity $\mathcal{O}(C_1C_2)$, where C_1 and C_2 designate the number of offers contained in each offer set.

2.6 Market Game Specification

A *game* is a description of strategic interaction among self-interested agents, defining allowable agent actions, as well as payoffs as a function of those actions. A *market game*

is simply a game in which interaction among agents is mediated by one or more auction mechanisms. Many market games are *Bayesian* games in which agent preferences are specified *probabilistically*, that is, agent preferences are unknown ex ante and drawn from a known distribution for each game instance.

In a market game, each participating agent seeks to maximize an objective, such as trading profits. In more complex games, agents may have objective functions that depend on the time-dependent completion of *tasks*, which require the acquisition of goods through trading. Task dependencies, dynamically arriving task assignments, and nontrivial resource consumption needs may add further complexity to agent objective functions.

2.6.1 AB3D Game Description Language

To facilitate the potentially complex task of defining and simulating a market game, the AB3D system includes a higher-level *game description language* (GDL), which supports the definition of complex preference information, including simple probabilistic constructs for the specification of Bayesian games.

GDL also supports the specification of auctions operating as part of a game. Many games feature sets of similar auctions with only minor parametric differences (e.g., closing times or clearing intervals). Similarly, common auction types occur in many different games with only minor variation (e.g., periodic call markets). To avoid the repeated specification of nearly identical auctions, AB3D supports the use of auction *templates*, which designate some parameters by keyword. These auction parameters may then be defined independently with GDL. The use of GDL for parameter specification also allows for the dynamic generation of auction parameters, allowing for probabilistic parameter settings (e.g., random closing times).

To help ground the presentation of GDL, in the following sections I provide examples from a task allocation scenario developed by Cheng et al. (2004) originally for an information-collection domain. In this scenario, agents accrue value by performing tasks, some of which are assigned at the start of the game, and some of which arrive dynamically throughout the game. Each task requires the acquisition of heterogeneous resources from a set of independent auctions, which are identical in structure to the two-phase auction presented in Figure 2.2. A set of simultaneous ascending auctions (SAAs) (Cramton, 1998; Ausubel and Milgrom, 2002) provides the initial allocation of resources, while a CDA aftermarket handles reallocation as dynamically assigned tasks modify agent objective functions.

2.6.2 GDL Operation

In the AB3D system, auction scripts and agent preference information are represented internally as XML documents. The XML-based auction definitions are provided to the auction engines at runtime to implement specific auction types. Similarly, the XML representations of agent preference information are sent to agents during the game (possibly throughout the game) so that they know their respective objective functions.

The AB3D game description language builds upon the simple syntax of XML, adding special processing commands which allow for the programmatic specification of iterative and hierarchical preference structures, the dynamic generation of probabilistic auction and preference data, as well as pattern-based value generation for string composition and simple arithmetic operations.

Absent any *commands* or defined *variables*, the GDL parser will simply reproduce its XML input for delivery to agents or auction engines. The use of commands and variables will cause the GDL parser to apply a transformation to the XML input in producing auction scripts and agent preferences. GDL commands are invoked by the use of the attribute `cmd` within an XML element, where required command *parameters* are either specified through additional attributes or via child elements. Variable substitution is invoked through the use of a defined variable, either within element content or as a command parameter.

2.6.3 Variable Substitution

The parser maintains both a global and a local variable *hash* which are used for variable substitution during processing. The scope of the global hash is universal, meaning that it is available throughout the game generation process, whereas the scope of the local hash is limited to a single agent or auction.

Variables are placed in the local hash with the **declare** command. Command parameters (presented as child elements) must define the `NAME` and `VALUE` of the entry. Note that the content of the `NAME` and `VALUE` elements can also be generated by GDL commands.

If element content contains text matching a variable in the global hash, the text will be replaced by the value of the variable. For convenience, some commonly used values are inserted into the global hash by the parser, including the game ID, path to game data, and the start and end times of the game. In addition to these common values, all parameters specified in a main game definition file will be inserted into the global hash.

Variable substitution for command parameters is part of a specialized processing sequence for non-numeric parameter data. When evaluating a command parameter, the parser

first searches the local and global hashes, respectively, using the parameter text as a key. If no matching variable is found, the parser attempts to process the content as an arithmetic expression and return the value, ultimately returning an error value if no conversion is possible.

2.6.4 Value Generation

In addition to the **declare** command, GDL includes commands for numeric and string value generation, which can be used both to generate element text content and to define the names or values of local variables.

The **pattern** command is based on Java-style string formatting, used to compose a composite expression or string by inserting various values into a pattern. A `format` attribute specifies the string format. A `type` attribute dictates whether the string itself is outputted as is (if defined as `string`), or evaluated and the result outputted (if defined as `value`). For example, the following rule for the task allocation scenario employs the pattern command with the `string` attribute:

```
<trigger>
  <when cmd="pattern" format="time >= {0} AND validBid" type="string">
    <arg index="0">phaseOneEndTime</arg>
  </when>
  <action>clear</action>
  <action>quote</action>
</trigger>
```

The above GDL code will generate a `when` clause for the auction script, dictating that after `phaseOneEndTime` (a variable stored in the local hash) the auction will clear and quote on any new bid, effectively implementing a CDA aftermarket.

Random values can be generated with the **distribution** command. A "distribution" attribute is additionally required to specify the name of the distribution. Currently only the `uniform` distribution is implemented, which requires a lower and upper bound as parameters. In the following example I set the local variable `phaseOneEndTime` to a random value, using the **pattern** command with a `value` attribute to specify an absolute time relative to the game start time:

```
<CMD cmd="declare">
  <NAME>phaseOneEndTime</NAME>
  <VALUE cmd="distribution" distribution="UNIFORM">
```

```
    <params>
      <param index="1">GameStartTime</param>
      <param index="2" cmd="pattern" format="{0}+100" type="value">
        <arg index="0">GameStartTime</arg>
      </param>
    </params>
  </VALUE>
</CMD>
```

The above script would have inserted a value for `phaseOneEndTime` into the local variable hash, with the value taken from the discrete uniform distribution $[\text{GameStartTime}, \text{GameStartTime} + 100]$. Note that I use the reserved `CMD` tag in the above example, for which the tag itself will not be reproduced as output.

2.6.5 Iterative Content Generation

To facilitate the definition of sets of variables, goods, or markets that vary parametrically, the GDL parser includes a looping construct, invoked with the **for** command. The **for** command takes a `var` attribute to name a looping variable (which is inserted into the local hash), and will iterate over an integer range defined by `from` and `to` attributes. All child content within a **for** command is processed once for each loop iteration.

In the following example, the **for** command is used to define the auctions for 2 unique resources, which are defined by a *type* and *subtype*. These resources need to be linked to their respective auction IDs so that agents know where to submit their bids. In this example, the **pattern** command is used to generate an auction ID for each good that is a function of both the local variable `TYPE` and the loop variable `X`.

```
<AuctionTuple cmd = "for" from="1" to="2" var="X">
  <Type>TYPE</Type>
    <subtype>X</subtype>
    <AuctionID cmd = "pattern" format="{0}*6+{1}" type="value">
      <arg index="0">TYPE</arg>
      <arg index="1">X</arg>
    </AuctionID>
</AuctionTuple>
```

Assuming a value of 3 for the `TYPE` variable, the XML output from the above GDL will be:

```
<AuctionTuple>
   <Type>3</Type>
```

```
   <subtype>1</subtype>
   <AuctionID>19</AuctionID>
</AuctionTuple>
<AuctionTuple>
   <Type>3</Type>
   <subtype>2</subtype>
   <AuctionID>20</AuctionID>
</AuctionTuple>
```

2.6.6 Dynamic Game Information

In dynamic Bayesian games, the "moves" of nature are often revealed to agents throughout the course of the game. GDL supports the dynamic provision of data to agents through the use of a dynamic preference file. Preference information specified in this file with an additional `time` element (with text content specifying an absolute time) will not be made available to agents until the specified time.

In the following example from the task allocation scenario, the time parameter is used to assign agent tasks dynamically at one of several predefined slots after the start of the game, each spaced `msPerSlot` milliseconds apart:

```
<task>
  <agentID>AGENTID-CONTENT</agentID>
  <resources>RESOURCE-CONTENT</resources>
  <value>VALUE-CONTENT</value>
  <time cmd = "pattern" format="{0}+{1}*{2}" type="value">
    <arg index="0">GAME_START_TIME</arg>
    <arg index="1">taskArrivalSlot</arg>
    <arg index="2">msPerSlot</arg>
  </time>
</task>
```

2.6.7 Externally Generated Content

To support the generation of content by means outside of the currently implemented GDL command constructs, the **extern** command will invoke a user-supplied Java class (defined by a "type" parameter) and include all output under the invoking element. The following example uses an external PoissonDistribution class for value generation:

```
<AuctionCloseTime cmd="extern" type="MyPoissonDistribution" />
```

2.7 AB3D-Supported Research

As testament to the utility of the AB3D platform, members of my research group have employed AB3D in support of a diverse set of market-based research efforts. Past research topics include the evaluation of alternative information feedback policies in a manufacturing domain with nonconvex preferences (Schvartzman and Wellman, 2007), as well as the task allocation scenario used to present GDL above (Cheng et al., 2004).

The AB3D system also supports the multiattribute simulations of Chapters 4 and 5. In Section 5.4 I describe the integration AB3D into the Trading Agent Competition Supply Management Game. As I describe in greater detail in Section 5.4, I replaced the existing negotiation protocol of the simulation environment with a multiattribute market. Integrating AB3D into TAC SCM required a simple proxy to translate and route offers from the simulation through the AB3D system. The AB3D system computes transactions based on the submitted offers, which are subsequently sent back to the simulation environment.

Integration of AB3D into TAC SCM required the use of one additional AB3D feature not previously discussed: an agent authenticated as an administrator (with the appropriate password) is permitted to send rule-messages to an auction via the Agent Manager interface. The TAC SCM game simulates a period of 220 days (with a wall-clock time of 15 seconds per day), where the manufacturer/customer negotiation must be performed each day. The synchronization of AB3D clear operations with the TAC SCM simulation is handled by the use of clearing rules, sent dynamically from the SCM simulation and conditioned on absolute time.

AB3D is also currently being used to simulate commodity markets for the training of human traders at Singapore Management University, using a domain-specific user interface. The goal of this research is twofold: train a group of humans to be successful commodity traders, and study their behavior for the development of automated commodity trading agents.

In contrast to financial trading simulations which link simulations to real-world markets, this simulation environment controls the evolution of trading scenarios in an event-driven way. In this domain, AB3D support for the dynamic provision of preference information allows for the rapid setup of a variety of experiment scenarios, by which it is possible to collect a wide range of information on human traders' responses and behaviors.

While still in early stages of experimentation, the initial results from this project are already quite encouraging. The first iteration of experiments shows that traders who have better qualitative understanding of disseminated information, and who can react swiftly to market events, indeed outperform those who are slower and less acute (Cheng, 2007).

2.8 Conclusion

The AB3D system provides a flexible platform for operating market games, combining a rule-based scripting language for the specification of auction mechanisms, and a structured game description language for the specification and implementation of multi-auction market games.

The increasing sophistication of auction mechanisms, coupled with further development in automated bidding agents, demands a structured method of auction representation. The AB3D scripting language provides a flexible medium for specifying a broad range of auction mechanisms. Its main innovation is rule-based invocation of auction events and policy revisions, supporting dynamically flexible auction behavior, while achieving many advantages of formal auction specification in an expressively convenient environment.

Chapter 3

Iterative Multiattribute Call Market Algorithms

The previous chapter introduced the AB3D market game platform, and described AB3D support for multiattribute auctions. This chapter details the algorithms which underlie my implementation of multiattribute auctions in AB3D.

I first present results from joint work with Engel and Wellman that identified restrictions on the form of multiattribute offers that reduce the complexity of clearing. These restrictions allow the clearing process to be decomposed into two discrete steps: bilateral matching, i.e., finding optimal pairwise trades, and subsequent network optimization based on the bilateral surpluses of these trades. I outline the operation of this algorithm for a form of multiattribute offer consistent with the multiattribute bidding languages implemented in AB3D. I give simulation results for this algorithm demonstrating its scalability with respect to bids and offers.

My main contribution in this chapter is a novel approach for computing quote information, which employs the same decomposition result as our clearing algorithm. I first present an example demonstrating the complexities of computing quote information given multiattribute bids. I then describe how the decomposition result used by our clearing algorithm may be employed to simplify the task of computing quotes. My approach operates in somewhat reverse fashion with respect to the clearing algorithm, first introducing a hypothetical new offer, and computing bilateral surplus values that would include this new offer in the optimal match. I present a polynomial algorithm for computing bilateral surplus values for quotes, and show that given these values, I can compute the quote for any single configuration with complexity linear in the number of offers.

3.1 Bidding Language

In this section I present a multiattribute bidding language admitting polynomial-time clear and quote operations. As discussed, the goods are assumed to be defined by a set of *attributes*, where a given assignment of attributes defines a *configuration*. The most simple multiattribute bidding unit expresses a maximum/minimum price at which to trade a given quantity of a single configuration.

Definition 1 (Multiattribute Point) *A* multiattribute point *of the form* (x, p, q) *indicates a willingness to buy up to a total quantity q of configuration x at a unit price no greater than p (for* $q > 0$*). Similarly, negative quantity (*$q < 0$*) would indicate a willingness to sell up to q units at a price no less than p.*

Participants in multiattribute auctions often wish to buy or sell one of several alternative configurations. This would happen, for example, if a buyer wishes to procure computers, and is willing to accept multiple alternatives with respect to attributes such as processor type/speed, memory type/size/speed, etc., but has configuration-dependent reserve prices. Conversely, sellers may be able to supply one of multiple PC alternatives at varying reserve prices. The following bid type supports the expression of this type of preference.

Definition 2 (Multiattribute *XR* Unit) *A* multiattribute *XR* unit *is a triple* $(configurations, prices, quantity)$ *of the form:* $((x_1, x_2, \ldots, x_N), (p_1, p_2, \ldots, p_N), q)$ *indicating a willingness to trade any combination of configurations* $(x_1, x_2, \ldots, x_N)$ *at respective unit prices* $(p_1, p_2, \ldots, p_N)$ *up to total quantity* $|q|$*, where* $q > 0$ *indicates a buy offer, and* $q < 0$ *indicates a sell offer.*

Defining the price of configurations not enumerated in the offer to be zero for buy offers and infinite for sell offers, the *XR* unit with positive (negative) quantity expresses a willingness to accept (provide) any allocation of total quantity not greater than $|q|$, given that the total payment is not greater than (less than) the sum of the unit prices expressed in the *XR* unit. For example, given *XR* unit $((x_1, x_2, x_3), (p_1, p_2, p_3), 4)$ the allocation $\{x_1, x_1, x_2\}$ would be acceptable at total payment not greater than $p_1 + p_1 + p_2$. The name *XR* derives from the fact that an *XR* unit with quantity 1 expresses an exclusive-or preference over a set of acceptable configurations.

In a slight abuse of notation, I define the *r* operator over an *XR* unit and a configuration to denote the reserve price for the given configuration in the *XR* unit, i.e., $r(XR, x) = p$ selects the unit reserve price for configuration *x* in the *XR* unit. Note that a multiattribute point is equivalent to an *XR* unit with 1-tuple configurations and prices. To simplify the

syntax of my examples, I use the multiattribute point notation when an *XR* unit defines a reserve price for only a single configuration.

I use a slightly more expressive bidding language in the multiattribute call market implemented here, which is an *OR* extension of the *XR* unit.

Definition 3 (Multiattribute *OXR* Bid) *A* multiattribute *OXR* bid *is a set of multiattribute XR units,* $\{XR_1, XR_2, \ldots, XR_M\}$ *indicating a willingness to trade any combination of configurations such that the aggregate allocation and payments to the bidder can be divided among the XR units such that each* (g, p) *pair is consistent with its respective XR unit.*

3.2 Clearing

Clearing the auction requires finding the global allocation that maximizes the total trade surplus, which is the *Global Multiattribute Allocation Problem* (*GMAP*). For a certain class of bids, which includes *OXR* bids, *GMAP* can be divided into two discrete steps: identifying optimal bilateral matches, i.e., the *Multiattribute Matching Problem* (MMP), and then maximizing total surplus as a function of the optimal bilateral matches.

Joint work with Engel and Wellman (Engel et al., 2006) explored the connection between bidding languages and clearing algorithms for this domain, identifying the *MMP-GMAP* decomposition result under slightly more general conditions. For this result, each offer has to exhibit *configuration parity*, meaning that each offer expresses a single quantity limit that can be met with an arbitrary set of configurations (subject to reserve price constraints). Offers additionally must adhere to one of the following two conditions:

1. *no aggregation*, meaning that the aggregation of quantity across multiple trading partners is disallowed in matching the offer.
2. *divisibility* and *linear pricing*, meaning the offer will allow any quantity up to the offered quantity, with a reserve price that is linear in quantities.

OXR bids meet the latter of these conditions, as each offer defines a divisible total quantity as well as unit prices for individual configurations, allowing quantity aggregation across alternate configurations and trading partners. In the case of *OXR* bids, the multiattribute matching problem determines the optimal configuration x to trade between each pair of buy and sell *XR* units, which is a function of the configuration-dependent reserve prices of the offers. For buy *XR* unit $XR_b = (configs^b, prices^b, q^b)$ and sell *XR* unit $XR_s = (configs^s, prices^s, q^s)$ let $MMP_x(XR_b, XR_s)$ denote the configuration that maximizes

bilateral trade surplus:

$$MMP_x(XR_b, XR_s) \quad = \quad \underset{x \in X}{\operatorname{argmax}}(r(XR_b, x) - r(XR_s, x)).$$

The *MMP surplus*, MMP_s, represents the value for trading a single unit of the optimal configuration:

$$MMP_s(XR_b, XR_s) = \max_{x \in X}(r(XR_b, x) - r(XR_s, x)).$$

Define BX as the set of all XR units contained in the buyers' OXR bids, and SX as the set of all XR units in the sellers' OXR bids. For $b_i = \{XR_{i,1}, XR_{i,2}, \ldots, XR_{i,M}\}$ for buyer $i \in C$ and $b_j = \{XR_{j,1}, XR_{j,2}, \ldots, XR_{j,M}\}$ for seller $j \in S$,

$$BX = \bigcup_{i \in C} \bigcup_{XR_{i,k} \in b_i} XR_{i,k},$$

$$SX = \bigcup_{j \in S} \bigcup_{XR_{j,k} \in b_j} XR_{j,k}.$$

Given the solution to MMP_s between each element in $BX \times SX$, *GMAP* can be formulated as a network flow algorithm, specifically the *transportation problem* (Ahuja et al., 1993), with source nodes SX, sink nodes BX, and link surplus (equivalently, negative link costs) equal to the values of MMP_s on $BX \times SX$. The network flow formulation of *GMAP* is depicted in Figure 3.1.

Once the clearing problem has been formulated as an instance of the transportation problem, the set of trades that maximize trade surplus can be found by solving for the optimal network flow. The solution flow along a given link designates a quantity traded between two agents (whose bids contain the respective XR units), and the configuration to be traded is the surplus-maximizing configuration between the XR units.

The transportation problem can be solved in $\mathcal{O}(n^3)$ time, where n is the number of nodes in the graph (Ahuja et al., 1993). Figure 3.2 plots the clear time in milliseconds as a function of the number of XR units submitted by traders, with the units evenly divided between buy and sell offers.[1] The computations were performed with an Intel Pentium 4 CPU 3.40GHz, with Java heap set to initial/max size of 512MB. As the number of XR units approaches 3000, the total clear time remains below 15 seconds.

[1] These offers can be arbitrarily divided among traders' bids. For example, if traders were restricted to bids comprising a single XR unit, Figure 3.2 would depict the clear time as a function of the number of traders.

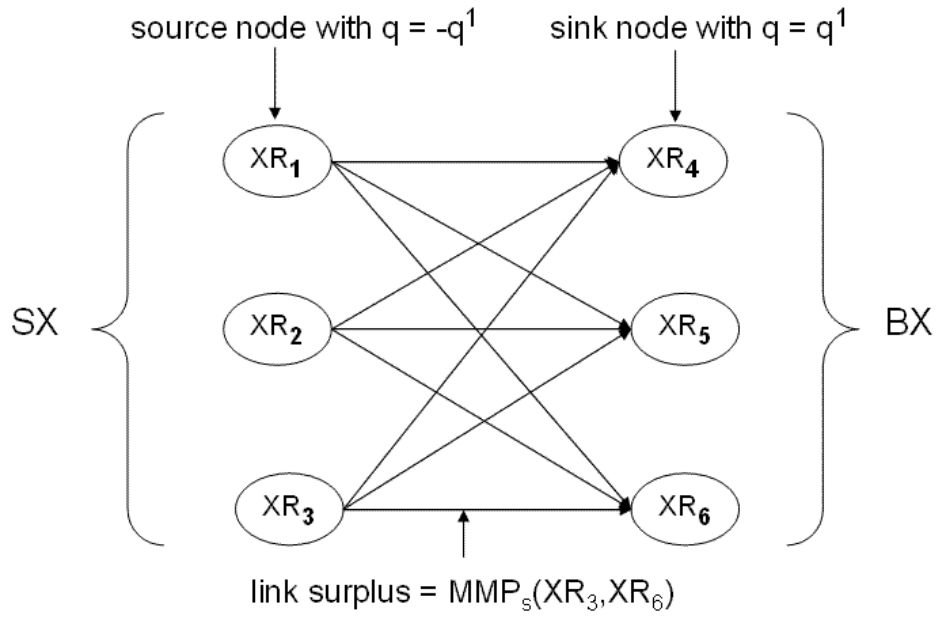

Figure 3.1: *GMAP* network flow formulation for *OXR* bids.

3.3 Information Feedback

Call market implementations in non-multiattribute settings typically provide information feedback in the form of single-unit price quotes. Provided in the form of ask and bid prices, these quotes constitute the limiting prices, respectively, at which a trader could place a winning buy or sell bid for a single unit, given the current state of the order book. These quotes are typically also *anonymous* (Wurman et al., 2001), meaning that the same price quotes are reported to all bidders.

More general approaches to information feedback may provide price quotes on arbitrary allocations, or may provide quotes that are *non-anonymous* (Schvartzman and Wellman, 2007), meaning that a bidder's current standing bid is taken into consideration when computing price quotes. I do not pursue that degree of generalization here, and instead focus on the computation of single-unit configuration-dependent price quotes.[2] For example, for all configurations x, I support the computation of the ask quote for x, defining the limiting price at which a buy offer for 1 unit of configuration x would be included in the hypothetical clear. The equivalent definition holds for sell offers given the bid quote for x.

[2]Anonymous quotes can be computed with the algorithms presented here by iteratively removing each agent's bid from the order book and subsequently computing price quotes. This approach would multiply the algorithmic complexity by the number of bidders.

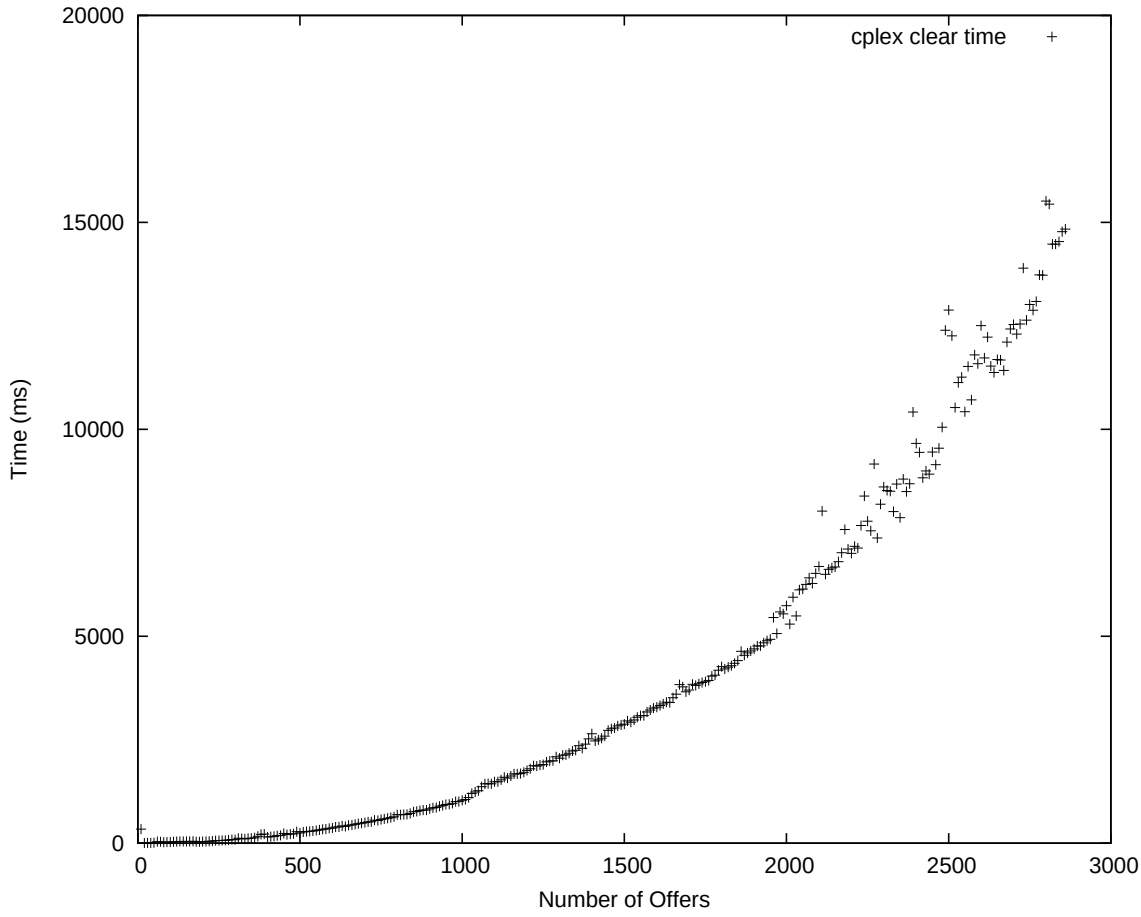

Figure 3.2: Network flow clear time as a function of the number of *XR* units using CPLEX solver.

3.3.1 Example

The following example illustrates how these quotes are calculated, and introduces some of the complexities of calculating quotes. I use $\emptyset$ to indicate that the quote for a given configuration is undefined.

Example 1 *Consider the following sequence of independent bids:*

t	agent	bid	bid quote (x_1,x_2,x_3,x_4)	ask quote (x_1,x_2,x_3,x_4)	trade set
1	1	$((x_1,x_2),(5,5),-1)$	$(\emptyset,\emptyset,\emptyset,\emptyset)$	$(5,5,\emptyset,\emptyset)$	$\{\}$
2	2	$((x_2,x_3),(6,6),1)$	$(\emptyset,5-\delta,5-\delta,\emptyset)$	$(6+\delta,6+\delta,\emptyset,\emptyset)$	$\{(1\leftrightarrow 2)\}$
3	3	$((x_3,x_4),(4,4),-1)$	$(\emptyset,4-\delta,4-\delta,\emptyset)$	$(5,5,5+\delta,5+\delta)$	$\{(2\leftrightarrow 3)\}$

- *Round 1* The only bid is bid 1: a sell offer for x_1 or x_2, which dictates ask prices for these configurations. Quotes are undefined for all configurations with no outstanding bids.

- *Round 2* Bid 2 is submitted, a buy offer for x_2 or x_3, and matches the sell offer on configuration x_2. With the inclusion of bid 2, a new successful buy offer must provide more than 1 unit of surplus on configurations x_1 or x_2, meaning an ask quote of $6+\delta$ for those configurations, where δ is the minimum bid increment. Similarly, a new successful sell offer must offer to trade at a price of $5-\delta$ or less on configurations x_2 or x_3.
- *Round 3* Another sell offer comes in below the bid quote on configuration x_3, placing it in the new matching set. A new sell offer must beat the surplus of the current trade for the lone buy offer, resulting in a bid price of $4-\delta$ for x_3 and x_4. A buy offer for configuration x_1 or x_2 must only equal the price of the no longer matching sell offer. For configurations x_3 or x_4, a new buy offer does not need to exceed the surplus of the existing trade (which would imply an ask price of $6+\delta$) because bid 2 may also be matched with bid 1. The total surplus lost from displacing the current trade is decreased by re-matching the trade with the remaining offers. In this case, a bid of $5+\delta$ would be sufficient to increase the total surplus.

3.3.2 Information Feedback under the *MMP-GMAP* Decomposition

Just as it reduces the complexity of clearing, the decomposition of *GMAP* into *MMP* and subsequent network optimization can also reduce the complexity of computing quote information. Recall that the single-unit quote defines a limit on the price at which a single-unit offer for a specific configuration will be included in the maximally efficient set of trades. *MMP* computes bilateral surpluses with offers in the order book as a function of the configuration-specific offer prices, and it is the bilateral surpluses which determine whether the trade is included in the maximally efficient set.

A quote for a given configuration can be found by first finding the required bilateral surplus (i.e., solution to MMP_s) for each existing offer such that a bilateral trade with that offer is included in the efficient set. The configuration-specific price at which a bilateral trade (with a specific offer) would be included in the efficient set can then be computed from the required bilateral trade surplus and the configuration-specific offer price. The computed price will be the quote for a $(\textit{configuration}, \textit{trader})$ pair; taking the min/max over all sellers/buyers yields the ask/bid quote for a configuration.

This process is best described through example. Figure 3.3 depicts the *GMAP* formulation for a set of 3 sell offers (shown at left) and 3 buy offers (shown at right). The solutions to MMP_s are indicated on the links connecting pairs of *XR* units. The solution to *GMAP* is indicated by the bold links (with trade quantity of 1 on each bold link), where the optimal solution has XR_5 and XR_2 trading 1 unit of x_2, and XR_3 and XR_6 trading 1 unit of x_2.

Now consider calculating the bid quote for x_2 given the set of bids from Figure 3.3. As depicted in Figure 3.4, we first add a dummy node (XR_D) to the graph and connect that

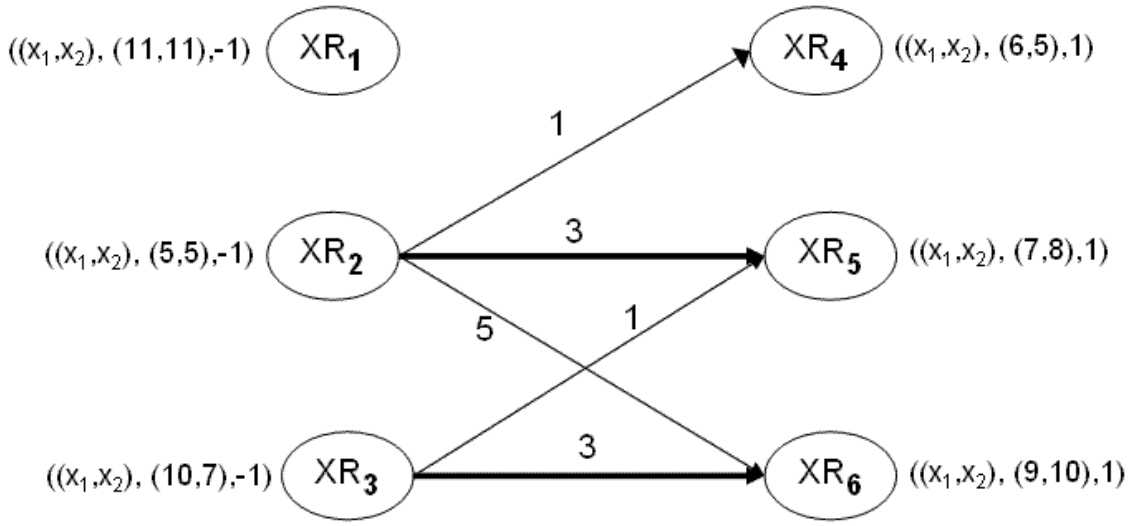

Figure 3.3: *GMAP* formulation with 3 buy offers and 3 sell offers. The optimal solution is indicated in bold.

node to one of the buy nodes (node XR_6 in Figure 3.4). We now calculate the minimum link surplus on the new edge that would increase the value of the optimal network flow. The computed link quote, LQ_6, is the trade surplus (i.e., solution to $MMP_s(XR_6, XR_D)$) required for a new bid to trade with node XR_6. The link quote for each buy node must be calculated, producing a link quote for each $XR_k \in BX$.

The bid quote for a given configuration x is then:

$$\max_k (r(XR_k, x) - LQ_k).$$

Continuing with the example, Figure 3.5 depicts the computed link quotes as well as the buy *XR* units. I present algorithms for computing these link quotes in Section 3.3.3, but for now take as given that these link quotes have been computed. In this instance, the bid quote for configuration x_2 with XR_6 would be the offered price of 10, less the required link surplus of 4, producing a quote of 6 to transact *with that unit*. The bid quote for the configuration is the maximum over all the units, which is also 6. An offer price of 6 for x_2 would be sufficient to trade with either XR_6 or XR_5, as the quoted price for XR_5 would also be 6 (with a reserve price of 8 and required link surplus of 2).

Finally, as confirmation that this process has produced a valid quote, we can consider the outcome of a new sell offer for a unit of x_2 at the quoted price. Figure 3.6 depicts this situation for the case that the new bid transacts with XR_5 (the algorithm will break the tie randomly) and shows that inclusion of the new bid increases the trade surplus by 1 to a

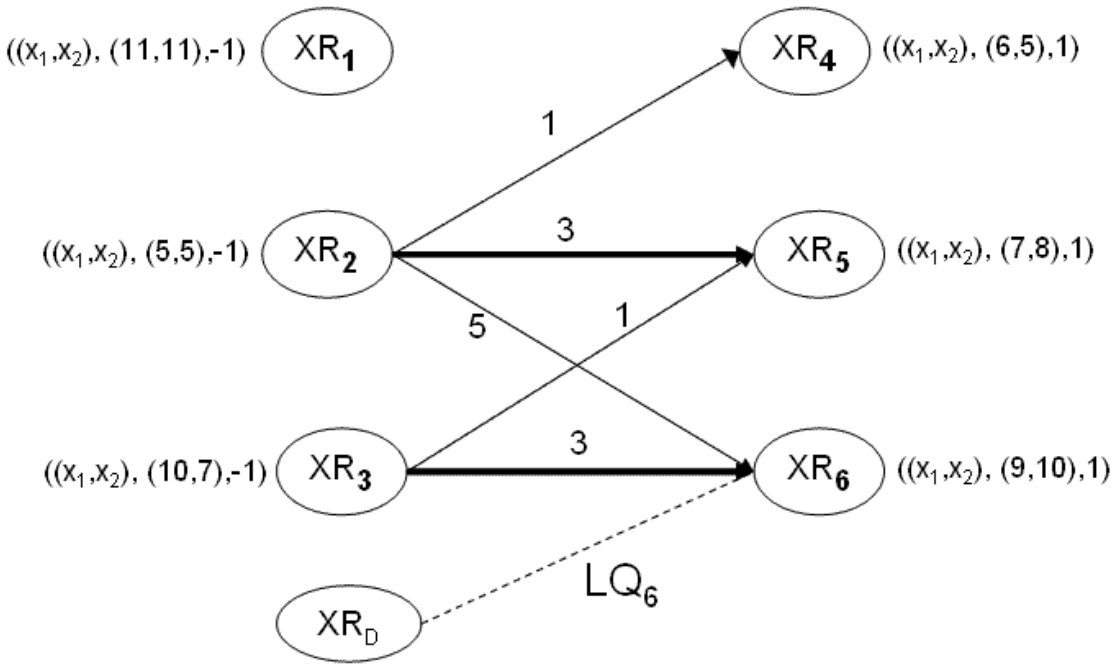

Figure 3.4: *GMAP* formulation of Figure 3.3 with dummy node added for computing a link quote.

total of 7.

It should be apparent from the *max* operation that once all link values have been determined, configuration quotes can be computed with complexity on the order of the number of *XR* units. This implies that the complexity of computing a single quote is invariant to the size of attribute space for a fixed number of *XR* units when the *MMP-GMAP* decomposition is applicable.

There may be instances where the number of configurations is large enough to preclude computing and pushing all configuration quotes. For example, in the case of continuous attributes, or even large numbers of attributes (which induce an exponential number of configurations), a full push of quote information is infeasible. In these instances it may become necessary to limit the number of configurations for which traders have access to price quote information. A query-based system is implemented in AB3D, whereby agents poll the system for quotes on specific configurations, thus limiting the computational and bandwidth usage of information feedback.

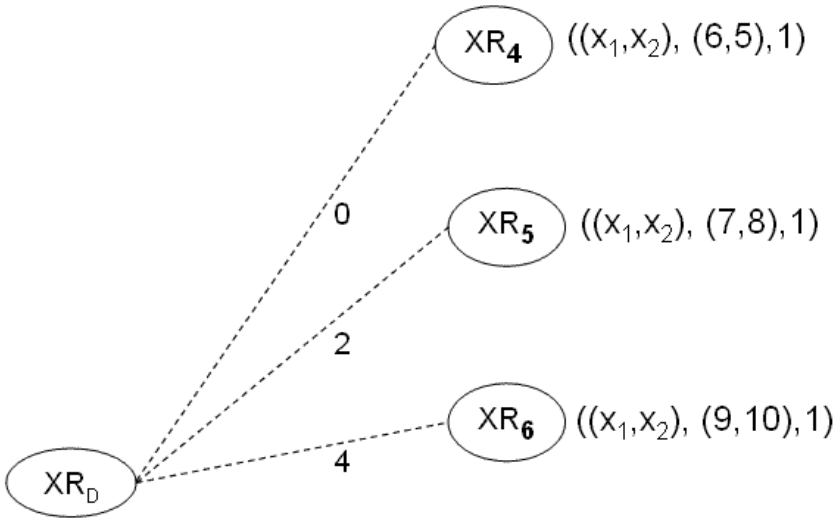

Figure 3.5: Link quotes computed for a bid quote given the *GMAP* formulation of Figure 3.3.

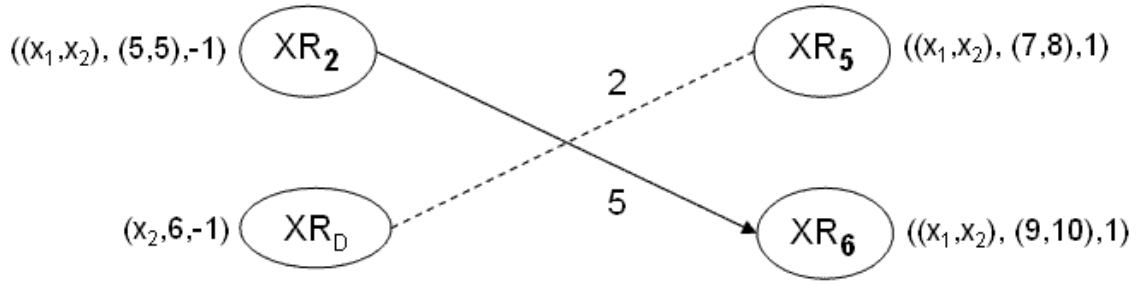

Figure 3.6: *GMAP* solution for Figure 3.3 with a new sell offer at the quoted price.

3.3.3 Link Quote Algorithms for the *MMP-GMAP* Decomposition

In Section 3.3.2, I showed that the *MMP-GMAP* decomposition can be used to decompose the process of computing quote information into computation of link quotes for all *XR* units, and subsequent computation of a specific configuration quote based on the computed link quotes. In this section, I present algorithms for computing link quotes, as well as empirical run time results for the algorithms presented.

Iteration of the Clearing Algorithm

To compute quote information, it is necessary to find an offer price for a new bid that would increase either the surplus or trading volume of the optimal allocation. For example, the

ask quote for a configuration is the price p at which the optimal surplus or trading volume is increased with the inclusion of a bid of $(x, p, 1)$. This description can be translated into an algorithmic procedure, whereby the clearing algorithm is applied once for each *XR* unit in the graph, with a dummy node connected to the respective *XR* unit. The quantity along the link from the dummy node is forced equal to 1, and the surplus of the link is set to zero. Applying the clearing algorithm to this new graph will produce a new trade surplus that is weakly less than the trade surplus of the original graph (given that the new solution includes a zero-surplus flow of one unit). The non-negative reduction in trade surplus with the addition of this zero-surplus link is the desired link quote for that *XR* unit.

Computing all such link quotes requires one iteration of the clearing algorithm for each *XR* unit in the order book, yielding total complexity of $\mathcal{O}(n^4)$, where n is the number of *XR* units. The running time of this approach (which I designate as *brute force*, given the fairly naïve implementation) is presented in Figure 3.15. Run times are presented using both CPLEX (cplex brute) and QSopt (qs brute) linear programming packages.

Sensitivity Analysis

Another approach I investigated for information feedback is based on the use of *sensitivity analysis* of the linear program used for clearing. Once a solution to a linear program has been computed, sensitivity results provide an upper and lower bound on the values of variables for which the current linear-programming basis remains optimal (Ahuja et al., 1993). Quote information can be computed by adding a dummy buy/sell node to the graph along with a link to each seller/buyer before solving the linear program, where the surplus of the link is set to zero. The upper bounds of the link values computed by sensitivity analysis are equal to the link values necessary (i.e., required MMP_s values between the dummy nodes and the respective *XR* units) to change the basis of the optimal solution.

The values computed by this approach are only an approximation, because the basis may change without affecting the solution to the linear program, i.e., matching the computed link value is a necessary but not sufficient condition for inclusion in the optimal global allocation. Although not carrying as clear of a semantics for bidders, this quote does convey information as to the true quote price, and could therefore have merit if it provides significant computational savings. Unfortunately, the computational savings were meager, as shown by the data labeled “cplex sensitivity” in Figure 3.15.

Shortest Path Approach

A more specialized technique I developed for computing link quotes takes the optimal flow computed by the clearing algorithm, and applies a shortest path algorithm to the *residual graph* (Ahuja et al., 1993). Given an optimal flow, the residual graph contains links indicating feasible augmenting flows. For example, assume a *GMAP* network flow graph contains a single buy and sell offer. These *XR* units, XR_1 and XR_2, respectively, each offer a quantity of 5. Further suppose that the link between XR_1 and XR_2 carries 3 units of flow in the optimal solution. The residual graph would have a directed arc with capacity of 3 from XR_2 to XR_1, indicating feasible flow that could *negate* the optimal flow. The residual graph would additionally have a directed arc with capacity 2 from XR_1 to XR_2, indicating additional positive flow that could be sent along this link.

The shortest path approach starts with the observation that in adding a unit of flow from the dummy node to an existing *XR* node, the optimal flow must now be augmented if any node balance constraints are violated (i.e., the quantity constraint of the *XR* unit is violated). There may be several paths by which flow can be augmented through the residual graph such that all node balance constraints are satisfied, each of which is a connected path starting from the quote node. I denote the terminal nodes of all such paths as *absorbing* nodes. The desired link quote is defined by the cost of the minimum-cost augmenting path. If the minimum-cost augmenting path adds a unit of quantity, the desired link quote for the *XR* node is equal to the efficiency loss from the augmenting flow (as the augmented solution would have equal trade surplus and increased trade volume, with respect to the optimal solution). If the augmenting path does not change the total quantity, the desired link quote is equal to the efficiency loss plus the minimum bid increment (as the augmented solution would have equal trade volume with the optimal solution, and therefore the new link would need to *increase* the trade surplus).

The following examples demonstrate how to compute link quote values for a bid quote using the shortest path approach. In these examples, I define the bilateral trade quantity between XR_j and XR_k to be the total quantity of flow between XR_j and XR_k in the optimal solution, which I denote by $TQ_{j,k}$. I similarly define the total trade quantity of node XR_k, to be the total quantity of flow through XR_k in the optimal solution, i.e., $TQ_k = \sum_j TQ_{j,k}$.

Example 2 *Figure 3.7 depicts an optimal flow for a single trade of quantity 2 between the only buy and sell XR nodes in the graph, for total surplus of 4. After adding flow of quantity 1 from the dummy node XR_D to buy node XR_2, the quantity constraint of XR_2 is violated, so the quantity traded between XR_1 and XR_2 must be reduced by 1, i.e., an augmenting flow of 1 is sent from XR_2 to XR_1, reducing the flow along that link. Node XR_1 is an absorbing*

node, as augmenting a unit of flow to XR_1 does not induce a new node balance constraint violation. The maximum-surplus flow in the augmented graph has surplus of 2, resulting in a surplus loss of 2 with the inclusion of unit flow from the dummy node, giving link quote $LQ_2 = 2$.

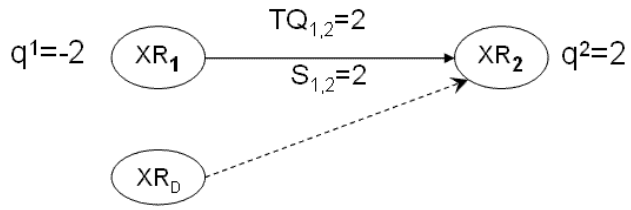

Figure 3.7: Network for Example 2. Adding a unit of flow from XR_D violates the node balance constraint of XR_2.

Example 3 *In the network flow graph of Figure 3.8, the quantity constraint of XR_3 is non-binding at the optimal solution. In this case, node XR_3 is an absorbing node, since augmenting a unit of flow into XR_3 does not violate node balance constraints. Any non-negative link value will therefore be sufficient to include a trade between the dummy node and XR_3, yielding link quote $LQ_3 = 0$.*

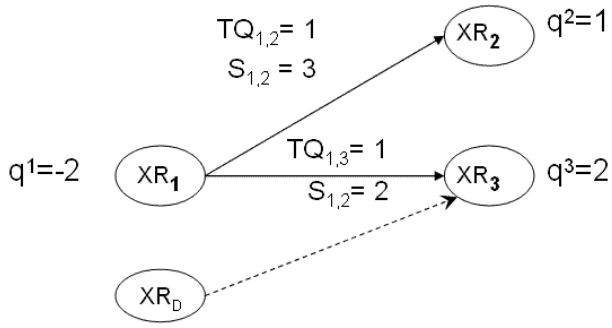

Figure 3.8: Network for Example 3. Adding a unit of flow from XR_D does not induce any node balance constraints.

Example 4 *Figure 3.9 depicts the optimal solution for three XR units, with a non-binding quantity constraint for XR_2, but binding quantity constraints for XR_1 and XR_3. Adding a*

unit of flow into XR_3 will violate its node balance constraint, so flow must be sent along the augmenting path from XR_3 to XR_1 for surplus loss of 2. Again, unit XR_1 is an absorbing node, but so too is node XR_2. In this case, 1 unit of flow can be augmented from XR_1 to XR_2, increasing surplus by 1 for a total efficiency loss of only 1 along the full augmenting path. In this case, the link quote would be 1, as this is the minimum cost augmenting path *to an absorbing node.*

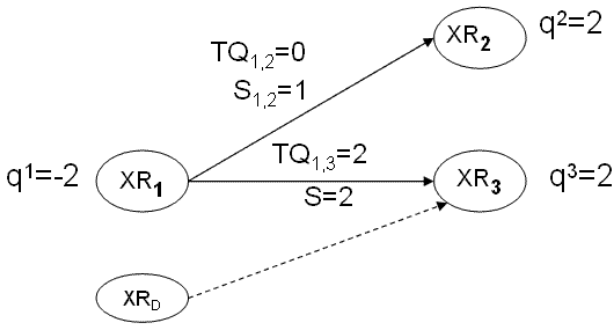

Figure 3.9: Network for Example 4. Adding a unit of flow from XR_D violates the node balance constraint of XR_3. XR_1 and XR_2 are both absorbing nodes, with the minimum-cost path terminating at XR_2.

In computing link quote LQ_j for XR_j, (which is XR_6 in Figure 3.4), a single unit of flow must be augmented through the residual graph from XR_D until all node balance constraints are satisfied, i.e., until we reach an absorbing node. The link quote LQ_j is equal to the minimum-cost augmenting path which satisfies node balance constraints, where positive augmenting flows have negative cost, while negative augmenting flows have positive cost.

Computing all link quotes then simply requires computing the shortest path from each node to an absorbing node, which can be done with the Floyd-Warshall all-pairs shortest path algorithm in $\mathcal{O}(n^3)$ time (Ahuja et al., 1993). The computation time for the shortest path approach is plotted as a function of the number of *XR* units in Figure 3.14, along with the CPLEX clear time for reference. The total time to clear and compute link quotes would be the sum of the two plotted times in Figure 3.14. Again, computations were performed on an Intel Pentium 4 CPU 3.40GHz, with Java heap set to initial/max size of 512MB. The quote time remains manageable at 10 seconds for up to 1000 *XR* units, and then quickly begins to escalate, reaching 70 seconds for 2000 *XR* units. The shortest path technique is compared against alternative information feedback techniques in Figure 3.15. The following example demonstrates how to construct the residual graph and identify absorbing nodes

for a larger problem instance.

Example 5 *The network flow graph shown in Figure 3.10 depicts a* GMAP *network flow formulation, with link values designating the pairwise values of* MMP_s. *The optimal flows for this formulation are shown in Figure 3.11.*

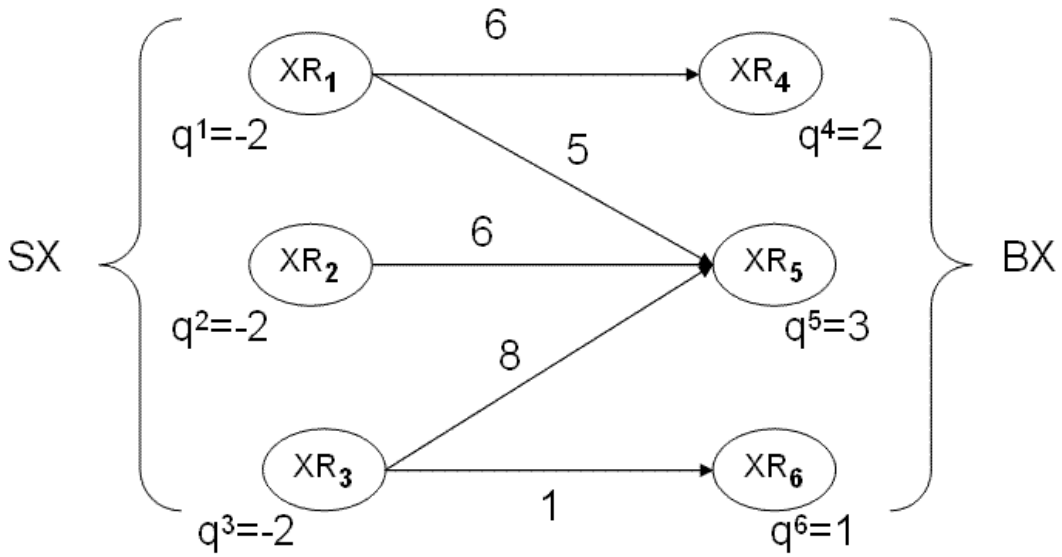

Figure 3.10: *GMAP* formulation for Example 5.

Bid Quote Formulation

If computing LQ_j for $XR_j \in BX$, the augmenting path will reduce trade quantity when traversing from a node in BX to a node in SX, and add trade quantity when traversing from a node in SX to a node in BX. The residual graph therefore has a link from node $XR_j \in BX$ to node $XR_k \in SX$ if $TQ_{jk} > 0$, with link cost equal to $MMP_s(XR_j, XR_k)$. The residual graph will have a link from node $XR_k \in SX$ to node $XR_j \in BX$ if $TQ_{jk} < min(q^j, q^k)$, with negative link cost of $MMP_s(XR_j, XR_k)$.

If $XR_q \in BX$, node balance constraints will be satisfied if the augmenting path from XR_q terminates on a buy node XR_k with $TQ_k < q^k$ or on a sell node XR_k with $TQ_k > 0$.

Figure 3.12 depicts the residual network for computing bid quotes from Figure 3.11, with absorbing nodes shaded black. In this case, the link quote for XR_6 is 0, as it is an ab-

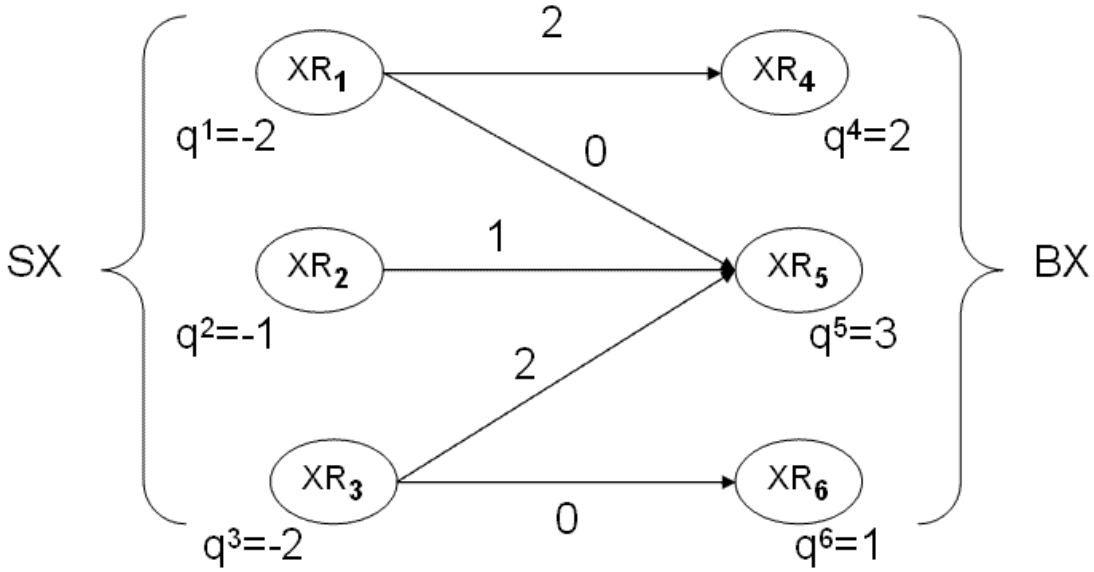

Figure 3.11: Optimal flows for Example 5.

sorbing node. The link quotes for XR_4 and XR_5 are both equal to 6, with nearest absorbing nodes of XR_1 and XR_2, respectively.

Ask Quote Formulation

If $XR_q \in SX$, the augmenting path will reduce trade quantity when traversing from a node in SX to a node in BX, and add trade quantity when traversing from a node in BX to a node in SX. The residual graph therefore has links which are equal in magnitude but opposite in direction to those for the bid quote formulation. The residual graph has a link from node $XR_j \in BX$ to node $XR_k \in SX$ if $TQ_{jk} < min(q^j, q^k)$, with negative link cost equal to $MMP_s(XR_j, XR_k)$. The residual graph will have a link from node $XR_k \in SX$ to node $XR_j \in BX$ if $TQ_{jk} > 0$, with link cost equal to $MMP_s(XR_j, XR_k)$.

If $XR_q \in SX$, node balance constraints will be satisfied if the augmenting path from XR_q terminates on a sell node with $TQ_k < q^k$, or on a buy node with $TQ_k > 0$.

Figure 3.13 depicts the residual network for computing ask quotes from Figure 3.11, with absorbing nodes shaded black. Here the link quote for XR_2 is zero, as it is an absorbing node. The link quotes for XR_1 and XR_3 are 6 and 2, respectively, with nearest absorbing nodes of XR_4 and XR_2, respectively.

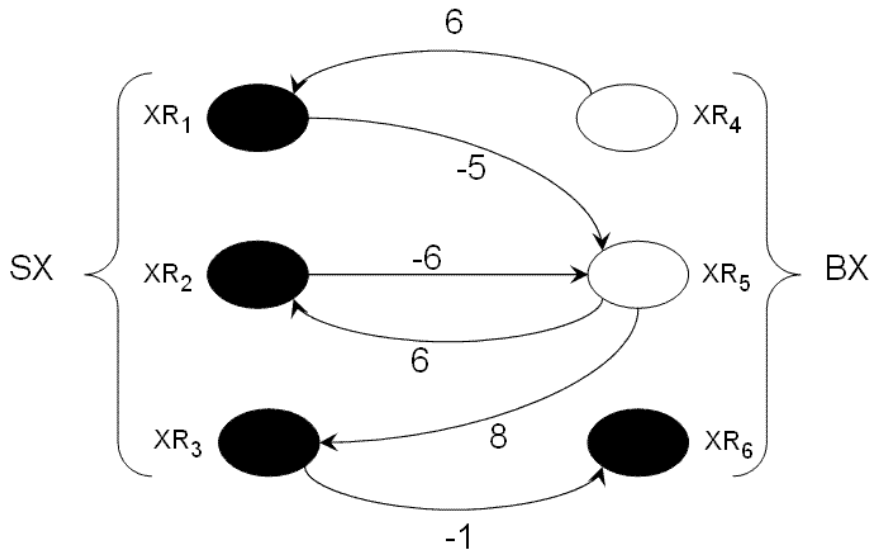

Figure 3.12: Residual network for Example 5, bid quote formulation. Absorbing nodes are colored black.

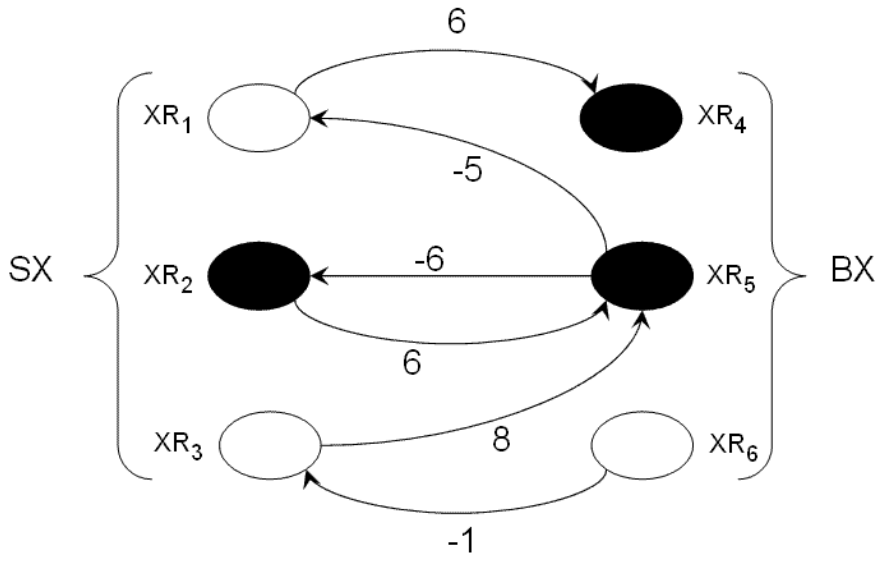

Figure 3.13: Residual network for Example 5, ask quote formulation. Absorbing nodes are colored black.

Run Time Comparisons

Figures 3.14 and 3.15 compare alternate algorithms/implementations for computing clearing and quote information in multiattribute markets based on the transportation problem.

I used the following linear optimization packages:

- QSopt Java package
- CPLEX 10.2 using the ILOG Java interface

The clearing problem was formulated as the transportation problem for both QSopt and CPLEX implementations, where the `NETWORK` algorithm was selected for the latter.

The quote information was computed in 3 separate ways:

1. **brute force** one iteration of the clearing algorithm for each node in the graph
2. **sensitivity** using the sensitivity analysis functionality of CPLEX
3. **shortest path** computing shortest path information between all nodes in the graph

All computations were performed on an Intel Pentium 4 CPU 3.40GHz, with Java heap set to initial/max size of 512MB.

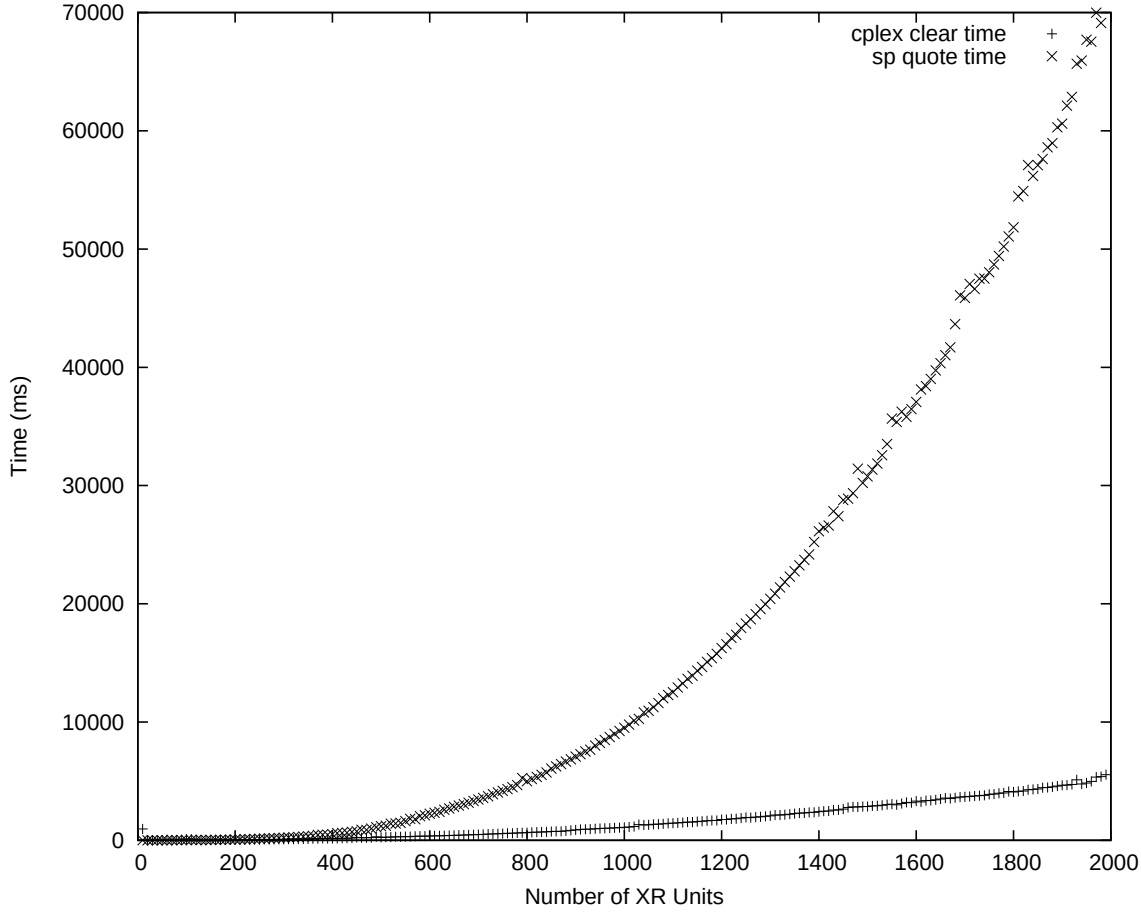

Figure 3.14: CPLEX clear time and shortest path quote time for all link quotes.

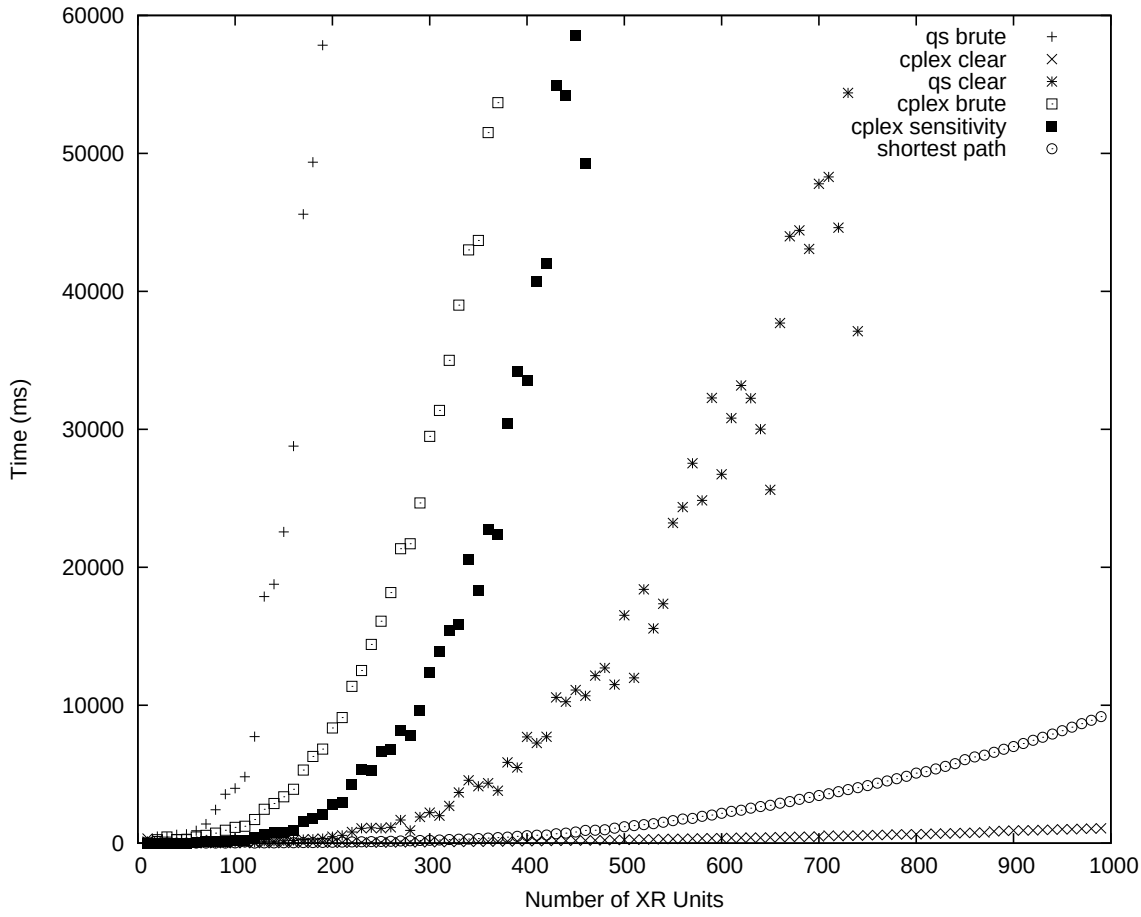

Figure 3.15: Clear and quote times, all methods.

3.4 Conclusion

For any auction design, efficient algorithms supporting clearing and information feedback are necessary for the auction to be of practical use. In this chapter I presented the algorithms which underlie multiattribute markets within the AB3D system. Although I outline and give complexity results for our multiattribute clearing algorithm, my main contribution in this chapter is in the development of a highly efficient information feedback procedure. This algorithm represents a significant contribution in that it provides the only known means of efficiently computing quote information for multiattribute call markets. As such, this algorithm is a crucial component supporting multiattribute call markets in AB3D.

I first described conditions on multiattribute bidding languages under which the clearing process may be separated into discrete steps of bilateral matching (*MMP*) and subsequent network optimization (*GMAP*). This approach was developed in joint work with Engel and Wellman. For a specific form of bids adhering to these conditions, I outlined the op-

eration of our clearing algorithm, and gave simulation results demonstrating algorithmic performance as a function of problem size.

Next, on a high level I described my novel approach to computing quote information. I described how the separation result employed by our clearing algorithm can be used to reduce the complexity of computing quote information. My approach starts with a solution to the *GMAP* network flow formulation, and computes link quote values which would admit new nodes to the optimal solution. Given these link quotes, I am then able to compute the quote for any single configuration with complexity linear in the number of offers. I present an example demonstrating how this method can be applied.

Next I turned to the problem of computing link quotes. I defined link quotes as the minimum solution to MMP_s with a given offer which increases either the trade surplus or the trade volume of the optimal solution. I introduced a naïve approach employing multiple iterations of the clearing algorithm. I also described how sensitivity analysis of the linear program could be used to approximate link quotes, and demonstrated that this approach does not provide sufficient computational savings to justify its use.

Finally, I introduced my shortest path algorithm, which can compute all link quotes with complexity that is polynomial in the number of offers. I presented several small examples giving the intuition behind this algorithm, as well as one larger example demonstrating the full algorithmic operation. I presented run time analysis showing that my shortest path approach achieves significant computational savings over both alternative approaches, demonstrating that this algorithm can compute link quotes for up to 1000 offers in under 10 seconds, using an inexpensive off-the-shelf processor.

Chapter 4

Complement-Free Valuations in Multiattribute Auctions

In Chapter 3 I introduced a multiattribute auction admitting polynomial-time clearing and information feedback operations. The polynomial-time guarantee for this mechanism comes at the expense of the expressiveness of the bidding language. That constraint imposed on the bidding language in turn places restrictions on the range of bidder valuations for which the mechanism is a suitable choice. In this chapter I attempt to characterize the relationship between bidder valuation classes and market efficiency for the multiattribute call market from Chapter 3.

I first present a hierarchy of bidder valuation classes derived by Lehmann et al. (2006), and use that hierarchy to characterize the bidding language from Section 3.1. Next I present known maximal polynomial-time-achievable efficiency ratios for several valuation classes, which I use to place the full iterative mechanism of Chapter 3 within the context of analytically derived efficiency results. I focus on the use of information feedback for the implementation of market-based algorithms in reaching favorable allocations. I review the existing literature on market-based algorithms, including the technical condition of gross substitutes under which such algorithms have been shown to obtain efficient outcomes.

I next present results from an agent-based simulation, using a model of valuations inspired by an existing supply chain simulation, the Trading Agent Competition Supply Chain Management game (TAC SCM) (Arunachalam and Sadeh, 2005). I show that in TAC SCM, the induced agent valuations naturally violate the substitutes condition, rendering inapplicable the efficiency results for market-based algorithms. Lacking analytical efficiency results given violations of the substitutes condition, I present a new metric for bidder valuations based on the severity with which agent valuations violate this condition. I provide experimental evidence from an agent-based simulation suggesting a correlation between this metric and expected efficiency loss, and also demonstrate that this efficiency loss is mitigated through the provision of information feedback.

4.1 Bidder Valuations

In this chapter I focus on the class of *complement-free* bidder valuations. A buyer exhibiting a complement-free valuation does not value a group of goods higher than the sum of the goods individually. For example, baseball card collectors often derive extra value from obtaining a complete set for a given year or team. Such a valuation would be complementary rather than complement-free. Non-complementarity assumptions are common in economic modeling. For example, consumers are typically assumed to have *diminishing marginal utilities* (Mas-Colell et al., 1995), meaning that consumers derive lower additional value from each additional unit of a good. I think of rich desserts as an example of a good with rapidly diminishing marginal utility, given that I usually start feeling ill before finishing my dessert.

For sellers, the notion of a complement-free valuation is amended slightly, as a seller valuation is for supplying rather than for consuming goods. A complement-free seller valuation does not exhibit a decreasing cost for supplying goods. Again, we can observe this assumption in economic modeling, as producers may be assumed to have increasing marginal costs (over some range of their production curve) or *decreasing returns to scale*. A common example for this type of valuation is a farmer, who in expanding his production is forced to use land which is decreasingly suitable for agriculture. His cost of producing each additional unit then increases as he needs to use more fertilizer or irrigation in utilizing lower quality land.

Multiattribute auctions typically assume agents have valuations exhibiting some degree of non-complementarity. For example, work on procurement auctions typically addresses the single-good setting (Che, 1997; Branco, 1997; Parkes and Kalagnanam, 2005), which imposes an extreme form of non-complementarity in defining goods as *perfect substitutes* (Mas-Colell et al., 1995).

As defined in Section 1.1, clearing an auction requires maximizing the trade surplus over all feasible and acceptable allocations. As valuations, bids constrain the set of acceptable global allocations, while also defining the objective function maximized in clearing. The extent to which bids do not accurately reflect agent valuations may induce suboptimal (but still *acceptable*) global allocations, as the optimization procedure is performed over an inaccurate objective function. A bidding language that is syntactically unable to fully convey agent valuations may therefore induce natural efficiency limitations in an auction. This effect may most easily be seen when agents have complementary valuations.

Example 6 *Consider the case of a baseball card collector who values a complete set of 10 baseball cards at a price $1000, but values each individual card at only $10. Further*

assume there is a competitor who does not care about the set, but values each individual card at $11. Efficiency dictates that an auction for these cards would allocate all 10 cards to the collector who values the complete set, as this maximizes the sum of utilities. If the cards are sold at auction and bidders are restricted to bidding on individual cards, our collector faces what is called an exposure problem *(Greenwald and Boyan, 2001). If he bids $100 on each card (i.e., $1000/10) he risks not completing his set and ending up with a net loss in utility from participating. If he is conservative in his bidding, he likely will just bid $10 on each card and end up winning none of them. This problem would be resolved if the auction allowed bidders to place bids on combinations of cards, as in a combinatorial auction (Cramton et al., 2005). The collector could then place a bid on the full set, indicating his willingness to pay $1000, but only if allocated the entire set.*

In Example 6, the bidding language was not expressive enough to allow the collector to provide his full valuation to the auction. This example demonstrates how an auction may yield suboptimal allocations if the bidding language is not sufficiently expressive with respect to the underlying bidder valuations.

Importantly, the computational complexity of computing an optimal global allocation increases with the expressiveness of the bidding language. This creates a natural tension between computational complexity and auction efficiency, even given the assumption of complement-free bidder valuations. As I show in the following sections, bidding languages that are appropriate from a computational standpoint are not sufficiently expressive for agent valuations likely to be encountered in multiattribute domains.

4.1.1 Complement-Free Valuations

The class of complement-free buyer valuations contains all valuations that are not *super-additive* over allocations.

Definition 4 *A buyer valuation is complement-free if for any two allocations g_a and g_b,*

$$v(g_a) + v(g_b) \geq v(g_a \cup g_b).$$

A seller valuation (cost function) is complement-free if it is not sub-additive over configurations. This condition precludes volume pricing discounts and minimum quantity preferences.

Definition 5 *A seller valuation is complement-free if for any two allocations g_a and g_b,*

$$v(g_a) + v(g_b) \leq v(g_a \cup g_b).$$

It is known that no polynomial clearing algorithm can guarantee better than a 2-approximation for the general class of complement-free valuations (Dobzinski et al., 2005). In the following subsections, I present subclasses of the class of complement-free valuations, providing known efficiency bounds for polynomial-time allocation.[1] Some of the valuation classes are defined by constraints on the relative valuations of alternate allocations, while others are defined by constraints on the syntax of representations. Valuation classes are typically provided from the context of a buyer; I provide both buyer and seller definitions where appropriate.

4.1.2 Submodular Valuations

For buyers, the condition of submodularity additionally imposes decreasing *marginal valuations*. Defining the marginal valuation of x with respect to allocation g as $v(x|g) = v(g \cup x) - v(g)$, submodularity requires that for $g_a \subseteq g_b, v(x|g_a) \geq v(x|g_b)$. The following alternate definition more clearly shows that submodular valuations are also complement-free.

Definition 6 *A buyer valuation v is submodular if for any two allocations g_a, g_b,*

$$v(g_a) + v(g_b) \geq v(g_a \cup g_b) + v(g_a \cap g_b).$$

Again, the direction of the inequality is simply reversed for seller valuations, implying non-decreasing marginal cost.

4.1.3 Gross Substitutes

Gross substitutes valuations are defined with respect to the goods an agent demands when facing a set of prices. To define valuations exhibiting gross substitutability, I first define an agent demand correspondence with respect to valuations and prices. The following definitions are with respect to buyers.

Definition 7 *Given valuation v and vector of configuration prices $\vec{p} = (p_{x_1}, p_{x_2}, \ldots, p_{x_n})$, the demand correspondence $d(v|\vec{p})$ maps to the set of allocations which maximize $v(g) - \sum_{x \in g} p_x$.*

[1] In the multiattribute setting, unique goods correspond to the configurations. I borrow both notation and analytical complexity results from Lehmann et al. (2006) in this section, with notation amended slightly for multiattribute domains.

The demand correspondence simply identifies the agent *demand set*, comprising the set of goods which maximize an agent valuation given a vector of good prices. Now, I can define the gross substitutes condition based on how the agent demand set changes with respect to prices.

Definition 8 *A valuation v is of the class gross substitutes (GS) if for any price vectors $\vec{p}$ and $\vec{q}$ with $p_i \leq q_i \; \forall i$ and $g_1 \in d(v|\vec{p})$, there exists $g_2 \in d(v|\vec{q})$ such that $\{x \in g_1 | p_x = q_x\} \subset g_2$.*

Informally, the gross substitutes condition for buyers states that the demand for a given configuration is nondecreasing in the price of any other configuration. For sellers, the condition changes so that the supply of a given configuration is nonincreasing in the prices of other configurations. To give some intuition for this condition, we can consider two different goods which have a common purpose, e.g., two alternate camera models. Assume that given the market prices of all cameras, a buyer had selected an ideal model to purchase. If the buyer's valuation for cameras satisfies the gross substitutes condition, then a price increase for one or more competitor models should not make him decide against buying his initial selection.

4.1.4 Syntactic Valuation Classes

Syntactic valuations are built from *atomic valuations* and operators on those valuations.

Definition 9 *The atomic valuation (x, p) gives the value p to any allocation containing a unit of configuration x, and value zero to all other allocations.*

Next, I define the operators *OR* and *XOR* over valuations as follows:

Definition 10 *Let v_1 and v_2 be two valuations defined on the space G of allocations. The valuations $v_1 + v_2$ (OR) and $v_1 \oplus v_2$ (XOR) are defined by:*

$$(v_1 + v_2)(g) = \max_{x \subseteq g}(v_1(x) + v_2(g \setminus x)),$$

$$(v_1 \oplus v_2)(g) = \max(v_1(g), v_2(g)).$$

Informally, the valuation $(v_1 + v_2)(g)$ divides up allocation g among valuations v_1 and v_2 such that the sum of the resulting valuations is maximized. The valuation $(v_1 \oplus v_2)(g)$ gives the entire allocation to v_1 or v_2, depending on which values g higher.

Subclasses of complement-free valuations are derived by placing restrictions on how the *OR* and *XOR* operators may be combined. Class *OS* valuations are created using only the *OR* operator over atomic valuations, and allow for the expression of additive valuations. Class *XS* valuations are created by applying the *XOR* operator over atomic valuations, and allow for the expression of substitute valuations.

Any valuation composed of *OR* and *XOR* (applied in an arbitrary order) is expressible by applying the *XOR* operator over *OS* valuations.

Definition 11 *A valuation is XOS if expressible through a sequence of applications of OR and XOR operators over atomic valuations.*

For example, as a buy bid, the *XOS* valuation

$$(x_1, p_1) \oplus ((x_2, p_2) + (x_3, p_3))$$

expresses a willingness to accept any one of the following (allocation,payment) pairs:

$$\{(x_1, p_1), (x_2, p_2), (x_3, p_3), (\{x_2, x_3\}, p_2 + p_3)\}.$$

For clearing a combinatorial auction given *XOS* valuations, the best approximation factor that can be guaranteed in polynomial time is known to be bounded above by 2, and bounded below by $\frac{4}{3}$ (Dobzinski et al., 2005).

Applying the *OR* operator over *XS* valuations yields valuations of class *OXS*, a subclass of *XOS*.

Definition 12 *A valuation is OXS if expressible through the application of OR operators over XS valuations.*

For example, as a buy bid, the valuation

$$(x_1, p_1) + ((x_2, p_2) \oplus (x_3, p_3))$$

expresses the willingness to buy x_1 at a price of p_1, and independently expresses a willingness to buy either x_2 at a price of p_2, or x_3 at a price of p_3 (but not both), giving the following acceptable allocations:

$$\{(x_1, p_1), (x_2, p_2), (x_3, p_3), (\{x_1, x_3\}, p_1 + p_3), (\{x_1, x_2\}, p_1 + p_2)\}.$$

If all valuations are of class *OXS*, valuations can be directly expressed through *OXR* bids, and the clearing algorithm presented in Section 3.2 will produce polynomial-time efficiency.

The bidding language constructs from Section 3.1 can be classified within this syntactic framework. The multiattribute point (x, p, q) expresses the valuation equivalent to an *OR* expression over $|q|$ atomic (x, p) valuations:

$$\underbrace{(x,p)+(x,p)+\cdots}_{\text{total of } |q| \text{ elements}}.$$

The additional quantity designation in a multiattribute point provides compactness over the equivalent *OR* expression when valuations are linear in quantity.

The multiattribute *XR* unit with quantity q defines the valuation equivalent to the following expression over atomic valuations:

$$\underbrace{((x_1,p_1)\oplus\cdots\oplus(x_N,p_N))+((x_1,p_1)\oplus\cdots\oplus(x_N,p_N))+\cdots}_{\text{total of } |q| \text{ elements}}.$$

The multiattribute *XR* unit is less expressive than the general class *OXS* because it defines an *OR* over a set of identical *XOR* expressions, thus imposing a constraint that valuations be linear in quantity, and *configuration parity*, i.e., the quantity offered by a bid is configuration-independent (Engel et al., 2006). The multiattribute *OXR* bid provides full expressiveness with respect to class *OXS*, where the multiattribute *OXR* bid $\{XR_1, XR_2, \ldots, XR_M\}$ is equivalent to the following *OR* expression over *XR* units:

$$(XR_1 + XR_2 + \ldots + XR_M).$$

4.1.5 Valuation Hierarchy

I have now defined the following valuation classes:

- *OXS* - valuations admitting an *OR*-of-*XOR* expression
- *GS* - gross substitute valuations
- *SM* - submodular valuations
- *XOS* - valuations admitting an *XOR*-of-*OR* expression
- *CF* - complement-free valuations

Lehmann et al. (2006) showed that the following hierarchy holds over the above classes, where the containment is strict in each case:

$$OXS \subset GS \subset SM \subset XOS \subset CF.$$

4.1.6 Valuations for Multiattribute Auctions

The call market presented in Chapter 3 supports the direct expression of *OXS* valuations, with information feedback supporting the implementation of market-based algorithms for efficiency under *GS* valuations. Unfortunately, many valuations which are natural to multi-attribute domains fall outside of class *OXS*. For example, Bichler and Kalagnanam (2005) cite the common need to enforce *homogeneity* of allocations, meaning that all configurations in an allocation must have identical values for one or more attributes. Valuations which place higher values on homogeneous allocations, or conversely, heterogeneous allocations, are expressible with an *XOS* bidding language but not with a language of class *OXS*.

My motivation in studying allocation with valuations of increasing complexity arose when applying the auction from Chapter 3 to TAC SCM. In TAC SCM, manufacturers assemble finished goods from a limited inventory of available components. As Example 7 makes clear, the induced seller valuations fall outside of class *OXS*, and may violate the gross substitutes condition.

Example 7 *Consider the case of a PC built from two components: cpu and memory. Assume that a seller has one unit of cpu = fast, one unit of cpu = slow, one unit each for memory* $\in \{large, medium, small\}$*, and the configurations are assembled in the following manner:*

1. *configuration* x_1*:* $\{fast, large\}$
2. *configuration* x_2*:* $\{fast, medium\}$
3. *configuration* x_3*:* $\{slow, small\}$
4. *configuration* x_4*:* $\{slow, medium\}$

The production possibilities are then:

$$\{x_1, x_4\}, \{x_1, x_3\}, \{x_2, x_3\}.$$

The induced seller valuation is not expressible using an OXS language. Designating the unit cost of x_i *by* p_i*, the nearest OXS approximations require the seller to either overstate (bid 4.1) or understate (bids 4.2 and 4.3) his production capabilities:*

$$((x_1, p_1) \oplus (x_2, p_2)) + ((x_3, p_3) \oplus (x_4, p_4)) \tag{4.1}$$

$$(x_1, p_1) + ((x_3, p_3) \oplus (x_4, p_4)) \tag{4.2}$$

$$((x_1, p_1) \oplus (x_2, p_2)) + (x_3, p_3) \tag{4.3}$$

While reserve prices over these allocations are not expressible using an OXS language, an

XOS-based language is sufficient:

$$((x_1, p_1) + (x_4, p_4)) \oplus ((x_1, p_1) + (x_3, p_3)) \oplus ((x_2, p_2) + (x_3, p_3)).$$

Furthermore, the valuations from Example 7 are also not in class *GS*. Assume that within the above production possibilities, the seller has a unit cost of 3 for all configurations, with total cost additive in unit cost. The equivalent *XOS* valuation would be:

$$((x_1, 3) + (x_4, 3)) \oplus ((x_1, 3) + (x_3, 3)) \oplus ((x_2, 3) + (x_3, 3)).$$

Prices for configurations are also necessary to evaluate the gross substitutes condition. Assume the following configuration prices:

$$p(x_1) = 5;\ p(x_2) = 4;\ p(x_3) = 4;\ p(x_4) = 5.$$

At these prices, the optimal production bundle is (x_1, x_4) which yields a surplus of 4. If the price of x_1 drops to zero, the optimal production bundle becomes (x_2, x_3), yielding a surplus of 2. Hence, the supply of x_4 decreases with a decrease in the price of x_1, which violates the gross substitutes condition for sellers.

4.2 Allocation with Complement-Free Valuations

In this section, I summarize hardness results for allocation with subclasses of complement-free valuations, and explain the theory underlying market-based algorithms.

4.2.1 Hardness Results

When underlying agent valuations are in the class *OXS*, valuations may be directly expressed with *OXR* bids, and the polynomial algorithm from Section 3.2 can obtain efficiency with a direct-revelation mechanism (presuming an adequate incentive structure).

The existence of efficient mechanisms when agent valuations fall somewhere between *GS* and *CF* is an open problem. For the class *GS*, a Walrasian equilibrium exists and a market-based algorithm can provide a fully polynomial approximation scheme (polynomial in the number of goods and traders) for efficient global allocation (Kelso and Crawford, 1982). An important caveat for multiattribute domains is that the number of goods is exponential in the number of discrete attributes, while continuous attributes induce an infinite

number of goods.

Treating a unit of a given configuration as a unique good, we can look to the combinatorial auction literature for results. These results tend to fall into two categories: hardness results for exact solutions, and efficiency guarantees for polynomial-time algorithms. Hardness results show that even with two valuations which are additive with a budget limit, optimal allocation is NP-hard (Lehmann et al., 2006). On the other hand, existing polynomial (greedy) algorithms for complement-free valuations guarantee only 50% efficiency (Lehmann et al., 2006). A further drawback is that the transformation of these algorithms to multiattribute domains (which lack a pre-defined set of goods) is not always trivial. These approximation algorithms are also intended for a single seller setting with no reserve price.

The following table from Dobzinski et al. (2005) gives known bounds for approximation factors achievable in polynomial time for several complement-free valuation classes:

Valuation Class	**Value**	**Demand**	**General Comm.**
General	$\mathcal{O}(\frac{m}{\log m})$	$\mathcal{O}(m^{\frac{1}{2}})$	$\Omega(m^{\frac{1}{2}-\varepsilon})$
CF	$\mathcal{O}(m^{\frac{1}{2}})$	$\mathcal{O}(\log m)$	≥ 2
XOS			≤ 2 $\geq \frac{4}{3}$
SM	≤ 2 ≥ 1.02		$\geq 1+\frac{1}{2m}$
GS	1		

Table 4.1: Known efficiency approximation factors for m goods achievable in polynomial time, from Dobzinski et al. (2005).

4.2.2 Market-Based Algorithms

Valuations satisfying the gross substitutes condition admit efficiency through *market-based algorithms.* Market-based algorithms derive from the ideas of *general equilibrium theory* (Arrow et al., 1959), under which markets simultaneously maximize efficiency and achieve a perfect balance of supply and demand, given profit-maximizing behavior of market participants. The condition of gross substitutes has been identified in a number of settings as sufficient to guarantee existence of a Walrasian equilibrium (Arrow et al., 1959; Kelso and Crawford, 1982; Gul and Stacchetti, 1999; Milgrom and Strulovici, 2006).

Market-based algorithms operate by iteratively providing agents with price quotes, requiring that agents express demand sets reflecting their optimal consumption or production choices at the given prices. Demand sets are expressible in any bidding language of com-

plexity equal to or greater than class *OS*. Prices are adjusted at each iteration based on the relative supply and demand of each type of good, until the market reaches equilibrium. Computationally, market-based algorithms provide a fully polynomial approximation scheme, with complexity that is polynomial in the number of bidders, goods, and the inverse of the approximation factor (Lehmann et al., 2006).

In market-based algorithms, market participants respond to price quotes by submitting supply and demand sets which maximize their respective utilities given prices, dynamically adjusting their supply and demand in response to market price adjustments. At equilibrium, the collective market demand equals the collective market supply, at which point the market maximizes efficiency, i.e., maximizes true surplus across all global allocations. Arrow et al. (1959) showed that gross substitutes is sufficient to guarantee existence of such an equilibrium.

The ideas of general equilibrium theory have subsequently been extended to show that the gross substitutes condition is sufficient to guarantee equilibrium with discrete heterogeneous goods and thin markets (Kelso and Crawford, 1982), and that gross substitutes is both sufficient and necessary to ensure equilibrium existence in discrete domains (Gul and Stacchetti, 1999).

Although the multiattribute bidding language presented in Section 3.1 supports the expression of demand sets, and the algorithm from Section 3.3 supports the provision of price quotes on all items, the full mechanism does not implement a provably convergent market-based algorithm (assuming myopic best response bidding). This is due to the form of the price adjustment process, which must be made proportional to the excess market demand to guarantee convergence. I explore the consequences of this distinction in Section 5.8. For the remainder of this chapter, I evaluate the induced efficiency of myopic best-response bidding given this alternate form of market-based algorithm.

4.3 A New Valuation Metric

Given the ability to implement a form of market-based algorithm, the question remains as to the efficiency limitations of my market design when valuations are not contained in *GS*. To better characterize the valuations for which market-based algorithms are appropriate, in this section I define a new metric which classifies valuations not contained in *GS*.

4.3.1 Gross Substitutes Revisited

As defined in Section 4.1.3, the gross substitutes condition requires that the demand for goods be nondecreasing in the prices of other goods. The motivation behind this condition is that a price adjustment process will ultimately reach equilibrium if a price perturbation intended to reduce the demand of over-demanded goods does not reduce the demand for other under-demanded goods. Similarly, a price decrease intended to increase demand for under-demanded goods should not increase demand for over-demanded goods.

For valuation v satisfying the gross substitutes condition, the demand correspondence condition holds for all price vectors and perturbations. Formally, given demand correspondence $d(v|\vec{p})$ as defined previously, the set of allocations maximizing the sum $v(g) - \sum_{x \in g} p_x$, for all vectors of configuration prices $\vec{p} = (p_{x_1}, p_{x_2}, \ldots, p_{x_n}) \in \Re^n_+$, and all single price perturbations $\overrightarrow{dp} \in \Re^n_+$, for any $g_1 \in d(v|\vec{p})$ there exists $g_2 \in d(v|\vec{q})$ such that $\{x \in g_1 | p_x = q_x\} \subset g_2$, where $\vec{q} = \vec{p} + \overrightarrow{dp}$.

4.3.2 Gross Substitutes Violations

I hypothesize that the *degree* to which the gross substitutes condition is violated will provide a metric on valuations which correlates with the efficiency of market-based algorithms. I first formulate the substitutes violation with respect to any single price vector and price perturbation, and subsequently define the substitutes violation for a valuation with respect to the full space of prices and single-price perturbations.

For non-negative vectors $\vec{p}$ and $\overrightarrow{dp}$, again define $\vec{q} = \vec{p} + \overrightarrow{dp}$. For each $g_i \in d(v|\vec{p})$, I define the *gross substitutes violation* as the aggregate decrease in demand for items with $p_x = q_x$. Since $d(v|\vec{q})$ may map to multiple demand sets, I take the minimum over all such sets.

$$GSV(v, \vec{p}, \vec{q}, g_i) = \min_{g \in d(v|\vec{q})} \; | \{x \in g_i | p_x = q_x\} \setminus \{x \in g | p_x = q_x\} | \, .$$

Intuitively, this measure counts the *number* of violations of the gross substitutes condition for a specific initial price vector and price change. Valuations satisfying the gross substitutes condition will have a violation count of zero for all initial prices, demand sets, and perturbations. Valuations which do not satisfy the gross substitutes condition will have positive values of *GSV* for one or more combinations of $(\vec{p}, \overrightarrow{dp})$.

To simplify the exposition, I assume $d(v|\vec{p})$ maps to a single g for any $\vec{p}$, and use $x \in d(v|\vec{p})$ to indicate a good from that demand set.

From any single price vector $\vec{p}$, define the gross substitutes violation of a valuation for that price vector as the average *GSV* over all minimal single-price perturbations which ensure a new demand set:

$$GSV(v,\vec{p}) = \frac{1}{n}\sum_{i=1}^{n} \quad |\{x \in d(v|\vec{p})|p_x = q_x\} \setminus \{x \in d(v|\vec{q})|p_x = q_x\}|.$$

where:

$$\vec{q} = (p_1, p_2, \ldots, p_i + dp_i, \ldots, p_n)$$

and

$$dp_i = \min_{dp} \quad \text{s.t.} \quad d(v|\vec{p}) \neq d(v|(p_1, p_2, \ldots, p_i + dp, \ldots, p_n)).$$

Next, define the *expected* gross substitutes violation for a valuation as the expected value of *GSV* for random $\vec{p}$:

$$EGSV(v) = E[GSV(v,\vec{p})],$$

where

$$\forall i \quad p_i \sim U[0,\bar{p}].$$

My hypothesis is that for a set of bidders with valuations drawn from a small range of *EGSV* values, market-based algorithms will reach a similar degree of efficiency, where the realized efficiency is decreasing in the average *EGSV* value. This result would extend naturally to the case of gross substitutes (equivalently, an *EGSV* value of zero), where market-based algorithms achieve full efficiency. The intuition behind the *EGSV* metric, specifically for using the expected *GSV* of a valuation (the average, rather than the maximum or minimum) is that since the price trajectory of a market-based algorithm covers only a subset of the full price space, the average violation factors in the probability of seeing any specific violation.

4.4 Testing the *EGSV*-Efficiency Relationship

To evaluate the relationship between *EGSV* values and market efficiency, I employed a component-based model of configurations like that presented in Example 7, where valuation complexity is determined by the constitution of the configurations, i.e., the *configuration structure*, as well as by the respective inventory levels and component costs of sellers.

For example, a valuation defined on a configuration structure with three alternate configurations will violate *GS* to the extent that swapping production from one configuration to another requires additional components that are allocated to the third configuration. Treating configurations $\{x_1, x_2, x_3\}$ as sets of components, assume that switching production from x_1 to x_2 requires additional components $x_2 \setminus x_1$. If an agent has no additional inventory of the components $(x_2 \setminus x_1) \cap x_3$ then the induced valuation will have a *GSV* of 1 for some price levels.

In this way, variation both in the composition of configurations and the inventory levels of agents induces different levels of substitutability in agent valuations. In the example above, if $x_2 \setminus x_1$ included 2 distinct components in use by 2 different configurations, then the bidder valuation would have a *GSV* of 2 for some price vectors and a non-zero *EGSV* value. Conversely, if an agent had excess inventory of $x_2 \setminus x_1$, then the induced valuation would have a *GSV* of zero for all prices, and therefore the valuation would have an *EGSV* value of zero.

4.4.1 Valuation Generation

My experimental approach to generating a configuration structure is to generate random configurations until a total of 20 unique configurations is produced. For each configuration, I probabilistically include any one of eight unique components in the configuration (i.e., configurations may have variable numbers of components), while additionally requiring that any single configuration have at least three components.

Once I have generated a set of 20 configurations, I randomly sample costs and inventory to generate a seller valuation, where component inventories and costs are each independent and identically distributed. Seller inventory for each component is drawn from the discrete uniform distribution $[0, 3]$, while seller costs per component are drawn from the discrete uniform distribution $[30, 80]$.

I then test the induced valuation with respect to the same price distribution from which agent valuations are drawn. I sample prices, and for each price sample $\vec{p}$, I compute

$GSV(v, \vec{p})$. To compute $GSV(v, \vec{p})$, I first compute the optimal production set $g^* = d(v|\vec{p})$, and sum the gross substitutes violations over all minimal single prices changes which ensure a new optimal production set.

I iterate the above process with random price samples until the standard error of the expected gross substitutes violation is below .05, producing a single valuation. I generated a set of 100 valuations for each configuration structure, recording the costs and inventory, along with the *EGSV* value for each such valuation. I generated and tested seller valuations for 277 configuration structures, yielding a total of 27700 seller valuations.

4.4.2 Market Simulation

Each problem instance comprises a set of 10 buyers and 10 sellers. For each configuration structure, I first sort the set of 100 generated seller valuations by *EGSV* values. I define a unique problem instance for each contiguous set of 10 seller valuations, using the previously generated inventories and costs for each valuation, and taking the average *EGSV* value of the 10 sellers to classify the problem instance. I denote this average *EGSV* by *aGSV*. I thus generate 90 problem instances for each configuration structure.

I randomly generate buyer valuations for each problem instance. Each buyer has demand for two units, with full substitutability among the goods (i.e., each will accept any combination of two goods at probabilistically generated configuration reserve prices). Buyer reserve prices are drawn from the discrete uniform distribution $[400, 500]$ for each configuration.

For each problem instance, I first formulate the allocation problem as a linear program to determine the maximum achievable efficiency. I then simulate bidding until quiescence, computing the fraction of maximal efficiency at quiescence. To identify the benefit of information feedback, I took the first iteration of bidding as the direct-revelation outcome.

Although I believe that the direct expression of substitutes—as in the *OXR* bidding language—is indispensable for large or continuous attribute domains, I conducted the same bidding simulation with a class *OS* bidding language to determine whether the expression of substitutes provides efficiency advantages in domains with small numbers of configurations. Each problem instance thus produces four data points (one for each of $(direct, iterative) \times (OS, OXR)$).

I simulated myopic best response bidding for buyers and sellers, having buyers and sellers bid their true values at each iteration on a profit-maximizing set of goods. Given that the bidding language is not sufficiently expressive to directly reveal seller valuations, sellers are forced to approximate their valuations with bids. To generate an *OS* bid, sellers

find the feasible production bundle that maximizes surplus at current prices (assuming a uniform price for all configurations when quotes are not available). To generate an optimal *OXR* bid, sellers start with the optimal *OS* bid, subsequently expanding the bid to a feasible *OXR* bid.

4.5 Simulation Results

I aggregated the simulation results over all configuration structures and sorted the data by aGSV value into 10 bins, computing the sample mean of the achieved fraction of total efficiency. Figure 4.1 plots the achieved fraction of maximal efficiency as a function of aGSV value (with aGSV value averaged over all samples within a bin), for both direct-revelation and iterative mechanisms, for both the *OS* and *OXR* bidding languages.

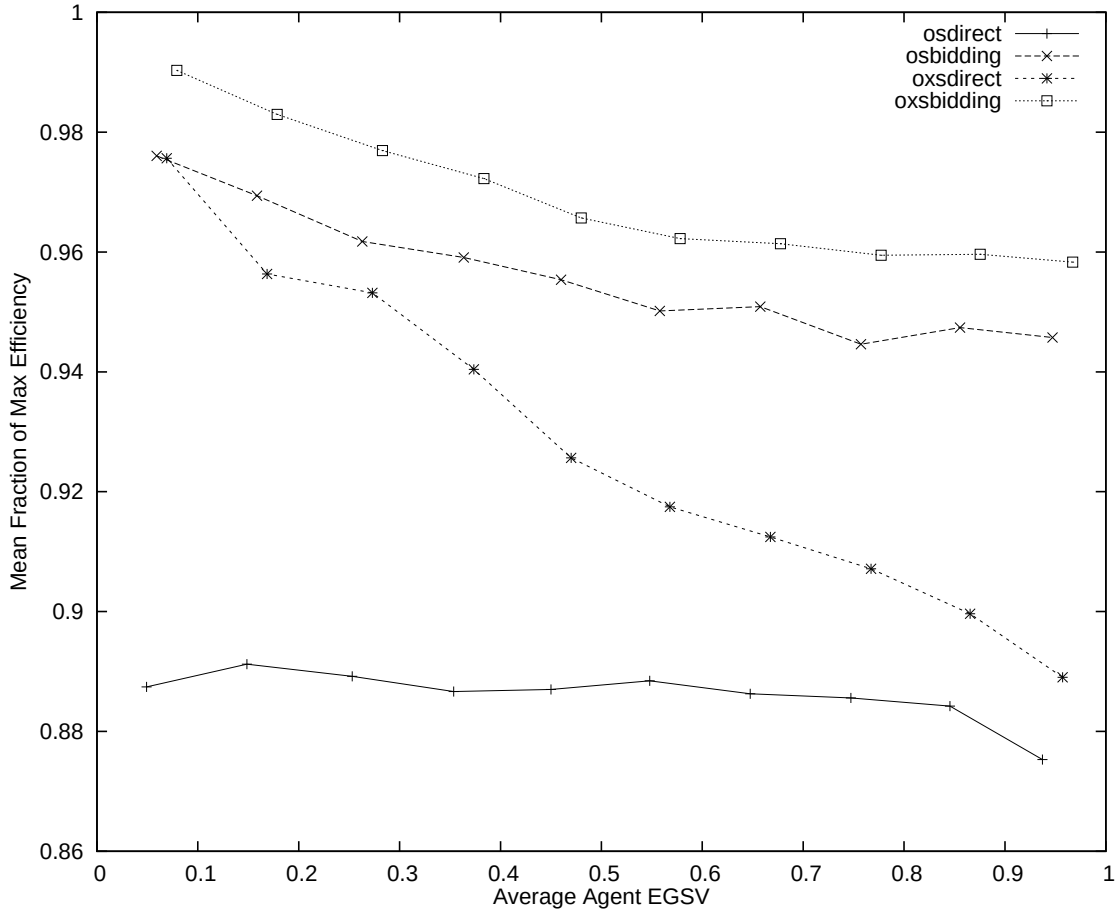

Figure 4.1: Relationship of mean efficiency and aGSV for different mechanisms.

For aGSV values close to zero, the substitutes condition is nearly satisfied for all valuations, indicating that prices inducing violations of the substitutes condition occur

infrequently. For such problem instances, I would expect iterative mechanisms to reach nearly maximal efficiency. Figure 4.1 confirms this hypothesis, as both iterative mechanisms average more than 97% efficiency for low aGSV values.

I note that while the direct *OS* mechanism suffers from relatively lower efficiency across all aGSV values ($\sim$ 90%), the direct *OXR* mechanism achieves a high fraction of efficiency for low aGSV values ($\sim$ 97%). I conjecture that the majority of low *EGSV* valuations were also in class *OXS*, and that given *OXS* valuations, the ability to express substitutability through bids provides a significant efficiency advantage in direct-revelation mechanisms.

Notable in Figure 4.1 is that the iterative mechanisms relatively outperform the direct *OXR* mechanism, by a margin that is increasing in aGSV value. I suspect this reflects bidder valuations which increasingly deviate from class *OXS* with higher *EGSV* values. Despite this increasing valuation complexity, the iterative mechanisms hold to a high level of efficiency, falling only to approximately 95% as aGSV values reach 1.4. From this I conclude that information feedback is able to compensate for the lack of expressive power of a class *OXS* bidding language.

I observe that the iterative *OXR* mechanism outperforms the *OS* mechanism over all aGSV values. I hypothesize that the direct expression of substitutes allows the market-based algorithm to escape local maxima, as this mechanism does not implement a provably convergent market-based algorithm for *OS* bids.

In my motivating problem, the TAC SCM game, seller costs and inventory levels are not static, but rather change throughout the course of a game. In seeking to better define market efficiency for a specific configuration structure given stochastic inventory levels and costs, I also averaged the *EGSV* values across all valuations from a given inventory structure, to produce an expected *EGSV* value for the configuration structure itself. The plot in Figure 4.2 presents the data for efficiency as a function of the expected *EGSV* of the configuration structures, where each data point averages data over 5 configuration structures.

In Figure 4.2 I observe a noisier relationship between average *EGSV* and efficiency, but note a fairly linear relationship between average *EGSV* of a configuration structure and efficiency of the iterative *OXR* mechanism. I infer from this plot that the expected *EGSV* for a given configuration structure provides a useful metric in predicting the performance of my mechanism for a given problem instance, most accurately for the higher-efficiency iterative *OXR* mechanism.

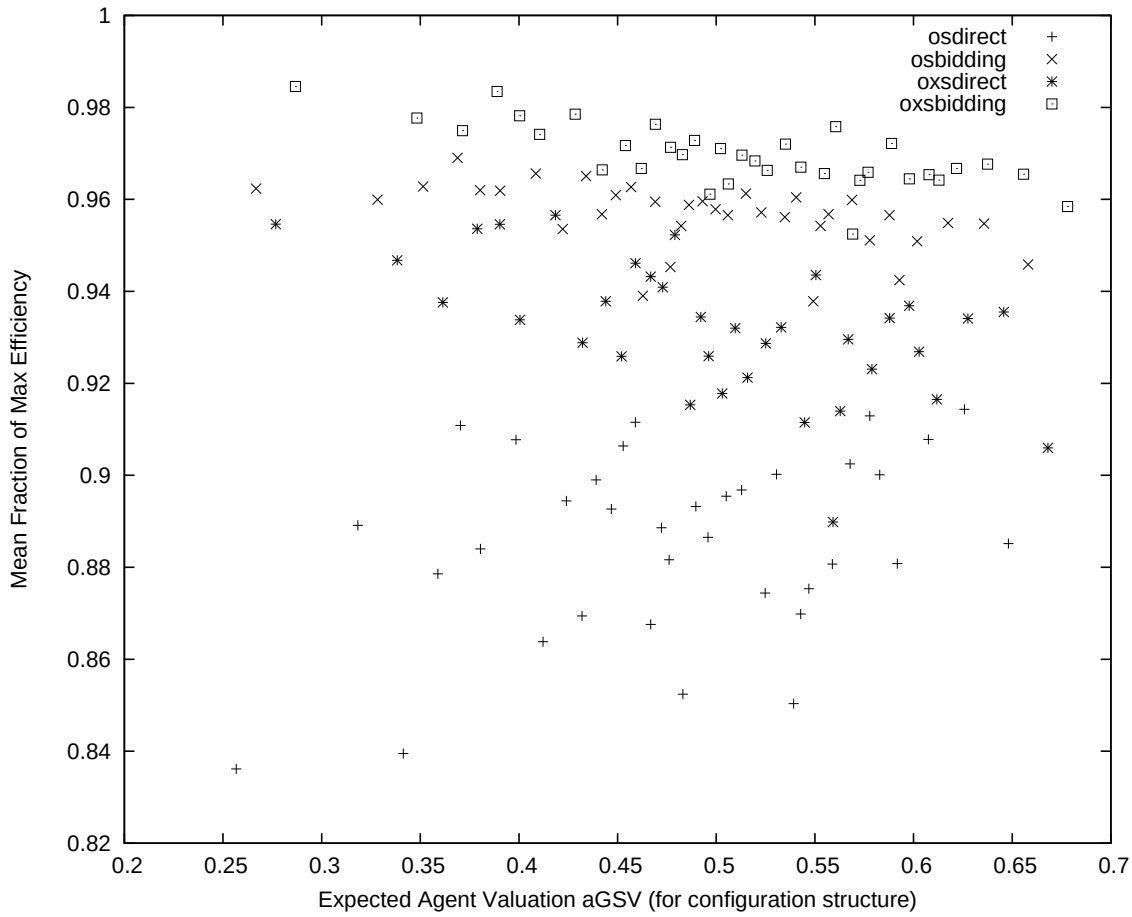

Figure 4.2: Relationship of mean efficiency and expected aGSV for different mechanisms.

4.6 Conclusion

In this chapter I characterized the expected efficiency of multiattribute call markets given various assumptions on bidder valuations. I discussed the expected efficiency of my mechanism from the perspective of known hardness results derived for combinatorial auction settings, given complement-free bidder valuations. I described the operation of market-based algorithms and the technical condition of gross substitutes. I demonstrated that the information-feedback functionality of my market design supports efficient allocations given valuations satisfying the gross substitutes condition.

Importantly, I demonstrated that the addition of information feedback functionality to an *OXS*-based multiattribute call market design successfully compensates for the expressive deficiencies imposed by a restricted bidding language. The addition of information feedback support to our previously developed clearing algorithm thus extends the range of bidder valuations for which this market design is able to support efficiency.

Inspired by a multiattribute supply chain setting, I presented natural ways in which multiattribute valuations may violate the gross substitutes condition. Lacking theoretical results as to the expected efficiency of my market design for valuations likely to be encountered in practice, I presented a new metric on bidder valuations, derived from the ways in which valuations violate the substitutes condition. I then presented evidence that this metric correlates with the expected efficiency of market-based algorithms. Also to my knowledge, this is the first such experiment presenting evidence of a correlation between the efficiency of market-based algorithms and the measured gross substitutes violation of bidder valuations.

Chapter 5
Multiattribute Supply Chain Simulation

The component-based valuation model of Chapter 4 was inspired by the Trading Agent Competition Supply Chain Management Game (TAC SCM) (Arunachalam and Sadeh, 2005), a multi-agent trading competition designed to foster research in agent design, specifically in the domain of supply chain automation. In this chapter I employ the full TAC SCM simulation environment in devising a benchmark for multiattribute auctions. Scoring game outcomes with an existing tool for calculating market efficiency of a game instance, I evaluate alternative market designs and corresponding bidder strategies within the simulation environment.

The term supply chain refers to the multiple levels within the economy at which goods are manufactured and used as inputs for the production of other goods, which may in turn be used as inputs once again in the next link of the chain. Traditionally, negotiations among supply chain participants take place in face-to-face settings, with supply contracts renegotiated on an infrequent basis.

Given the global volume of business-to-business transactions, the automation of supply chain negotiation presents an opportunity to realize substantial economic gains. The details, or attributes, inherent in most procurement contracts preclude the use of price-only auction formats for negotiation in many supply chain settings. The use of multiattribute reverse auctions, as described in Section 1, as well as electronic *request for quote*(RFQ) negotiations constitute two main thrusts of technology-based approaches to increasing supply chain efficiency (Kaufmann and Carter, 2004). In a request for quote, buyers solicit supplier bids to provide a specific product or service. RFQ-based negotiations function much like reverse auctions, but lack formal rules for selecting winning bids.

TAC SCM simulates one link of a supply chain in a computer manufacturing domain, where suppliers provide inputs in the form of computer parts, while customers provide final demand for finished computer products. In the middle stand manufacturing agents, who procure parts, assemble them into finished products, and ultimately sell the manufactured PCs to customers. The game aspect of TAC SCM comes from the fact that while suppliers

and customers are part of the simulation environment (with stochastically determined behavior), the manufacturers are designed by contestants, who compete to generate the most profit over the course of a game. All negotiation within TAC SCM is conducted through a request-for-quote process, whereby manufacturing agents solicit bids from suppliers through RFQs, in turn bidding on customer RFQs to win PC supply contracts. Figure 5.1 depicts the high-level interaction among agents in TAC SCM.

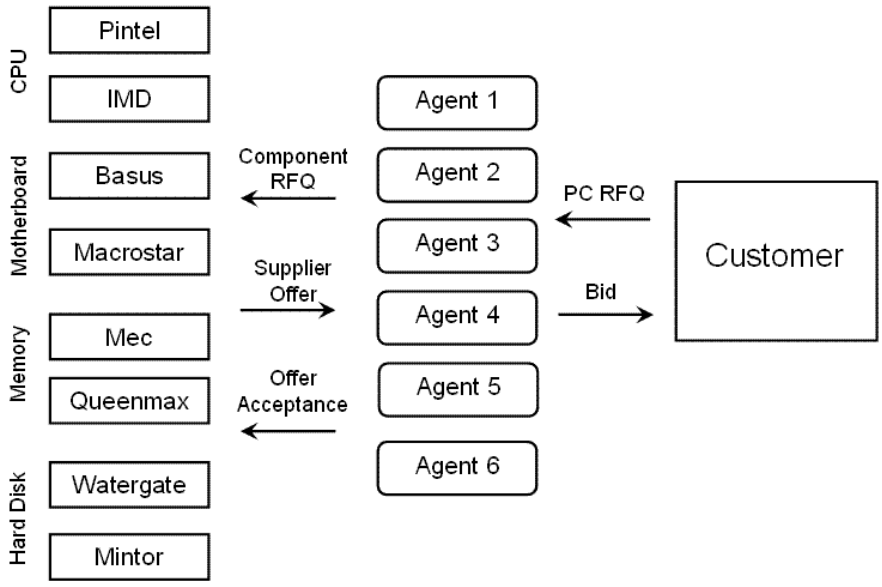

Figure 5.1: Diagram of TAC SCM supply chain, with suppliers shown at left, manufacturing agents in the middle, and customer at right.

TAC SCM presents an existing simulation environment approximating a real-world potential application domain for multiattribute call markets. The PCs traded between customers and manufacturers are inherently multiattribute goods, with attributes providing contract details of SKU, delivery date, and penalty. By replacing the existing RFQ-based customer negotiation mechanism with a multiattribute market, I can measure efficiency in the context of a broad operational scenario.

Whereas in Section 4.4.1 I defined agent costs, and inventories as exogenous variables for each problem instance, in this chapter these variables become time-dependent endogenous variables which evolve with the outcome of game play. In using the full TAC SCM simulation, I obtain problem instances with bidder valuations that are grounded in a higher degree of realism, while the use of an existing simulation environment provides a more objective benchmark for mechanism evaluation. Additionally, measuring the market efficiency across a multi-period simulation changes the notion of mechanism to one which may more accurately model real-world domains.

After describing TAC SCM game as implemented for the 2005 TAC SCM competition,[1] I present an efficiency metric developed by Jordan et al. (2006) which was used to evaluate the 2005 competition. I describe the basic assumptions that underlie this efficiency calculation, and present a brief sketch of the calculation itself.

Next, I describe the integration of multiattribute markets into the TAC SCM simulation, denoting this modified simulation environment as *TACSCM-MA*. I describe the adaptation of the simulation software to support AB3D, as well as the AB3D implementation. I present the AB3D auction rules and bidding language for this domain, as well as some specialized AB3D features which economize somewhat on the computational complexity of the simulations.

I present my own entry in the 2005 TAC SCM competition, `GoBlueOval`, which I use as a baseline agent strategy for simulations. After describing the implementation of `GoBlueOval` in the existing TAC SCM environment, I present multiattribute bidding strategies developed for the modified simulation environment.

After presenting preliminary results which are somewhat negative (i.e., the multiattribute markets yield *lower* efficiency than the existing RFQ-based mechanism), I revisit the fundamentals of market-based algorithms and the technical assumptions on bidding behavior and price adjustments which underlie the efficiency guarantees for market-based algorithms. In analyzing the subtleties of employing market-based algorithms, I present a new stylized subproblem which is more faithful to the daily allocation problem found in TAC SCM. I show that the higher quantities of agent demand found in the daily TAC SCM PC market introduce new complexities absent from the model of Section 4.4.1. After finding bidding strategies for this subproblem which generate improved efficiency given my information feedback policy, I apply the optimal subproblem strategy to the full simulation and demonstrate an efficiency advantage over the existing TAC SCM market design.

5.1 The Trading Agent Competition Supply Chain Management Game

To participate in TAC SCM, competitors submit entries in the form of *trading agents*, autonomous software entities which compete in a game by making procurement, production, and selling decisions over the course of 220 simulated days. Agents compete with each other in procuring from a limited supply of components and securing a limited number of customer sales contracts.

[1] The TAC SCM rules often change slightly from year to year.

Each day of a TAC SCM simulation lasts 15 seconds, during which time the manufacturing agents must make choices for the following activities:

- bidding on customer RFQs
- sending RFQs to suppliers
- accepting supplier offers
- manufacturing PCs from existing component inventory
- completing orders by shipping from inventory of manufactured PCs

Each day, customers issue requests for quotes for finished PCs and select winning bids from among the prior day's set of submitted manufacturer bids. Manufacturers are free to bid on any or all of a given day's customer RFQs, by submitting a price at which they will provide the given order. Manufacturing agents are limited in production by a fixed daily assembly line capacity and available component inventory. Available component inventory is maintained by procuring components from a set of eight simulated suppliers.

The following sections describe specifics of TAC SCM that are relevant to my analysis, but I do not provide a full game description. For a complete specification of the TAC SCM rules and simulation environment, see the description provided by Arunachalam et al. (2004).

5.1.1 The Goods

Each PC requires four unique components: memory, hard disk, CPU, and motherboard. The CPUs come in two different "families" (`Pintel` and `IMD`), each family with two different speeds, for a total of four different CPU types. Motherboards are available in each family type, and are incompatible with CPUs of the alternate family type (i.e., the CPU and Motherboard in a PC must be of the same family). Memory and hard disks are each available in two sizes, compatible with both family types. The complete list of PC parts is presented in Table 5.1.

Constraining the CPU and motherboard to agree on family but otherwise allowing for arbitrary combinations of components yields 16 unique "builds." Each build additionally requires a specific quantity of production cycles to manufacture. The TAC SCM bill of materials shown in Table 5.2 specifies the complete set of builds, including required production cycles for each build. Table 5.2 additionally identifies the *market segment* for each SKU, which identifies one of three underlying stochastic processes from which the customer demand is drawn each day.

Component	Base price	Supplier(s)	Description
100	1000	Pintel	Pintel CPU, 2.0 GHz
101	1500	Pintel	Pintel CPU, 5.0 GHz
110	1000	IMD	IMD CPU, 2.0 GHz
111	1500	IMD	IMD CPU, 5.0 GHz
200	250	Basus, Macrostar	Pintel motherboard
210	250	Basus, Macrostar	IMD motherboard
300	100	MEC, Queenmax	Memory, 1GB
301	200	MEC, Queenmax	Memory, 2GB
400	300	Watergate, Mintor	Hard disk, 300GB
401	400	Watergate, Mintor	Hard disk, 500GB

Table 5.1: List of TAC SCM components.

SKU	Components	Cycles	Market segment
1	100,200,300,400	4	Low
2	100,200,300,401	5	Low
3	100,200,301,400	5	Mid
4	100,200,301,401	6	Mid
5	101,200,300,400	5	Mid
6	101,200,300,401	6	High
7	101,200,301,401	6	High
8	101,200,301,401	7	High
9	110,210,300,401	4	Low
10	110,210,300,401	5	Low
11	110,210,301,401	5	Low
12	110,210,301,401	6	Mid
13	111,210,300,401	5	Mid
14	111,210,300,401	6	Mid
15	111,210,301,401	6	High
16	111,210,301,401	7	High

Table 5.2: TAC SCM bill of materials.

5.1.2 Component Market

Agents procure components by issuing RFQs to one of eight suppliers. The two CPU suppliers specialize in one CPU family, while other suppliers provide both types of a single component.

Each day, each agent may send up to five RFQs to each supplier for each of the products offered by that supplier, for a total of ten RFQs per supplier. Each RFQ represents a request for a specified quantity of a particular component type to be delivered on a specific date,

but only if the quoted price is no higher than a reserve price. Once the supplier computes prices, a bid will be generated for each RFQ, having a possibly reduced quantity. If the reserve price cannot be met, then the quoted quantity will be 0. On the following day, the supplier sends back to each agent an offer for each RFQ, containing the price, adjusted quantity, and due date. The manufacturing agents may then choose to accept any subset of supplier offers.

Suppliers determine offer prices based on the ratio of free capacity to expected total production capacity. Roughly,[2] the offered price is computed as:

$$P_{base}\left(1-\delta(\frac{C_{avail}}{C_{total}})\right)$$

where P_{base} is a constant baseline component price (defined for each component), C_{avail} is the expected value of uncommitted production capacity (up to the requested due date), and C_{total} is the expected value of total production capacity (up to the requested due date). Both expectations are derived using the current day's capacity as the expected future daily capacity. While suppliers are not perfectly rational in their offer pricing, the crude capacity signals conveyed through component prices provide agents with some idea of the relative availabilities of alternate components. Agents use some of their allotted supplier RFQs to query the state of suppliers by sending *price probes*. Price probes are RFQs requesting zero quantity for a specific date. Suppliers will respond to price probes with a price for the requested offer date for an order of zero quantity.

5.1.3 PC Market

Customer demand is expressed as requests for quotes (customer RFQs), each of which specifies a product type, quantity, due date, reserve price, and penalty for late delivery. The specific instantiation of values for a given RFQ is determined stochastically. The product type is randomly chosen from the available types, quantity is chosen uniformly from $[1,20]$, and the due date is chosen uniformly from $[3,12]$ days in the future. The reserve price ranges from between $[75,125]$ as a percentage of the baseline component prices, and late delivery penalty is a percentage of the reserve price, chosen uniformly from $[5,15]$.

The customer will submit N RFQs on day d, where N is distributed according to the poisson distribution, $Poisson(Q_d)$, with Q_d defined by a random-walk-following trend. The more highly competitive agents in recent TAC SCM tournaments reason with respect to the

[2] This captures the idea behind supplier pricing. See the full game description for the exact supplier pricing calculation.

expectation of Q_d in anticipating future customer demand trends (Kiekintveld et al., 2007).

At the start of each day, each manufacturing agent is sent the complete set of customer RFQs for that day. Agents may then submit bids for any of the RFQs for that day by sending a bid amount that is weakly less than the customer reserve price, thereby committing to supply the entire requested quantity on the specified delivery date if awarded the contract, or incur the specified late penalties. Each customer collects the set of bids submitted by all agents, selecting the agent with the lowest bid for each RFQ to be the chosen supplier, with ties broken at random. This allocation process is equivalent to a set of independent sealed-bid auctions, one for each RFQ.

5.2 Gross Substitutes Violations

Recall that in Section 4.4.1, I generated bidder valuations from a given configuration structure (i.e., bill of materials) by sampling costs and inventories, and classified the induced valuations according to *EGSV* value. I then simulated bidding with a set of such valuations, and presented a plot demonstrating a correlation between efficiency and average *EGSV* value of a problem instance, aggregating data across multiple configuration structures.

Taking random samples of inventory, and omitting consideration of production cycles, due dates, and penalties for the time being, we can classify this bill of materials within the same framework introduced in Section 4.4. Figure 5.2 depicts the cumulative distribution function of *EGSV* values for 400 randomly generated valuations induced by the TAC SCM bill of materials, taking component costs and inventory samples as defined in Section 4.4.1.

We can also perform the same market efficiency simulation as that from Section 4.4.2, computing the expected efficiency as a function of the average *EGSV* value, but using only the TAC SCM bill of materials. Results from 390 bidding simulations are depicted in Figure 5.3, where each simulation takes a contiguous set of 10 *EGSV* values from those depicted in Figure 5.3.

The addition of cycle constraints, due dates, and penalties, complicates the bidding problem, and has an effect on agent valuations, but should not affect the *EGSV* of the agent valuations. While different production cycle requirements preclude configurations from being *perfect substitutes*, i.e., they are not exchangeable on a 1-to-1 basis, cycle constraints do not impose new violations of the gross substitutability of the configurations. The cycle constraints do have an impact on the effectiveness of sealed-bid multiattribute mechanisms, as I demonstrate in Example 8 of Section 5.7.

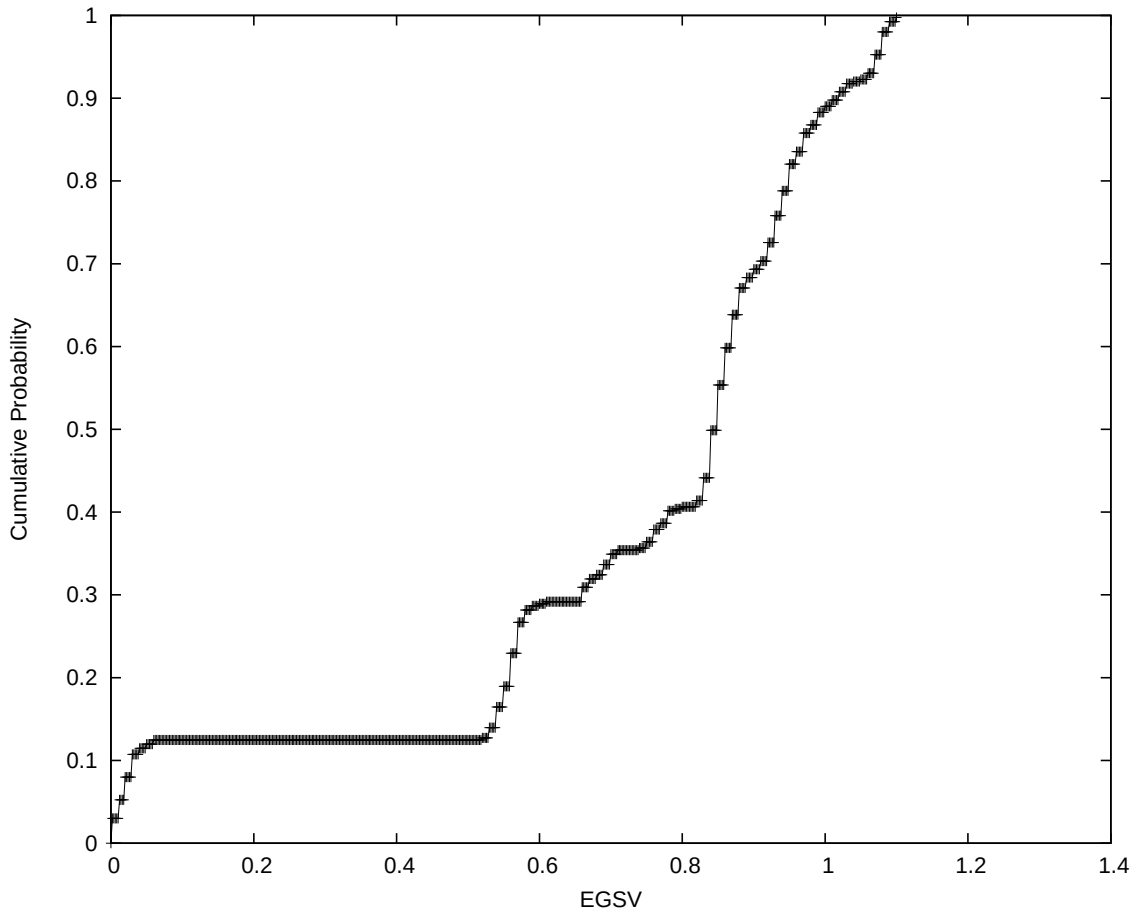

Figure 5.2: CDF of *EGSV* values for the TAC SCM bill of materials, using 400 valuations generated with random inventory and costs.

5.3 TAC SCM Market Efficiency

To evaluate the efficiency of the TAC SCM market, the entire supply chain from supplier to customer is treated as a centralized optimization problem, where the objective is to maximize the sum of realized agent utilities. As discussed in Section 1.2, transfer payments leave global utility unaffected, so the objective is then to maximize the sum of the realized agent valuations. The prices paid for goods and any profits generated by manufacturers net out to zero, and the market efficiency is a measure of the fraction of the maximal true surplus achieved during the simulation. Loosely, this measures how well the limited resources of the system are allocated in meeting customer demand.

In formulating the objective function, we then need to consider the valuations of all the agents involved. In measuring the overall efficiency of the TAC SCM market, I employ a software tool developed by Jordan et al. (2006). I borrow their model in making the following assumptions about agent valuations:

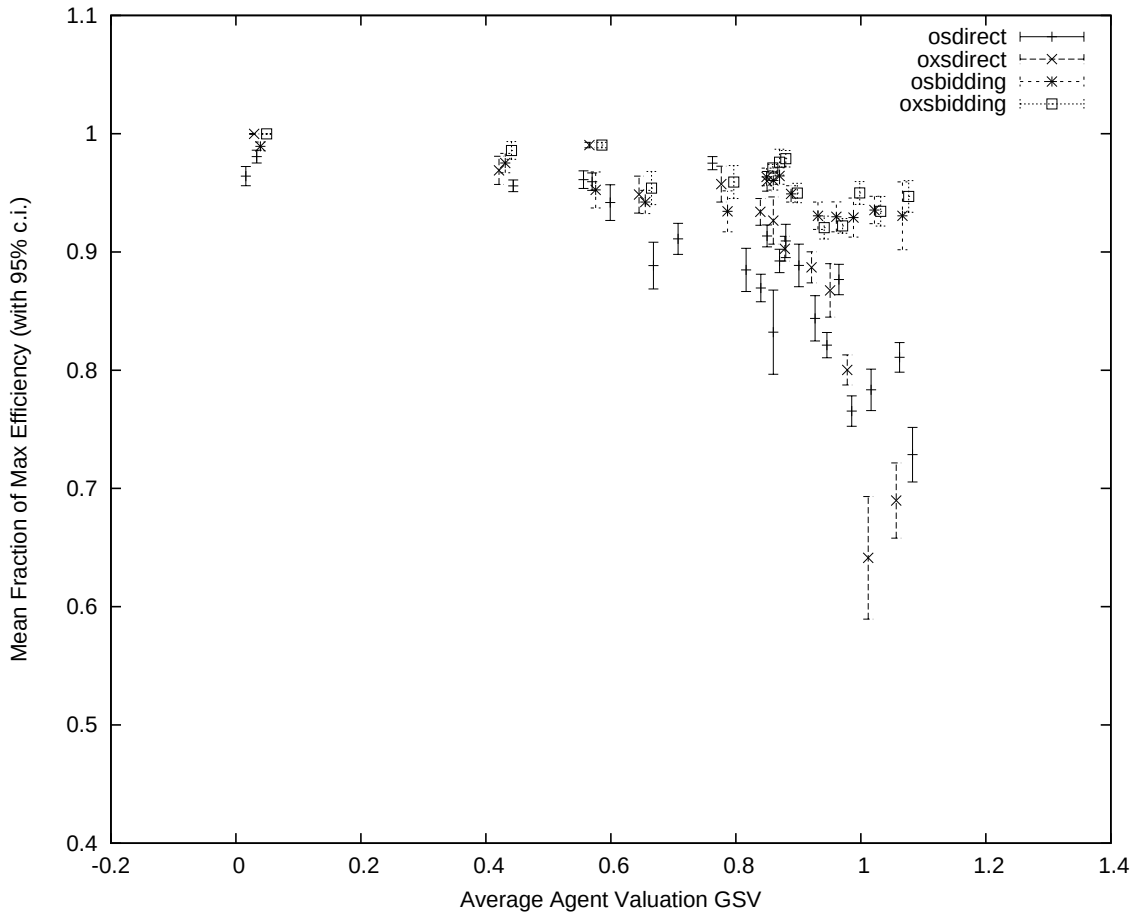

Figure 5.3: Relationship between efficiency and *EGSV* for the TAC SCM bill of materials with 95% confidence intervals. Each problem instance used 10 contiguous valuations from those depicted in Figure 5.2.

- **suppliers** Suppliers are assumed to incur linear component costs equal to the discounted base price per component, $P_c^{base}(1-\delta)$, up to their realized production capacities.
- **customers** Customers derive value for satisfied orders equal to the face value of RFQs, less the costs of any penalties incurred. This assumption is equivalent to assuming that customers are truthful in submitting RFQ reserve prices.
- **manufacturers** Manufacturers are assumed to incur only storage costs, but are limited in their production capacities at 2000 cycles per day.[3]

The optimization can be framed from the standpoint of the optimal set of filled RFQs, A, where the filled RFQs on any day d is A_d, which must be a subset of the customer RFQs

[3]In the simulation, manufacturers "supply" production capacity, inventory storage, and computation directed toward decision making. While all of these resources would have value in a real-world environment (likely equal to the variable cost of operation), within the simulation environment manufacturers incur only storage costs.

for that day, RFQ_d. The value of A_d, $\mathscr{V}(A_d)$, is simply the sum of unit reserve prices of the individual RFQs, multiplied by the respective RFQ quantities. The cost of A_d is the sum of the costs of required components, with each component valued at half its base price. I further define $Factory(A_d)$ and $Components(A_d)$, respectively, as factory cycles and components required to manufacture the PCs for A_d. Denoting the aggregate set of components (across all suppliers) that can be produced on day d by C_d, and the aggregate production capacity (across all manufacturers) on day d by F_d, the maximization of global surplus was formulated as the following integer program:

$$\begin{array}{ll} \max_{A_d | A_d \subset RFQ_d} & \sum_{d \in days} \mathscr{V}(A_d) - \mathscr{C}(A_d) \\ s.t. & \forall d \in days \quad \begin{cases} \bigcup_{i=0}^{d} Components(A_i) \subseteq \bigcup_{i=0}^{d} C_i \\ \bigcup_{i=0}^{d} Factory(A_i) \subseteq \bigcup_{i=0}^{d} F_i \end{cases} \end{array}$$

This formulation depicts the algorithm as implemented by Jordan et al.. This formulation yields only a close approximation to the true optimal surplus, as it does not require production capacity used for a given RFQ to be available on the same day as the parts required for that RFQ. That is to say, it allows for production capacity available on day d to be used in assembling PCs with components that are not available until day $d' > d$, so long as the cycle and inventory constraints are independently satisfied.

Jordan et al. (2006) applied their efficiency tool to the TAC SCM competition for years 2003-2005, finding a general trend toward higher efficiency across years, as well as higher efficiency in advanced tournament rounds. The efficiencies for each year are presented in Table 5.3, segregated into the quarterfinals (QF), semifinals (SF) and final (F) rounds.[4]

	QF	SF	F
2003	59.8%	59.1%	62.9%
2004	67.4%	75.9%	57.1%
2005	86.9%	87.6%	90.5%

Table 5.3: Average market efficiency for quarterfinals, semifinals, and finals of TAC SCM competition, years 2003–2005.

Citing the increasing trend in market efficiency, Jordan et al. hypothesized a positive correlation between efficiency and agent competence, but as noted by Wellman et al. (2003), not all advances in agent competence lend themselves to increases in market efficiency. A broad distinction is made in much of economic theory between *competitive*

[4]The efficiency values reported by Jordan et al. (2006) are $\sim$ 3% lower than those reported here, as their algorithm for calculating the maximal surplus mistakenly defined several configurations as not requiring a motherboard.

and *strategic* behavior (Mas-Colell et al., 1995). Competitive behavior, also called *price-taking* behavior, implies that agents observe market prices and optimize with respect to those prices. We can observe price-taking behavior behind the implementation of market-based algorithms, as agents myopically optimize with respect to prices. In contrast, when engaging in strategic behavior, agents account for their own effects on prices as well as their effects on other agents when formulating strategies.

Given that finding an optimal allocation for a game instance is NP-hard, it should be expected that finding strategies that maximize efficiency will be a challenging problem, even when dictating all agent strategies (i.e., even when including non-equilibrium strategy profiles). A goal in formulating a market design for any environment is that competitive behavior (i.e., optimal agent strategies) should lead to efficient outcomes while also being robust to strategic behavior.

In observing the increasing efficiency trend in successive years of the TAC SCM competition, we can argue that the RFQ-based negotiation process is a good mechanism—as agents have become more competent, the market efficiency has increased. The question I address here is whether a multiattribute market is able to improve on this mechanism.

5.4 Multiattribute Simulation Implementation

I modified the TAC SCM simulation environment to support the negotiation of customer orders through a single multiattribute market. In the modified environment, the AB3D system is invoked at the beginning of each simulation, and runs in parallel throughout the simulation. Each day, rather than sending RFQs to the manufacturing agents, customers translate each RFQ into an equivalent multiattribute offer (the set of all RFQs collectively comprising a class *OS* bid, in the framework from Section 4.1.4) and send the aggregate set of offers to the AB3D system. Manufacturers likewise send multiattribute bids to the AB3D system. At the end of each simulation day, the SCM simulation sends a `clear` trigger message to the AB3D system, subsequently downloading any transactions generated in the clear operation. The transactions are then translated back into orders, which are sent to the respective trade parties of the transaction.

5.4.1 Scenario Modifications

To enable a fair comparison between the original game formulation and my multiattribute implementation, I kept all exogenous variables intact, and replaced only the manufacturer-

customer negotiation process. As the clearing algorithm requires that bids be divisible, I similarly had to require that RFQs be divisible in the multiattribute implementation, meaning that a single RFQ may be partially filled, or filled by multiple agents. Although this is a possible source of bias in favor of the multiattribute implementation, the effect is relatively minor, as the quantity of each RFQ is small in relation to the aggregate quantity of PCs requested per day.

In order for the simulation environment to properly credit a shipment made in fulfillment of an order, messages designating an RFQ, bid, and order must all have been sent between the trade parties. Therefore, as a matter of bookkeeping, a set of RFQs is retroactively generated which is equivalent in aggregate to the initial set of customer RFQs, but with quantities amended so that quantity from a single RFQ is not split among multiple orders. A set of manufacturer bids is then generated which correspond with the transactions and orders generated by the AB3D system.

For the iterative implementation, at the end of each day, subsequent to each bid submission, the AB3D system computes quotes for each (SKU, day). The simulation invokes manufacturing agents serially during a bidding round, allowing them to submit bids based on quote information incorporating all prior bids.

5.4.2 AB3D Auction Implementation

The following sections describe the three dimensions of the AB3D auction specification: bidding language, clearing policy, and information revelation policy.

Bids

Multiattribute bids specify the SKU, due date, penalty, quantity, and reserve price. I used the following mapping to attribute ID numbers:

due date	1
penalty	2
SKU	3
RFQ ID	4

The RFQ ID is necessary for the bookkeeping within the SCM simulation, as the simulation requires an existing RFQ which corresponds with each order. Manufacturers omit the RFQ ID, allowing for a match on any RFQ ID value.

For customer bids, each RFQ is directly translated into a single offer set, where the set of all RFQs for a given day are aggregated into a single bid. The clearing algorithm does not support indivisible bids, so each equivalent multiattribute offer must be made divisible. To support this, penalties are translated from an aggregate penalty (for the entire RFQ) into a unit penalty (penalty per unit quantity). Figure 5.4 depicts a single RFQ translated into an offer set of quantity 8, with due date of 115, penalty of 119 (originally $119 \cdot 8 = 952$ in the RFQ), SKU 10, RFQ ID of 480, and reserve price of 2397.0.

```
<OS>
  <Q>8</Q>
  <O>
    <P>2397.0</P>
    <FCS>
      <A ID="1" T="1"><V>115</V></A>
      <A ID="2" T="1"><V>119</V></A>
      <A ID="3" T="1"><V>10</V></A>
      <A ID="4" T="1"><V>480</V></A>
    </FCS>
  </O>
</OS>
```

Figure 5.4: Customer RFQ translated into a multiattribute offer.

Manufacturer offers may specify a range of possible values for due dates, and may specify multiple offers within an offer set. The following offer set of quantity 8 specifies two alternate configurations, either configuration 2 at a price of 995, valid over the date range of $[108, 120]$, or configuration 10 at a price of 1130.0, also valid over dates $[108, 120]$.

```
<OS>
  <Q>-8</Q>
  <O>
    <P>995.0</P>
    <FCS>
      <A ID="1" T="1"><R><V>108</V><V>120</V></R></A>
      <A ID="3" T="1"><V>2</V></A>
    </FCS>
  </O>
  <O>
    <P>1330.0</P>
    <FCS>
      <A ID="1" T="1"><R><V>108</V><V>120</V></R></A>
      <A ID="3" T="1"><V>10</V></A>
    </FCS>
  </O>
</OS>
```

Figure 5.5: Multiattribute manufacturer offer.

The above manufacturer and customer bids would match on configuration 10 and due date 115. Since the manufacturer bid omitted the RFQ ID and the penalty amount, the bid will match against any values for those attributes. The resulting match between the two offer sets would have unit penalty of 119 and RFQ ID of 480. The surplus of this match (i.e., the solution to MMP_s which would be used in the optimization problem of the clear operation) would be $2397.0 - 995.0 = 1067$.

Clearing

The timing of the clear is left undefined within the AB3D auction script, as all clears are generated by trigger messages sent from the simulation. Figure 5.6 provides an example of a clear message, which instructs the AB3D system to clear at unix time of 1204779620000 (equivalently, 2008-03-06 00:00:20), and to subsequently clear all bids from the order book. In order to send such trigger messages, the simulation authenticated with the AB3D system at the start of the game using an administrator password, receiving an agent ID of 0.

```
<Trigger>
  <auctionID>1</auctionID>
  <agentID>0</agentID>
  <TimeStamp>1204779620000</TimeStamp>
  <action>clear</action>
  <action>flushBids</action>
</Trigger>
```

Figure 5.6: Example trigger message to clear the AB3D auction.

Information Feedback

The general purpose multiattribute order book supports only query-based information feedback, as the complexity of computing quotes for all configurations is prohibitive in the general case. In the case of the SCM simulation, the quote-based bidding algorithm used by agents employs all available quote information, i.e., all quotes for (SKU,day) pairs. To economize somewhat on communication between AB3D and agents, I extended the existing order book with a quote operation that computes bid quotes for all (SKU,day) pairs, ultimately computing a total of 160 bid quotes per quote operation. The AB3D auction script for the SCM simulation includes a rule to quote on any successful bid submission, computing bid quotes for all (SKU,day) pairs and sending the aggregate quote to the cache. The SCM simulation then polls for an updated quote after each agent's bidding round, providing the updated quote to the next bidder. Agents thus are able to reason based on quote

information that incorporates all previously submitted bids.

5.5 Baseline Manufacturing Agent

The baseline manufacturing agent is derived from my entry in the 2005 TAC SCM competition, `GoBlueOval`.[5] On a high level `GoBlueOval` behaves in an approximately price-taking manner, purchasing components for any price less than their estimated marginal cost, and selling computers at prices greater than estimated sum of component costs. The bids `GoBlueOval` places for customer RFQs are determined by available inventory and production capacity, in conjunction with empirical distributions which model the probability of winning an order. Although the PC market bidding is the primary focus in this chapter (as well as the only modification across the standard and multiattribute treatments), for completeness I give a sketch of the full `GoBlueOval` behavior in the following sections.

5.5.1 Component Procurement

`GoBlueOval` employs a very simple heuristic in maintaining its component inventory. `GoBlueOval` uses static target levels of inventory as a function of the time into the future. At the end of each day, `GoBlueOval` computes an expected inventory for the next 30 days, taking into account open supplier orders and open customer orders. In deciding which RFQs to send to suppliers, `GoBlueOval` scans over the next 30 days in increments of 3 days, sending an RFQ with quantity set to the expected inventory deficit, up to a maximum quantity of 300. The reserve price for each supplier RFQ is set to the approximated marginal value of the component, averaged over all PCs that contain the component.

5.5.2 Manufacturing Scheduling and Shipping

Each day, `GoBlueOval` builds and ships PCs for as many open customer orders as possible, subject to inventory and production capacity constraints. In selecting orders to build, `GoBlueOval` first produces PCs for any orders that are due in two days (i.e., they will be delivered on time if produced today, but will be late if produced tomorrow). Next, any late

[5] GoBlueOval was the 2nd place agent in the seeding round, but fared poorly in the quarterfinal round due to a bug arising only in the presence of specific opponent strategies. `GoBlueOval` was developed with the help of Yagil Engel, Daniel Kuo, and Maurice Solomon, and also with the support of Ford Motor Company.

orders that can still be delivered before cancelation are produced. Finally, any open orders due later than two days in the future are produced.

Orders can be shipped if the agent has adequate inventory of built PCs. Order shipping is performed in the same sequence, where orders due in two days with sufficient PC inventory will be shipped first, followed by late orders, followed by orders due more than two days in the future.

5.5.3 Customer RFQ Bidding

Without the uncertainty inherent in customer bidding, the daily customer bidding problem reduces to a knapsack problem: bid on the set of RFQs generating the most profit, subject to inventory and capacity constraints. `GoBlueOval` simplifies the problem by ignoring inventory constraints and relaxing integrality constraints, thus solving the relaxed (fractional) knapsack problem of maximizing profit subject to cycle constraints. Computing the solution to a fractional knapsack problem is fairly simple: starting with the highest density item, add items of decreasing density until the capacity has been reached. In the case of maximizing profit subject to production capacity constraints, the "density" is the profit per cycle (which I denote by *surplus per cycle* or *SPC*) of any given RFQ. For example, an RFQ with an estimated cost of $500, requiring 5 production cycles per unit, would have a surplus per cycle of $\frac{p-500}{5}$ per unit as a function of the bid price, p. The estimated cost for each PC is estimated as the sum of component costs, where the cost for each component is estimated by sending a single price probe 11 days in the future.

This problem is complicated by the uncertainty of winning any single RFQ. Both the probability of winning and the surplus per cycle are bid-dependent. `GoBlueOval` approaches the problem probabilistically and finds the profit-maximizing *SPC* to bid for all RFQs subject to production capacity constraints. Denoting the *SPC*-dependent probability of winning a customer order by $\Pi_w(SPC)$, `GoBlueOval` solves the following optimization:

$$\begin{array}{rl} \max_{SPC} & \sum_{rfq \in RFQs} \Pi_w(SPC) \cdot SPC \cdot cycles(rfq) \\ s.t. & \sum_{rfq \in RFQs} \Pi_w(SPC) \cdot cycles(rfq) < capacity \end{array}$$

Bid Win Distribution

`GoBlueOval` models the probability of winning a given RFQ through empirically updated distributions, which map from *SPC* to the probability of winning. A unique distribution is

maintained for each lead time, yielding 10 such distributions: 1 each for lead times of 3 through 12. The distributions are updated each day based on the observed win probabilities of the prior day's bids.

Customer Bidding Optimization

Given empirical distributions, `GoBlueOval` calculates a profit-maximizing surplus to bid on each lead time, given $(lead - 3)$ additional days for which to bid in satisfying a given day's production capacity. Taking the current day's set of customer RFQs as the expected set of RFQs that will be observed in future days, `GoBlueOval` finds the single level of *SPC* which would maximize profits, subject to capacity constraints.

Bidding

`GoBlueOval` then processes all of the day's RFQs in decreasing order of maximum possible *SPC* (i.e., $\frac{reserve-cost}{cycles}$, where cost is estimated through supplier price probes), placing a bid with unit price of $(cost + SPC \cdot cycles)$, until the inventory and manufacturing capacity constraints are met probabilistically (using the bid win distribution to estimate probabilities).

To maintain better accuracy in the bid distributions, the *SPC* used for any given bid is perturbed stochastically between $[SPC - 20, SPC + 20]$. Given the perturbed *SPC* for an RFQ, `GoBlueOval` computes a bid as $b^* = cost + SPC \cdot cycles$, submitting a bid if the computed bid value is below the RFQ reserve price. The final bid amount is again perturbed stochastically between $[b^* - 5, b^* + 5]$, to avoid predictability in bidding behavior.

5.6 Bidding Strategies

As discussed in Section 5.3, the market efficiency of a TAC SCM game depends critically on agent strategies. The work by Jordan et al. (2006) showed that not just the overall competence of agents, but also the mix of agents in a given game has an influence on market efficiency.

Efficiency results from the annual TAC SCM competition provide an indication of the expected efficiency of TAC SCM under maximally competitive agents, given that significant efforts are devoted to agent design, with those efforts coming from a diverse group of sophisticated competitors. The creation of a multiattribute TAC SCM competition would

present a valuable measure of multiattribute market efficiency in this domain. Lacking such a competition to spur agent strategy design, I employ variations of `GoBlueOval`, an agent that was a top competitor in early 2005 TAC SCM rounds. Using the existing `GoBlueOval` implementation as a second RFQ-based efficiency benchmark, I modify the customer bidding algorithm with myopic multiattribute bidding strategies. While results from such a restricted space of bidding strategies will not provide conclusive evidence for the relative efficiency of multiattribute markets, I argue that the use of naïve multiattribute bidding strategies provides evidence for the expected efficiency given agents that are more strategic in nature.

The following sections describe various bidding strategies for both the standard and multiattribute bidding simulations. For all of the following bidding strategies, the component ordering, manufacturing, and shipping algorithms are identical, with variation in customer bidding strategies. For multiattribute bidding strategies, all agents bid their estimated cost on all SKUs, i.e., only the set of goods on which to bid varies across the different implementations.

5.6.1 GoBlueOval Random

This strategy simply bids a random value between cost and the reserve price of the RFQ, where the RFQs are processed in decreasing order of maximum possible *SPC*. This strategy uses the bidding distribution, but only to determine when to stop bidding on RFQs (i.e., when inventory or cycle capacity has been exhausted in expectation).

5.6.2 GoBlueOval Cost

This strategy uses an inventory-dependent cost in determining the bid level, subsequently computing the probability of winning the RFQ based on the cost as defined in the baseline customer bidding algorithm (i.e., the cost as estimated through supplier price probes). This strategy uses the bidding distribution, but only to determine when to stop bidding on RFQs (i.e., when inventory or cycle capacity has been exhausted in expectation).

5.6.3 Multiattribute Direct

In the direct multiattribute implementation, agents formulate bids without receiving any quote information. Agents use a simple heuristic in formulating direct multiattribute bids.

Starting with lead time 3, the agents generate up to 2 offer sets for each lead time, iterating over all lead times up to 12. For each lead time, the agents first generate an offer set by iterating over all SKUs, including all SKUs to the offer set for which the maximum possible production of the SKU is within 100 of the maximal production of any SKU on that day. This offer set is added to the bid, inventory is debited, and a second pass is made for the same lead time. This process is iterated for each subsequent lead time, thus generating a multiattribute bid with up to 20 offer sets. For each offer set, the penalty is not specified in the bid (allowing a match on any penalty amount), while the price is set to the estimated sum of component costs for each SKU.

5.6.4 Iterative Multiattribute Bidding

In the iterative implementation, each agent receives updated quote information at the beginning of its allotted bidding round. The agent takes this quote information and computes an optimal feasible production set. The following linear program computes the optimal feasible production set given bid quotes (denoted by *bq*) by maximizing surplus subject to inventory and capacity constraints:

$$\begin{array}{ll} \max & \sum_d \sum_{SKU}(bq_{SKU,day} - cost_{SKU}))q_{SKU,day} \\ s.t. & \forall d \in days \quad \begin{cases} \sum_{SKU} \sum_{d=0}^{day} cycles(SKU) q_{SKU,d} \leq \sum_{d=0}^{day} cycles_d^{avail} \\ \forall comp \sum_{SKU|comp \in SKU} \sum_{d=0}^{day} q_{SKU,d} \leq \sum_{d=0}^{day} inventory_{comp,d} \end{cases} \end{array}$$

The production quantities which solve the above linear program constitute a feasible production set, which can be formulated as a feasible *OS* bid. The agent takes this feasible *OS* bid and expands it to a feasible *OXS* bid by iterating over each SKU in the *OS* bid, and adding all other SKUs with sufficient inventory and equivalent production cycles.

5.7 Simulation Results

I first present results for the alternative `GoBlueOval` strategies. The CDF and mean efficiencies are plotted in Figure 5.7 and Figure 5.8, respectively, with sample sizes of 188 for `GoBlueOval` (depicted as gbo), 69 for randomized `GoBlueOval`(depicted as gbo_r), and 54 for cost-bidding `GoBlueOval` (depicted as gbo_c).

The version of `GoBlueOval` submitting random bids (above cost, and probabilisti-

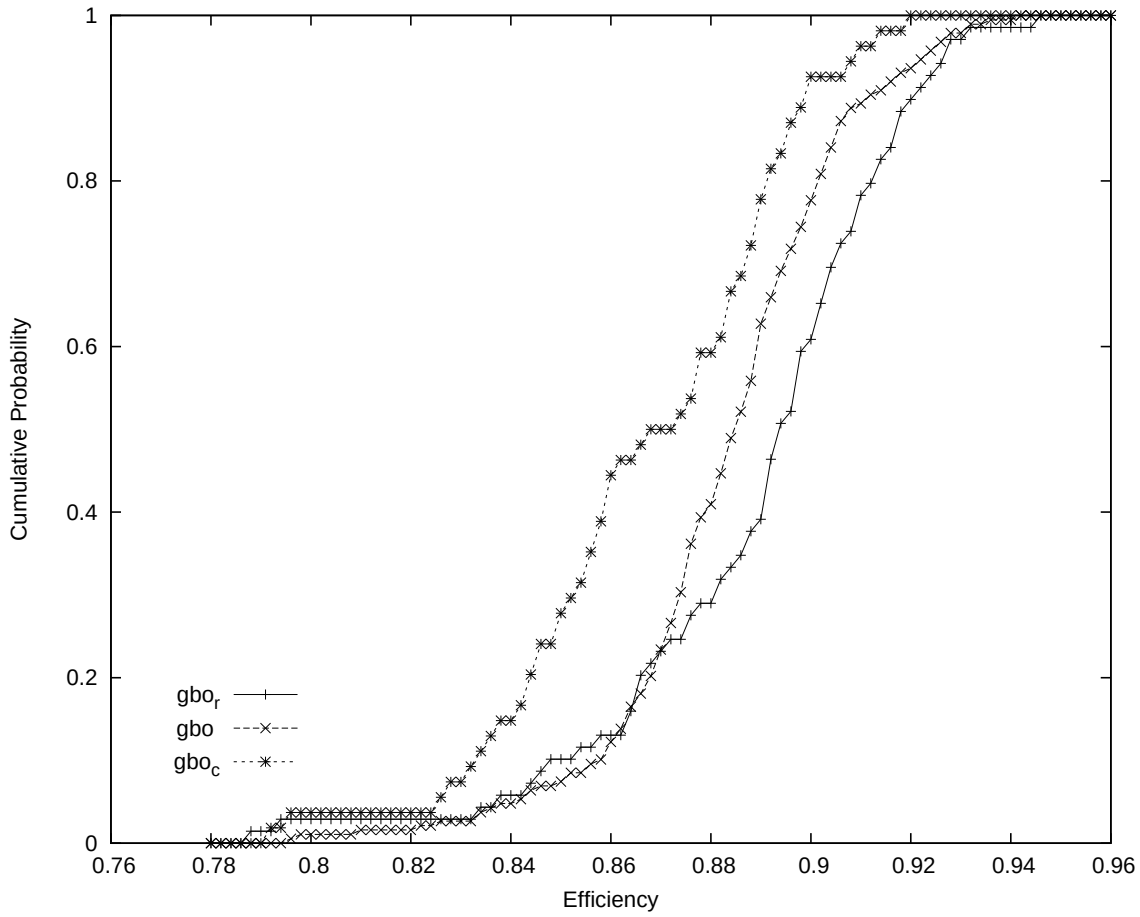

Figure 5.7: CDF of realized efficiencies of `GoBlueOval` strategies for baseline (gbo), randomized (gbo_r), and cost-bidding (gbo_c) implementations.

cally filling inventory) yielded higher efficiency than the baseline `GoBlueOval`, while the version of `GoBlueOval` submitting bids of true cost fared the worst. A possible explanation for the superior performance of the randomized implementation is that the baseline implementation implements strategic demand reduction to a certain degree, in that the profit-maximizing *SPC*may not fully utilize production capacity or inventory (at a price above cost). Although the randomized strategy is far from an equilibrium strategy, I will take the randomized version as the "best" RFQ-based implementation, and use that as my comparison benchmark for the multiattribute implementations.

Next, I compare efficiency for the randomized `GoBlueOval` implementation against the multiattribute implementations. Figure 5.9 plots the CDF of efficiency for each strategy, and 5.10 the mean efficiency with 95% confidence intervals. Sample sizes were 55 for iterative multiattribute (depicted as ma_i), 186 for sealed-bid multiattribute (depicted as (ma_d)) and 188 for randomized `GoBlueOval` (depicted as gbo_r).

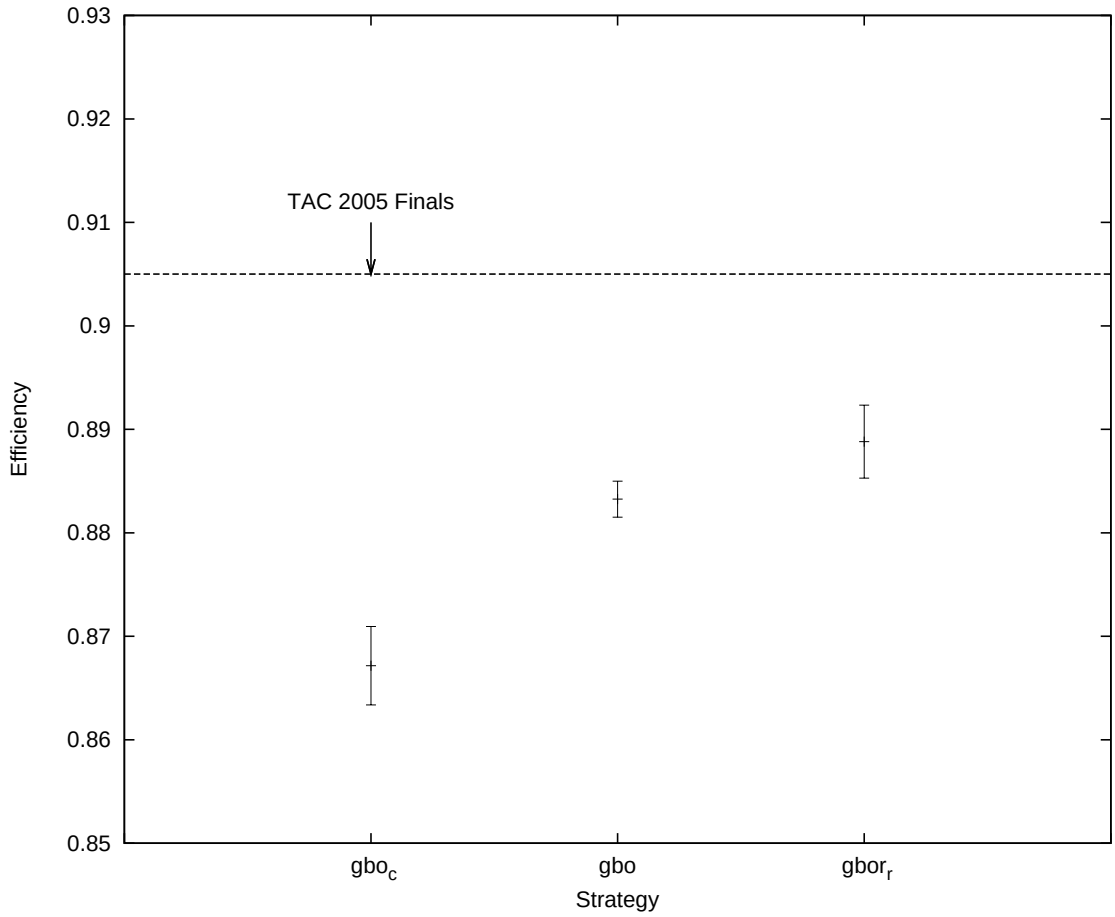

Figure 5.8: Mean realized efficiencies with 95% confidence intervals of `GoBlueOval` strategies for baseline (gbo), randomized (gbo_r), and cost-bidding (gbo_c) implementations. Average efficiency achieved in the final round of the 2005 TAC SCM competition is depicted by the dashed horizontal line.

The initial simulation results were somewhat surprising, in that both the direct and iterative multiattribute implementations fared poorly in comparison to the baseline agent. While both gave reasonable performance, yielding higher efficiency than the early rounds of the 2005 TAC SCM tournament, my hypothesis was that multiattribute bidding would lead to higher measures of efficiency when holding other agent behaviors (e.g., supplier ordering) constant.

We must bear in mind the different market information available in the multiattribute and standard settings. For the standard setting, the entire set of customer RFQs is made available to all agents, essentially disclosing the entire preference structure of the customer agent. In contrast, agents had no information about customer bids in the sealed-bid multiattribute setting,[6] whereas in the iterative setting agents had only limited knowledge of the

[6]Agents could potentially reason about the expected customer bid based on the customer demand process

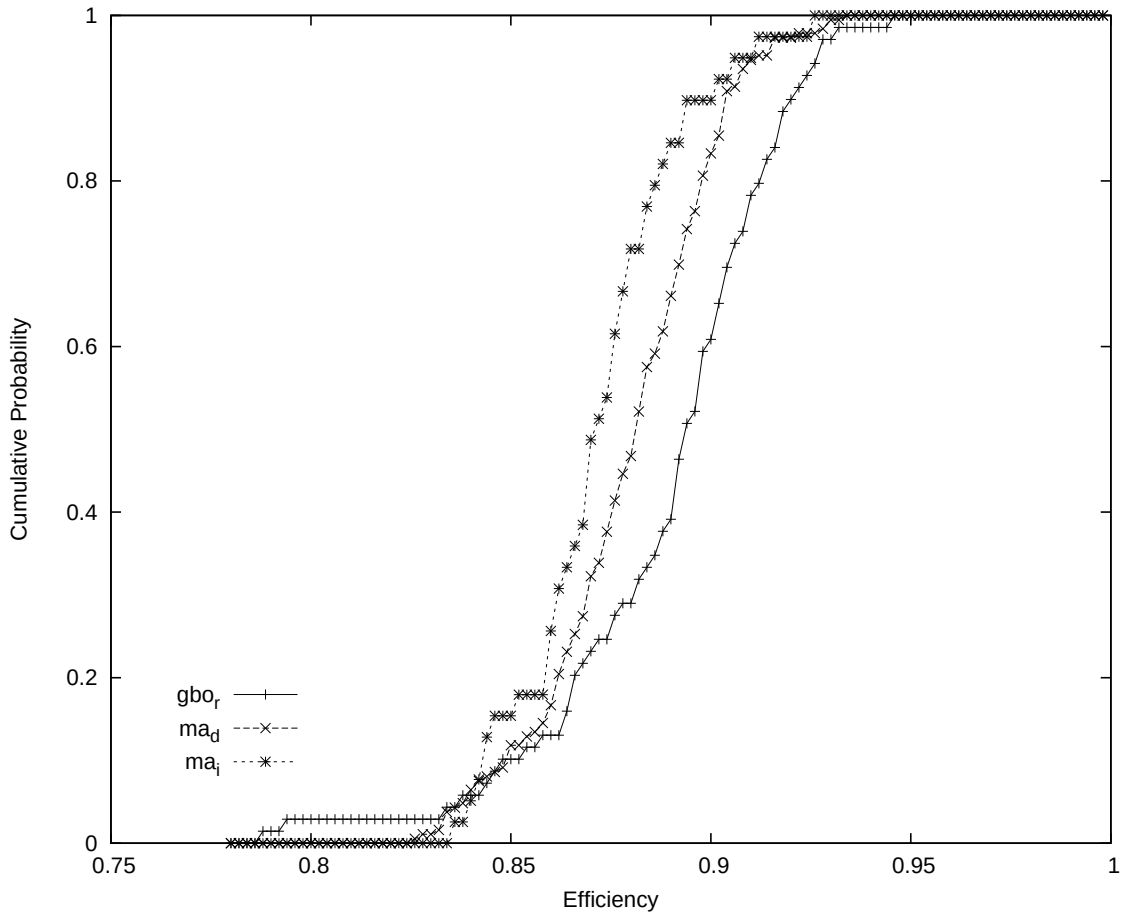

Figure 5.9: CDF of realized efficiencies for randomized `GoBlueOval` (gbo_r), sealed-bid multiattribute (ma_d), and iterative multiattribute (ma_i) implementations.

full set customer bids through price quotes. The direct multiattribute implementation could be considered a viable alternative to the existing protocol in that comparable efficiency obtains with improved information disclosure policies.

The difficulty for the direct revelation was presaged in the results from Section 4.5. Although the expected gross substitutes violation at the margin would be expected to be consistent for varying levels of inventory, the induced agent valuations become increasingly difficult to express in an *OXS* bidding language. The following example demonstrates how differing manufacturing cycle requirements across configurations prevents the direct expression of valuations with class *OXS* bids.

Example 8 *Consider a producer who can manufacture two unique goods, each requiring two unique parts, assembled in the following manner:*

given prior customer bids, but no such reasoning was implemented in these test agents.

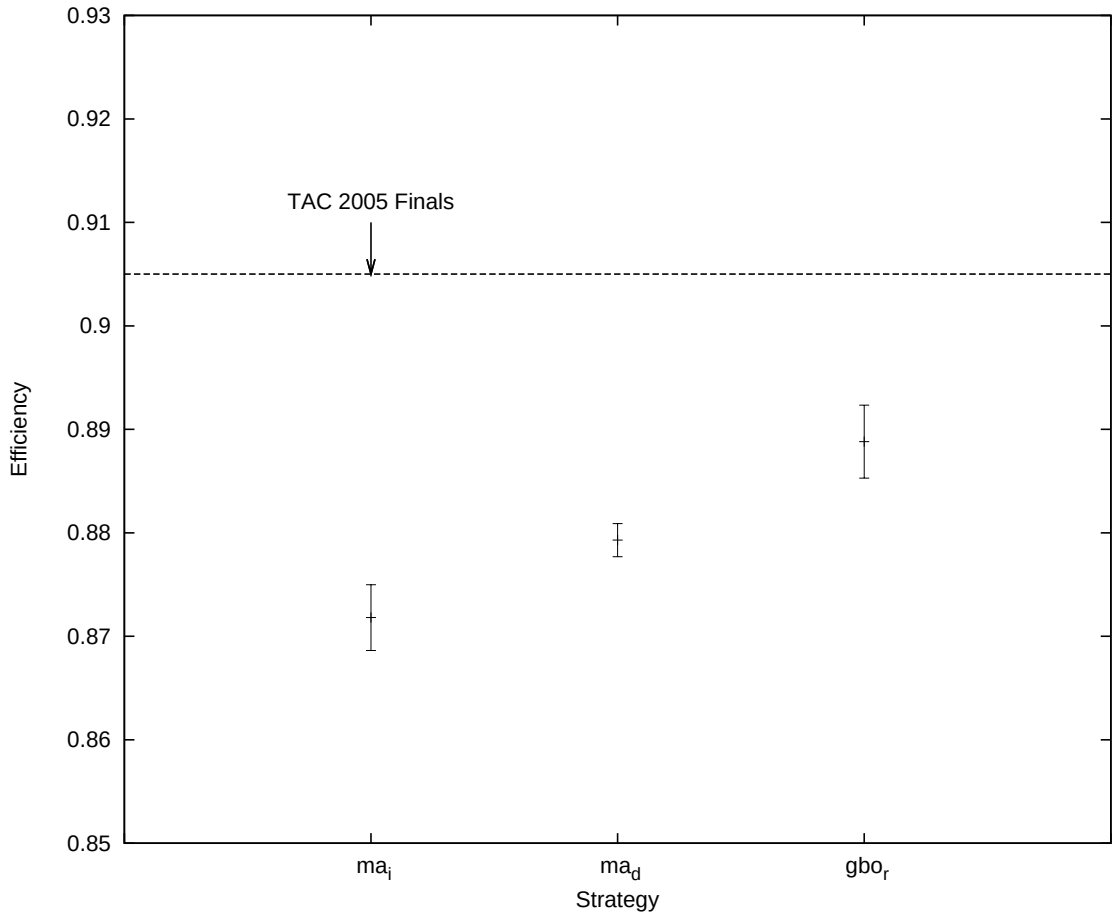

Figure 5.10: Mean realized efficiencies with 95% confidence intervals for randomized `GoBlueOval` (gbo_r), sealed-bid multiattribute (ma_d), and iterative multiattribute (ma_i) implementations. Average efficiency achieved in the final round of 2005 TAC SCM competition is depicted by the dashed horizontal line.

good	*components*	*cycles*
A	$\{1\}$	*1*
B	$\{1\}$	*2*

The goods require the same component, but different numbers of cycles. With two cycles available, the producer could produce two units of good A, or 1 unit of good B. These sets are not expressible in an OXR bid, because such a bid does not allow for the expression of alternate configurations of varying quantity, e.g., "2 of good A or 1 of good B."

Now consider what happens when the producer has 8 production cycles. He now has the following cycle-constrained maximized production sets:

- $\{(A,8)\}$
- $\{(A,6),(B,1)\}$
- $\{(A,4),(B,2)\}$

- $\{(A,2),(B,3)\}$
- $\{(B,4)\}$

If trying to maximally express production capabilities, the agent can then select among the following OXR bids:

- $\{(A,p_A,8)\}$
- $\{(A,p_A,6),((A,B),(p_A,p_B),1)\}$
- $\{(A,p_A,4),((A,B),(p_A,p_B),2)\}$
- $\{(A,p_A,2),((A,B),(p_A,p_B),3)\}$
- $\{(A,B),(p_A,p_B),4)\}$

In this example the number of feasible and unique OXR bids grows linearly with the number of production cycles. The optimal bid will depend on the submitted bids of other bidders, meaning that the bidding problem is made more difficult, as the probability of randomly submitting the optimal bid is decreasing in the number of available production cycles.

In Example 8, the difficulty of submitting an optimal sealed *OXR* bid increases despite not violating the gross substitutes condition. One would then expect a decrease in the efficiency of the sealed-bid (i.e., one-shot) implementation with respect to the expected efficiency implied by the prior sub-problem test.

The relative under-performance of the iterative multiattribute mechanism was more unexpected, as the test results from Section 4.5 indicated a high expected degree of efficiency, given the CDF of *EGSV* values presented in Figure 5.2 and the efficiency plot of Figure 5.3. To investigate the cause of this under-performance, in the following section I revisit the theory underlying market-based algorithms and address the extent to which my market design adheres to theoretical conditions.

5.8 Market-Based Algorithm Convergence

As discussed in Section 4.2.2, the gross substitutes condition is sufficient to guarantee the existence of a unique efficiency-maximizing price equilibrium, and also guarantees the equilibrium to be stable under a *tatonnement* price adjustment process. In this section I explain that the multiattribute information feedback process as presented in Section 3.3 does not implement a provably convergent price adjustment process.

5.8.1 Tatonnement Processes

The original conception of the tatonnement process is attributed to Leon Walras, who envisioned an iterative process whereby agents adjust their respective demand sets in response to market prices, while market prices adjust to balance supply and demand. Arguing that a change in a good's own price affects the agent's demand for that good to a greater extent than it affects the agent's demand for other goods, Walras concluded that the process would ultimately reach a market-clearing equilibrium.

Walras' original conception of equilibrium and tatonnement received more rigorous analysis from Samuelson (1947), who formulated a price adjustment process which ensures stability of equilibrium, and also from Arrow et al. (1959), who identified the condition of gross substitutes as sufficient to guarantee the existence of equilibrium. To enable the use of calculus-based proof methods, these early treatments make the assumption that goods and money are perfectly divisible.

Applying the gross substitutes condition to discrete settings, Kelso and Crawford (1982) consider a job-matching scenario in which firms make offers to workers in discrete periods, while workers accept the highest offer received in any given period. Kelso and Crawford show that the iterative process reaches a core allocation which coincides with a price equilibrium in salaries. They require that offers accepted in any given period remain in force in the subsequent period, i.e., firms cannot renege on accepted offers, although workers are free to switch firms upon receipt of a better offer.

Building on the work of Kelso and Crawford, Gul and Stacchetti (1999) show that gross substitutes is both sufficient and necessary for a Walrasian equilibrium in discrete settings, and show that the iterative upward perturbation of prices for all goods in excess converges to the smallest Walrasian price vector (Gul and Stacchetti, 2000).

In more recent work, Milgrom and Strulovici (2006) refine the concept of gross substitutes for settings in which agents have utility for multiple units of each good (alternatively, settings in which multiple copies of the same good are not priced uniquely). Milgrom and Strulovici show that when distinct units of the same goods are substitutes for each other (i.e., *class substitutes*), a monotonic price adjustment process with linear good prices will converge to a Walrasian equilibrium.

These above tatonnement formulations assume proportional price adjustment processes in which prices are perturbed proportionately to the current levels of excess demand. In intermediate stages of these tatonnement processes, prices generally do not reflect market-clearing conditions given agent demand sets. The WALRAS algorithm (Cheng and Wellman, 1998) implements a market-based algorithm more closely resembling my own. WALRAS is a distributed implementation of the tatonnement process, where agents submit

price-quantity demand schedules for each good, given current prices of other goods. The WALRAS algorithm, to my knowledge, is the only example of a tatonnement procedure in which market-clearing prices are computed at each iteration of the process, as opposed to a proportional price adjustment process. Cheng and Wellman also demonstrate convergence of a approach making more modest demands on agent behavior, in which agents incrementally adjust bids by adding or deleting a single point. This incremental approach comes at the expense of requiring approximately 70 more bidding cycles to reach equilibrium. In a spirit similar to the WALRAS algorithm, Schvartzman and Wellman (2007) apply an incremental model of bidding to a multi-unit double-auction market with indivisibilities, also finding high degrees of efficiency with the approach.

In the models of Chapter 4 and in the iterative multiattribute algorithm defined above, agents reason with respect to market clearing prices, i.e., the price adjustment process is not proportional to relative levels of excess demand. Hence, the convergence results derived for proportional price adjustment processes do not hold. Given market-clearing prices, agents in my model do not submit a price-dependent demand schedule for each market (i.e., each configuration), nor do these models feature an incremental approach to bidding. Rather, agents submit a single coherent demand set, given current prices, and submit own costs as their offer prices for all units. At each bidding iteration, agents calculate new preferred demand sets, disregarding their hypothetical transactions or individual effects on current prices (given prior bids).

The penalty for deviating from established tatonnement methods is that the guaranteed convergence found in prior work does not translate to the specific form of market-based algorithm that I employ. The lack of a convergence guarantee weakens the rationale for myopic best response bidding. Furthermore, as the following example demonstrates, the form of myopic best response implemented in the iterative multiattribute simulation is often suboptimal when agents have multi-unit demand for a given configuration.

Example 9 *Assume two goods, good B and good B, both offered at a price of 1. For simplicity, assume a single buyer, with the following valuation:*

$$\begin{aligned} v(A) &= 8 \\ v(B) &= 3 \\ v(A,A) &= 12 \\ v(B,B) &= 4 \\ v(A,B) &= 10 \end{aligned}$$

The optimal allocation has the buyer purchasing both units. Next, I outline the bidding and price trajectories in classical (discrete) tatonnement, the WALRAS algorithm, and my own formulation.

***Classical Tatonnement**: In classical tatonnement, the price of A is iteratively perturbed upward until it reaches 4, at which point the agent responds with $\{A,B\}$ (for surplus of $10-4-1=6>12-4-4=4$, and the market has reached equilibrium.*

$(p(A),p(B))$	$(d(A),d(B))$
$(1,1)$	$(2,0)$
$(2,1)$	$(2,0)$
$(3,1)$	$(2,0)$
$(4,1)$	$(1,1)$

***WALRAS**: In the WALRAS formulation, goods are equilibrated independently, with agents submitting a price-quantity schedule for a given good, taking the prices of other goods as given. I present these price-quantity schedules as (price,quantity) pairs. I use "—" to indicate that the WALRAS algorithm does not solicit a bid for the good during a particular iteration. In the following sequence, I assume the algorithm begins with the market for good B, and that prices begin at $(1,1)$. Note that at these initial prices, there is no price at which the agent has demand for good B.*

$(p(A),p(B))$	$d(A)$	$d(B))$
$(1,1)$	—	$(0,0)$
$(1,1)$	$(3,1)(7,1)$	—
$(4,1)$	—	$(3,1)$

***Multiattribute Call Market Implementation** In the multiattribute formulation, agents bid for only those items in their optimal demand sets as in classical tatonnement, but specify prices for all units in the demand set. Bids are again expressed as (price,quantity) pairs. The following bidding sequence, terminates in a cycle, never reaching equilibrium.*

$(p(A),p(B))$	$d(A)$	$d(B)$
$(1,1)$	$(4,1),(8,1)$	
$(1,9)$		$(1,1),(3,1)$
$(2,1)$	$(4,1),(8,1)$	
$(1,9)$		$(1,1),(3,1)$

Note that in bidding $(4,1),(8,1)$ in the first iteration, the bid values deviate from the first iteration of bidding on good *A* in the WALRAS algorithm. This is because in the WALRAS algorithm, the agent is computing marginal values with respect to the best alternative bundle, under the assumption that it will be able to secure the additional units in subsequent bidding rounds. In contrast, my bidding algorithm, agents are focused on winning the desired bundle, and bid the prices at which they are willing to buy that bundle.

One may characterize the deficiency of my implementation as a problem of bidder irrationality, in that agents are assuming an infinite supply at the quoted prices. In Example 9, the agent cycles endlessly between bidding on 2 units of A and then 2 units of B when facing market-clearing price quotes. This effect was not apparent in the simulations presented in Section 4.5, because the problem is most acute when agents are able to supply many copies of alternate configurations. Section 4.4.1 defined seller valuations which supported the production of at most 3 units of any given configuration, and fewer units in expectation.

In contrast, this effect is readily apparent in the TAC SCM simulation. I observed that agents typically computed optimal bids, i.e., optimal demand sets, which devoted all production to only a few (SKU,day) pairs, offering as many as 200 units for those pairs maximizing the difference between quoted price and cost. In the TAC SCM game, no single RFQ has quantity greater than 20, so it is inherently irrational for an agent to bid for and expect to transact quantity greater than 20 units at the quoted price. The following section imbues agents with a very simplistic heuristic to account for the non-linearity of good prices (equivalently, to account for the limited quantity offered at the quoted prices).

5.8.2 Bidding Optimization Quantity Limits

In seeking to understand and address the relative inefficiency of the iterative multiattribute mechanism in the TAC SCM simulation, I implemented a simplified model simulating a single day instance of the TAC SCM customer market auction. This model takes a single customer bid (from a prior TAC SCM simulation) for a problem instance, and generates agent state by randomly assigning component costs, capacities, and inventories in the following manner:

- **inventories** given a 10-day scope of target inventory levels, $I_o, I_o+\delta, I_o+2\delta, \ldots$ agents are assigned 10 days of random supplier deliveries, each distributed $\sim U[0,\delta]$
- **costs** Each agent is given a unique cost for each component as a percentage of the component's base price, where the percentage is taken from the discrete uniform distribution $[70,100]$.
- **manufacturing capacities** agents were each assigned 10 days of manufacturing capacity, each day distributed uniformly on $[0,200]$.

Agents then formulate bids as in the full TAC SCM simulation, bidding with respect to updated quote information at each iteration. I applied the same bidding optimization as defined in Section 5.6.4, but introduced an additional q_{max} parameter limiting the maximum quantity that agents apply to any single (RFQ,day) combination. The modified bidding optimization is then:

$$\begin{array}{ll} \max & \sum_d \sum_{SKU}(bq_{SKU,day} - cost_{SKU})) \cdot q_{SKU,day} \\ s.t. & \forall d \in days \quad \begin{cases} \sum_{SKU} \sum_{d=0}^{day} cycles(SKU) \cdot q_{SKU,d} \leq \sum_{d=0}^{day} cycles_d^{avail} \\ \forall comp \sum_{SKU|comp \in SKU} \sum_{d=0}^{day} q_{SKU,d} \leq \sum_{d=0}^{day} inventory_{comp,d} \\ \forall SKU, q_{SKU,day} < q_{max} \end{cases} \end{array}$$

I ran 50 simulations with 6 bidders for each q_{max} value depicted in the graph, averaging the efficiency as a function of bidding iteration (so iteration 6 is the point at which all agents have submitted a bid). I additionally ran 50 simulations with the sealed-bid multiattribute bidding strategy. The results depicted in Figure 5.11 show that limiting the quantity produced higher realized efficiency, with a maximum efficiency at $q_{max} = 8$. The efficiency of direct multiattribute bidding is depicted by the horizontal line at .779.

I next introduced best performing q_{max}-based strategy back into the full TAC SCM simulation. Figures 5.12 and 5.13 depict the CDF of efficiencies and mean efficiencies, respectively, for the optimum subproblem strategy, the original iterative multiattribute strategy and `GoBlueOval`. Sample sizes were 55 for iterative multiattribute (depicted as ma_i), 56 for the best performing multiattribute subproblem strategy (depicted as $\left(ma_i^{opt}\right)$) and 188 for randomized `GoBlueOval` (depicted as gbo_r)

As can be seen in Figures 5.12 and 5.13, a profile of multiattribute agents with a quantity-limited bidding optimization outperformed all other profiles with respect to market efficiency. This profile additionally yielded higher average efficiency than the that achieved in any round of the TAC SCM tournament in years 2003–2005.

While one may argue that searching for an optimally efficient strategy profile may bias the experiment in favor of the multiattribute market, restricting the search space to a single quantity parameter limits the extent of any such bias. Furthermore, it should be kept in mind that the multiattribute agents are fairly unsophisticated with respect to those of the 2005 TAC SCM finals. These agents do not implement customer demand forecasting or supplier capacity modeling, which are both behaviors which should increase aggregate market efficiency.

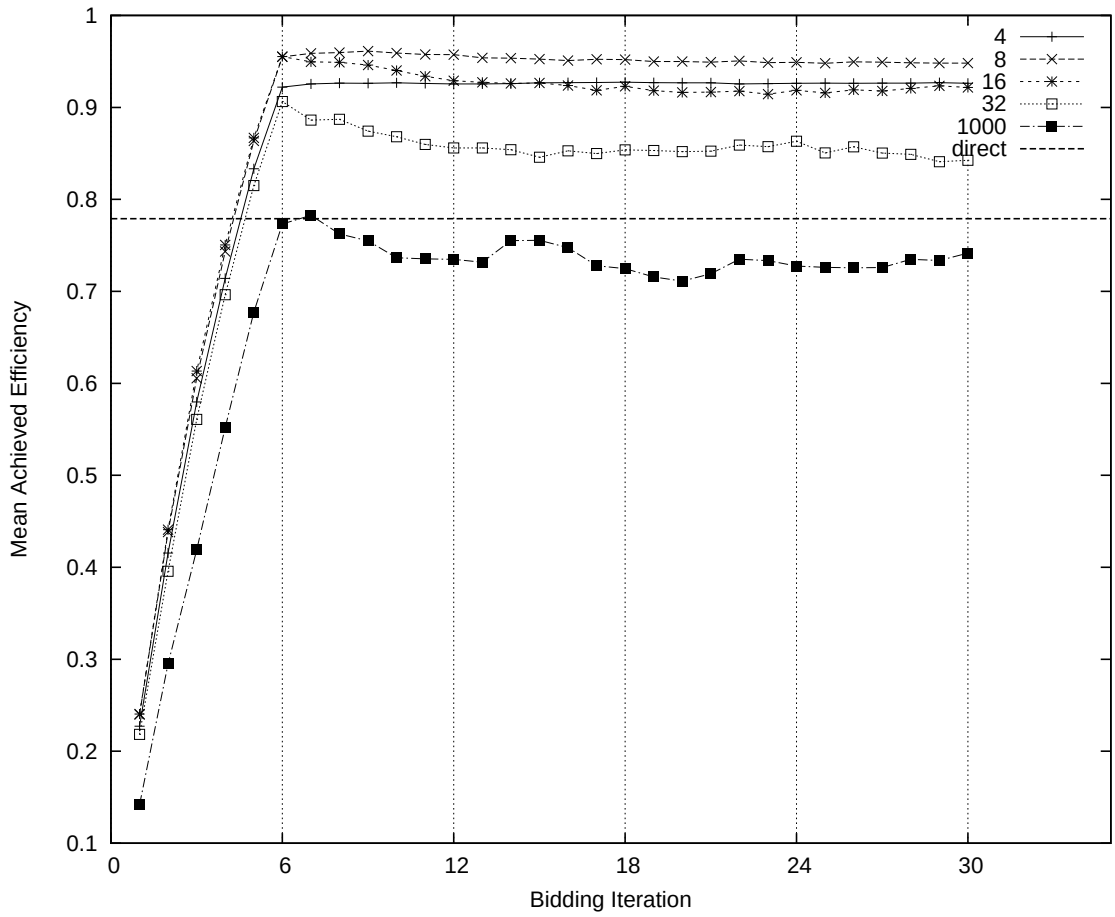

Figure 5.11: Mean efficiency as a function of bidding iteration for various maximum quantity restrictions on the iterative multiattribute agent bidding optimization.

5.9 Conclusion

In this chapter I evaluated my multiattribute call market design within the context of the Trading Agent Competition Supply Chain Management Game. As an existing simulation environment, TAC SCM provides a more objective evaluation domain, in that I do not have control over how problem instances are formulated. TAC SCM also provides a more realistic evaluation domain, in that agent valuations are derived from repeated market interactions, with other agents and with stochastically driven supply chain participants. To the extent that TAC SCM faithfully models a real supply chain, this domain also provides a benchmark for the potential benefit of replacing existing real-world supply chain negotiation procedures with multiattribute double auctions.

I described the TAC SCM game as operated in the 2005 TAC SCM competition, and presented an efficiency metric developed to measure the efficiency of a game instance. After showing an equivalence between customer RFQs and multiattribute offers, I outlined

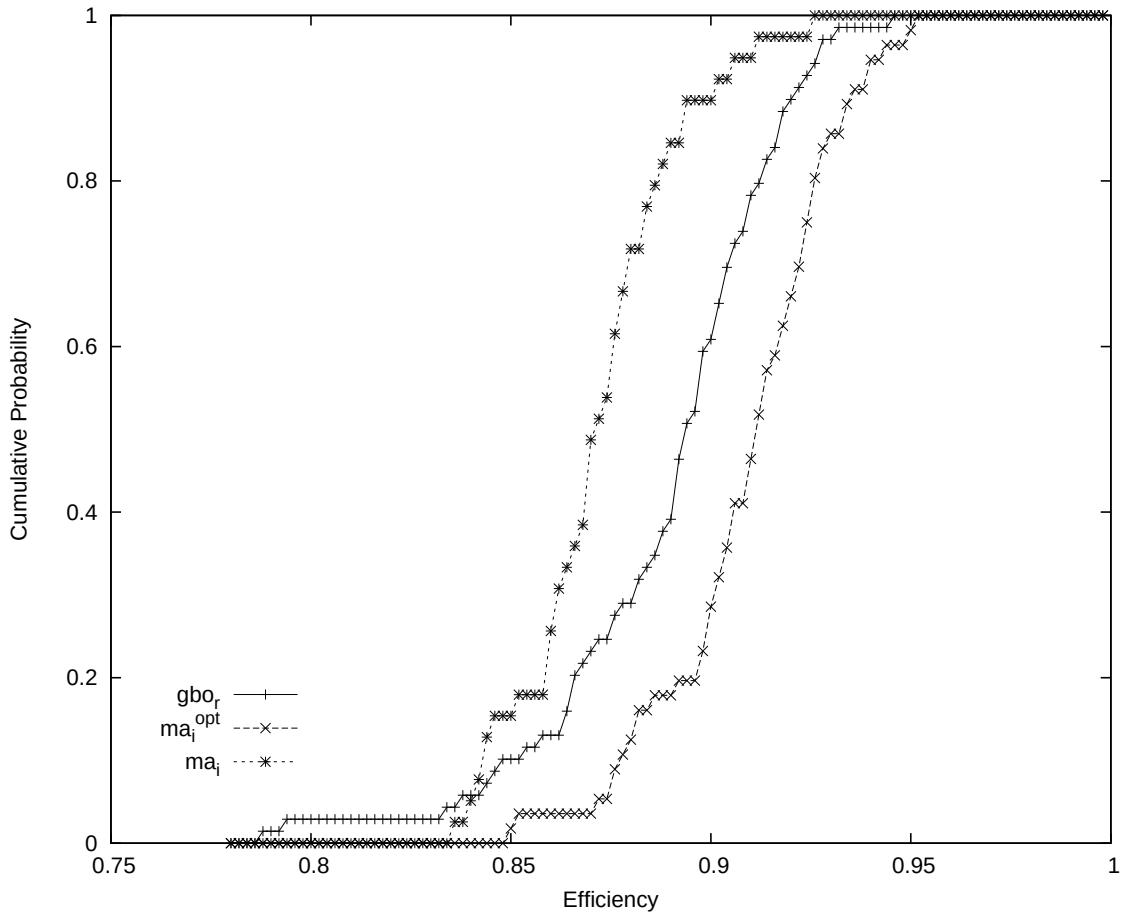

Figure 5.12: CDF of realized efficiencies for randomized `GoBlueOval` (gbo_r), iterative multiattribute (ma_i), and optimal subproblem strategy (ma_i^{opt}).

my procedure in integrating multiattribute markets into the TAC SCM simulation environment. This integration included modification of the existing simulation software, as well as definition of AB3D auction rules and bidding policies. In describing the integration of TAC SCM and AB3D, I provided a demonstration of the ease with which the AB3D system can be leveraged for market-based research.

I presented `GoBlueOval`, my entry in the 2005 TAC SCM competition, which served as a baseline agent strategy for simulations. I described the behavior of `GoBlueOval` in the existing TAC SCM environment, and presented alternative customer bidding algorithms for this environment. I showed that a randomized customer bidding strategy yielded higher efficiency than the strategic bidding algorithm employed in the 2005 competition, and subsequently used this randomized strategy as a baseline for comparison.

I presented sealed-bid and iterative multiattribute bidding strategies for the modified simulation environment, TACSCM-MA. After finding decreased market efficiency in

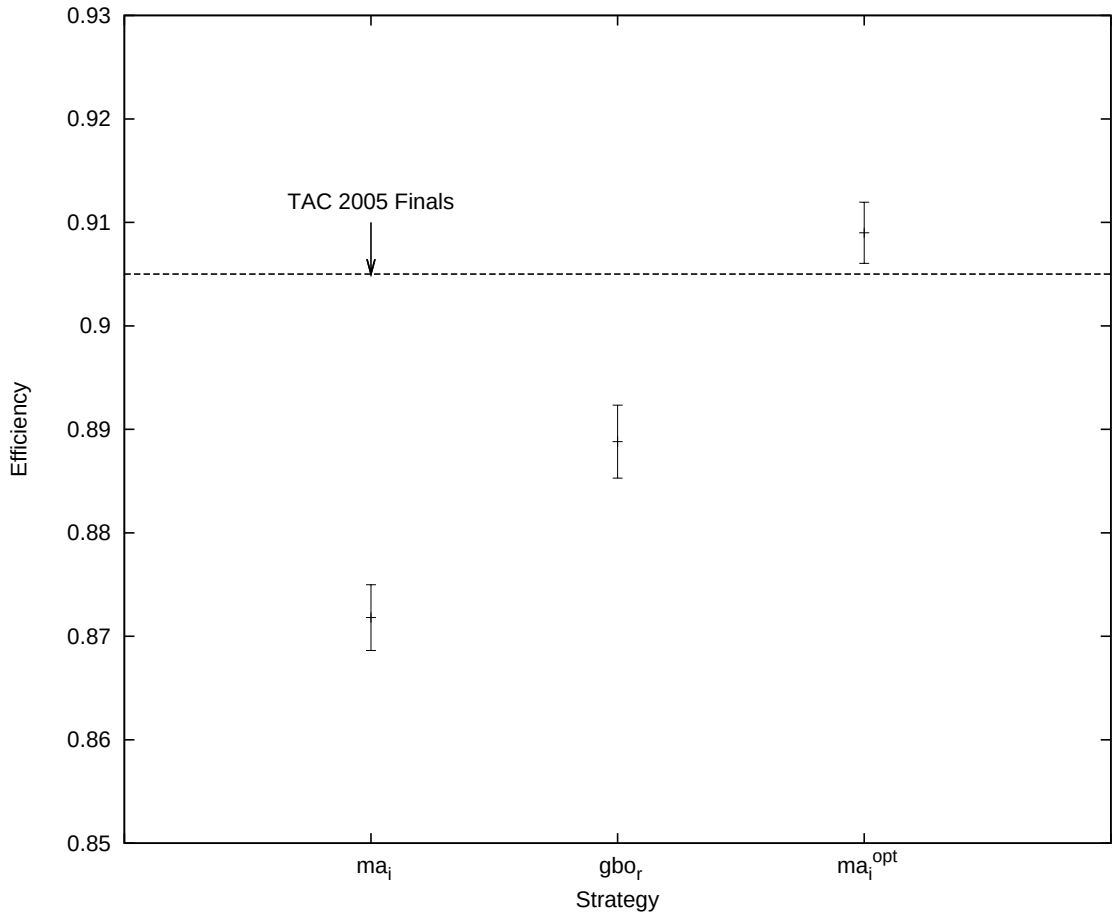

Figure 5.13: Mean realized efficiencies with 95% confidence intervals for randomized `GoBlueOval` (gbo_r), iterative multiattribute (ma_i), and optimal subproblem strategy $\left(ma_i^{opt}\right)$. Average efficiency for final round of 2005 TAC SCM competition is depicted by the dashed horizontal line.

TACSCM-MA with respect to the RFQ-based implementation, I described how the market-based algorithm implemented by my call market design violated technical convergence results.

I addressed the lack of convergence by exploring alternative bidding strategies on a stylized subproblem approximating the daily PC market in TAC SCM. Using a simple maximum quantity parameter within the manufacturer bidding optimization, I found alternative bidding strategies yielding significantly higher efficiency on the stylized subproblem. Using the best such bidding strategy, I presented new simulation results for TACSCM-MA, demonstrating higher levels of efficiency than that previously achieved by any profile of agents in the standard TAC SCM environment. In so doing, I presented evidence for the likely benefit of using multiattribute call markets in real-world domains.

Chapter 6
Summary of Contributions

In Chapter 1, I introduced and defined multiattribute goods, and suggested ways in which multiattribute double auctions offer potential efficiency benefits over standard negotiation practices for such goods. Specifically, in supporting centralized clearing exchanges for multiattribute goods, double auctions could translate the negotiation of multiattribute trade from a large set of disjoint markets into a small set of aggregated markets, mediating trade among a significantly greater set of traders. In doing so, they offer the potential for vast economic savings in many negotiation domains, by providing a constant source of trade liquidity.

I then stated my primary objective as facilitating the deployment of multiattribute double auctions by developing multiattribute call markets that are computationally tractable. This thesis meets that objective, in presenting and evaluating what is to my knowledge the first implemented multiattribute call market. In achieving my goal, I have made several contributions.

I presented an approach to information feedback that separates the algorithmic task into first finding configuration-free bilateral trade surpluses required to transact with any specific offer in the order book, and subsequently computing the quote for a specific configuration based on the set of all such surpluses. Given these bilateral surpluses, my approach allows specific configuration quotes to be generated only as needed, with time linear in the number of offers. My separation-based approach to information feedback thus partially addresses the computational challenge posed by the potentially huge space of configurations in a multiattribute auction.

My approach would still remain infeasible for all but the smallest problem sizes without an efficient method for calculating the required bilateral trade surpluses. I addressed this problem by devising a shortest path algorithm which can tackle the entire problem with complexity that is polynomial in the number of offers. I outlined the operation of this algorithm, presented examples demonstrating its use, and presented empirical results demonstrating its running time for various problem sizes.

I then combined my algorithm for information feedback with our polynomial-time clearing algorithm, and implemented these algorithms, along with supporting bidding languages, into a useable auction system. This work therefore presents an implemented multiattribute call market which supports clearing and information feedback with polynomial complexity.

Underlying my multiattribute call market implementation are two other contributions of this thesis, the AB3D market game platform, and the AB3D scripting language. The AB3D market game platform is capable of implementing a broad range of auctions, facilitating diverse market-game research agendas. The AB3D scripting language enables the concise specification of complex auction mechanisms. As these auction specifications are executable on the AB3D system, AB3DSL additionally enables the rapid deployment of complex auction mechanisms. I introduced my unique approach to auction specification, which employs a novel combination of static parameter settings and rule-based invocation of auction processes. Static parameter settings select among bidding language families and algorithmic implementations of auction processes that elude concise specification in a general-purpose language. Rule-based constructs support the specification of temporally complex mechanisms with a clear and simple syntax, allowing for qualitative modification of auction behavior through rule-based parameter modification. I discussed how my approach to auction specification addresses the relative weaknesses inherent in the strict use or strict avoidance of parameter-based specification. As evidence of the contribution of AB3D, I documented several research agendas which employ AB3D for market-based simulation, including my own evaluation of multiattribute call markets.

In assessing the efficiency of my market design in alternative problem domains, I presented a model of multi-unit multiattribute valuations derived from a component-based model of production. I demonstrated a natural way in which multiattribute valuations may elude direct expression given our required restrictions on multiattribute offers. I investigated the extent to which information feedback is able to compensate for the lack of expressive power of bids, exploring the extent to which the information feedback functionality of my design is able to support market-based algorithms. In so doing, I developed a metric on bidder valuations which correlates with the expected efficiency of my market design. To my knowledge, this valuation metric is the first attempt to characterize the relationship between bidder valuations and the efficiency of market-based algorithms for valuations which violate technical conditions for equilibrium.

Finally, I provided evidence for the applicability of multiattribute markets in real-world domains by integrating them into an existing supply chain simulation. I showed that the myopic bidding strategies assumed in provably convergent market-based algorithms are in-

compatible with the price-adjustment process inherent in my market design. I introduced a parametrized family of bidder strategies designed to reason more soundly with respect to market-clearing price quotes. Using a simplified model of the full supply chain simulation, I evaluated expected market efficiency for this family of strategies. I then integrated the strategy yielding maximal efficiency on this simplified model back into the full simulation, demonstrating an efficiency improvement over a more conventional negotiation procedure.

Bibliography

Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, Inc., 1993.

Kenneth J. Arrow, H. D. Block, and Leonid Hurwicz. On the stability of competitive equilibrium, II. *Econometrica*, 27:82–109, 1959.

Raghu Arunachalam and Norman M. Sadeh. The supply chain trading agent competition. *Electronic Commerce Research and Applications*, 4:63–81, 2005.

Raghu Arunachalam, Joakim Eriksson, Niclas Finne, Sverker Janson, and Norman Sadeh. The Supply Chain Management Game for the Trading Agent Competition 2004. Technical Report T2004-09, Swedish Institute of Computer Science, 2004.

Lawrence Ausubel and Paul Milgrom. Ascending auctions with package bidding. *Frontiers of Theoretical Economics*, 1(1):1–45, 2002.

Martin Bichler. *The Future of e-Markets: Multi-Dimensional Market Mechanisms*. Cambridge University Press, New York, NY, USA, 2001.

Martin Bichler and Jayant Kalagnanam. Configurable offers and winner determination in multi-attribute auctions. *European Journal of Operational Research*, 160:380–394, 2005.

Sushil Bikhchandani and Chi-Fu Huang. The economics of Treasury securities markets. *Journal of Economic Perspectives*, 7:117–134, 1993.

Justin Boyan and Amy Greenwald. Bid determination in simultaneous auctions: An agent architecture. In *Third ACM Conference on Electronic Commerce*, pages 210–212, 2001.

Fernando Branco. The design of multidimensional auctions. *RAND Journal of Economics*, 28:63–81, 1997.

Yeon-Koo Che. Design competition through multidimensional auctions. *RAND Journal of Economics*, 24:668–680, 1997.

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

Shih-Fen Cheng. Designing the market game for a commodity trading simulation. In *IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, pages 445–449, 2007.

Shih-Fen Cheng, Michael P. Wellman, and Dennis G. Perry. Market-based resource allocation for information-collection scenarios. In Koichi Kurumatani, Shu-Heng Chen, and Azuma Ohuchi, editors, *Multiagent for Mass User Support (MAMUS-03)*, pages 33–47. Springer-Verlag, 2004.

E. H. Clarke. Multipart pricing of public goods. *Public Choice*, 11:17–33, 1971.

Dave Cliff. Explorations in evolutionary design of online auction market mechanisms. *Electronic Commerce Research and Applications*, 2:162–175, 2003.

John Collins, Wolfgang Ketter, and Maria Gini. A multi-agent negotiation testbed for contracting tasks with temporal and precedence constraints. *International Journal of Electronic Commerce*, 7(1):35–57, 2002.

Vincent Conitzer and Tuomas Sandholm. Revenue failures and collusion in combinatorial auctions and exchanges with VCG payments. In *Fifth ACM conference on Electronic commerce*, pages 266–267, 2004a.

Vincent Conitzer and Tuomas Sandholm. Computational criticisms of the revelation principle. In *Fifth ACM Conference on Electronic Commerce*, pages 262–263, 2004b.

Peter Cramton. Ascending auctions. *European Economic Review*, 42(3-5):745–756, 1998.

Peter Cramton, Yoav Shoham, and Richard Steinberg, editors. *Combinatorial Auctions*. MIT Press, 2005.

Shahar Dobzinski, Noam Nisan, and Michael Schapira. Approximation algorithms for combinatorial auctions with complement-free bidders. In *Thirty-Seventh Annual ACM Symposium on Theory of Computing*, pages 610–618, 2005.

Nicholas Economides and Robert A. Schwartz. Electronic call market trading. *Journal of Portfolio Management*, 21(3):10–18, 1995.

Yagil Engel and Michael P. Wellman. Generalized value decomposition and structured multiattribute auctions. In *Eighth ACM Conference on Electronic Commerce*, pages 227–236, 2007.

Yagil Engel, Michael P. Wellman, and Kevin M Lochner. Bid expressiveness and clearing algorithms in multiattribute double auctions. In *Seventh ACM Conference on Electronic Commerce*, pages 110–119, 2006.

Maria Fasli and Michael Michalakopoulos. e-Game: A platform for developing auction-based market simulations. *Decision Support Systems*, 44(2):469–481, 2008.

FCC. Federal Communications Commission notice and filing requirements, minimum opening bids, reserve prices, upfront payments, and other procedures for auctions 73 and 76. `http://hraunfoss.fcc.gov/edocs_public/attachmatch/DA-07-4171A1.pdf`, 2007.

Eugene Fink, Josh Johnson, and Jenny Hu. Exchange market for complex goods: Theory and experiments. *Netnomics*, 6(1):21–42, 2004.

Robert Fourer, David M. Gay, and Brian W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. Duxbury Press, 2002.

Daniel Friedman and John Rust, editors. *The Double Auction Market*. Addison-Wesley, 1993.

Henner Gimpel, Juho Makio, and Christof Weinhardt. Multi-attribute double auctions in financial trading. In *Seventh IEEE International Conference on E-Commerce Technology*, pages 366–369, 2005.

Jianli Gong. *Exchanges for Complex Commodities: Search for Optimal Matches*. PhD thesis, University of South Florida, 2002.

Amy Greenwald and Justin Boyan. Bidding algorithms for simultaneous auctions. In *Third ACM Conference on Electronic Commerce*, pages 115–124, 2001.

Theodore Groves. Incentives in teams. *Econometrica*, pages 617–631, 1973.

Faruk Gul and Ennio Stacchetti. The English auction with differentiated commodities. *Journal of Economic Theory*, 92(1):66–95, May 2000.

Faruk Gul and Ennio Stacchetti. Walrasian equilibrium with gross substitutes. *Journal of Economic Theory*, 87(1):95–124, July 1999.

Patrick R. Jordan, Christopher Kiekintveld, Jason Miller, and Michael P. Wellman. Market efficiency, sales competition, and the bullwhip effect in the TAC SCM tournaments. In *AAMAS-06 Workshop on Trading Agent Design and Analysis*, pages 62–74, 2006.

Patrick R. Jordan, Christopher Kiekintveld, and Michael P. Wellman. Empirical game-theoretic analysis of the TAC supply chain game. In *Sixth International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 1188–1195, 2007.

Jayant R. Kalagnanam, Andrew J. Davenport, and Ho S. Lee. Computational aspects of clearing continuous call double auctions with assignment constraints and indivisible demand. *Electronic Commerce Research*, 1(3):221–238, 2001.

Lutz Kaufmann and Craig R. Carter. Deciding on the mode of negotiation: To auction or not to auction electronically. *Journal of Supply Chain Management*, 40(2), 2004.

A. S. Kelso and V. P. Crawford. Job matching, coalition formation, and gross substitutes. *Econometrica*, 50:1483–1504, 1982.

Christopher Kiekintveld, Jason Miller, Patrick R. Jordan, and Michael P. Wellman. Forecasting market prices in a supply chain game. In *Sixth International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 1–8, 2007.

Kate Larson and Tuomas Sandholm. Costly valuation computation in auctions. In *Eighth Conference on Theoretical Aspects of Rationality and Knowledge*, pages 169–182, 2001.

Benny Lehmann, Daniel Lehmann, and Noam Nisan. Combinatorial auctions with decreasing marginal utilities. *Games and Economic Behavior*, 55(2):270–296, 2006.

A. R. Lomuscio, M. Wooldridge, and N. R. Jennings. A classification scheme for negotiation in electronic commerce. In F. Dignum and C. Sierra, editors, *Agent Mediated Electronic Commerce: A European Perspective*, pages 19–33. Springer-Verlag, 2000.

Jeffrey K. MacKie-Mason and Michael P. Wellman. Automated markets and trading agents. In Leigh Tesfatsion and Kenneth L. Judd, editors, *Handbook of Agent-Based Computational Economics*. Elsevier, 2006.

Andreu Mas-Colell, Michael D. Whinston, and Jerry R. Green. *Microeconomic Theory*. Oxford University Press, New York, 1995.

Paul Milgrom and Bruno Strulovici. Concepts and properties of substitute goods. Technical report, Economics Group, Nuffield College, University of Oxford, May 2006.

Roger B Myerson. Incentive compatibility and the bargaining problem. *Econometrica*, 47 (1):61–73, 1979.

Roger B. Myerson. Optimal auction design. *Mathematics of Operations Research*, 6(1): 58–73, 1981.

D. Neumann, C. Holtmann, H. Weltzien, C. Lattemann, and Ch. Weinhardt. Towards a generic e-market design. In J. Monteiro, P. M. C. Swatman, and L. V. Tavares, editors, *Towards the Knowledge Society: e-Commerce, e-Business and e-Government*, pages 289–305. Kluwer Academic Publishers, 2002.

David C. Parkes and Jayant Kalagnanam. Models for iterative multiattribute Vickrey auctions. *Management Science*, 51:435–451, 2005.

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

Steve Phelps, Simon Parsons, Peter McBurney, and Elizabeth Sklar. Co-evolution of auction mechanisms and trading strategies: Towards a novel approach to microeconomic design. In *GECCO-02 Workshop on Evolutionary Computation in Multi-Agent Systems*, pages 65–72, 2002.

David Porter, Stephen Rassenti, Anil Roopnarine, and Vernon Smith. Combinatorial auction design. *Proceedings of the National Academy of Sciences of the United States of America*, 100(19):11153–11157, 2003.

Daniel M. Reeves. *Generating Trading Agent Strategies*. PhD thesis, University of Michigan, 2005.

Daniel M. Reeves, Michael P. Wellman, and Benjamin N. Grosof. Automated negotiation from declarative contract descriptions. *Computational Intelligence*, 18:482–500, 2002.

Daniel M. Reeves, Michael P. Wellman, Jeffrey K. MacKie-Mason, and Anna Osepayshvili. Exploring bidding strategies for market-based scheduling. *Decision Support Systems*, 39:67–85, 2005.

John G. Riley and William F. Samuelson. Optimal auctions. *American Economic Review*, 71(3):381–392, June 1981.

Juan A. Rodriguez-Aguilar, Francisco J. Martin, Pere Garcia, and Carles Sierra. Competitive scenarios for heterogeneous trading agents. In *Second International Conference on Autonomous Agents*, pages 293–300, 1998.

Daniel Rolli, Stefan Luckner, Henner Gimpel, and Christof Weinhardt. A descriptive auction language. *Electronic Markets*, 16(1):51–62, 2006.

Michael H. Rothkopf. Thirteen reasons why the Vickrey-Clarke-Groves process is not practical. *Operations Research*, 45(2):191–197, 2007.

J. Rust, J. H. Miller, and R. Palmer. Characterizing effective trading strategies. *Journal of Economic Dynamics and Control*, 18:61–96, 1994.

John Rust and John H. Miller. Behavior of trading automata in a computerized double auction market. In Friedman and Rust (1993), chapter 6, pages 155–198.

Paul A. Samuelson. *Foundations of Economic Analysis*. Harvard University Press, 1947.

Tuomas Sandholm. The winner determination problem. In Cramton et al. (2005).

Tuomas Sandholm. eMediator: A next generation electronic commerce server. *Computational Intelligence*, 18(4):656–676, 2004.

Tuomas Sandholm, Subhash Suri, Andrew Gilpin, and David Levine. Winner determination in combinatorial auction generalizations. In *First International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 69–76, 2002.

Mark A. Satterthwaite and Steven R. Williams. Bilateral trade with the sealed bid *k*-double auction: Existence and efficiency. *Journal of Economic Theory*, 48:107–133, 1989.

L. Julian Schvartzman and Michael P. Wellman. Market-based allocation with indivisible bids. *Production and Operations Management*, 16:495–509, 2007.

Ilya Segal. The communication requirements of combinatorial allocation problems. In Cramton et al. (2005).

Jason Shachat and J. Todd Swarthout. Procurement auctions for differentiated goods. Technical Report 0310004, Economics Working Paper Archive at WUSTL, October 2003.

Aditya V. Sunderam and David C. Parkes. Preference elicitation in proxied multiattribute auctions. In *Fourth ACM Conference on Electronic Commerce*, pages 214–215, 2003.

William Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *Journal of Finance*, 16(1):8–37, 1961.

Yevgeniy Vorobeychik, Christopher Kiekintveld, and Michael P. Wellman. Empirical mechanism design: Methods, with application to a supply-chain scenario. In *Seventh ACM Conference on Electronic Commerce*, pages 306–315, 2006.

William E. Walsh, Rajarshi Das, Gerald Tesauro, and Jeffrey O. Kephart. Analyzing complex strategic interactions in multi-agent games. In *AAAI-02 Workshop on Game-Theoretic and Decision-Theoretic Agents*, 2002.

Michael P. Wellman, Shih-Fen Cheng, Daniel M. Reeves, and Kevin M. Lochner. Trading agents competing: Performance, progress, and market effectiveness. *IEEE Intelligent Systems*, 18(6):48–53, 2003.

Michael P. Wellman, Daniel M. Reeves, Kevin M. Lochner, Shih-Fen Cheng, and Rahul Suri. Approximate strategic reasoning through hierarchical reduction of large symmetric games. In *Twentieth National Conference on Artificial Intelligence*, pages 502–508, 2005.

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

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

Peter R. Wurman, Michael P. Wellman, and William E. Walsh. A parametrization of the auction design space. *Games and Economic Behavior*, 35:304–338, 2001.

Makoto Yokoo, Yuko Sakurai, and Shigeo Matsubara. The effect of false-name bids in combinatorial auctions: New fraud in Internet auctions. *Games and Economic Behavior*, 46(1):174–188, 2004.