Toward the Development of an Intelligent Agent for the Supply Chain Management Game of the 2003 Trading Agent Competition

Colin McMillen

May 9, 2003

Abstract

This thesis considers the problems posed by the supply chain management game that is a new addition to the 2003 Trading Agent Competition (TAC). An instance of this game involves six software agents attempting to maximize profits by manufacturing computers in a simulated market economy. The design of such an agent is an interesting problem because a good solution is thought to require a synthesis of several fields, including industrial operations research, economics, game theory, artificial intelligence, machine learning, multiagent systems, computational complexity theory, statistics, and probability.

The primary contribution of this thesis is its extensive analysis of the TAC supply chain management (SCM) game. We present a survey of work from a wide variety of fields that are related to the TAC SCM competition. We present a detailed analysis of many aspects of the TAC SCM game, including proofs of theoretical results, metrics by which the performance of agents can be measured, and suggestions of strategies that may be of use in the implementation of sophisticated TAC SCM agents. We describe our initial implementation of a simple TAC SCM agent, and present preliminary experimental results that seem to indicate that this implementation is at least as sophisticated as those of other teams. These experiments also illustrate areas where our agent's performance could be improved. We conclude with a discussion of future research directions.

Acknowledgements

I would like to thank Professor Maria Gini. Without her support, guidance, and encouragement, this thesis would not have been possible. I would also like to thank Professor Nikolaos Papanikolopoulos for initially challenging me to pursue scientific research, and the rest of my professors at the University of Minnesota for creating an educational, enjoyable, and challenging atmosphere.

Meetings with members of the MAGNET research project led to the development of many of the ideas presented in this thesis. I would especially like to thank John Collins and Amrudin Agovic, who implemented a substantial portion of our current agent. Without their hard work, the experimental results presented in this thesis would not have been possible. John was also instrumental in the design of the software architecture of our agent.

I would also like to thank Kristen Stubbs, Michael Zaimont, and Paul Rybski. Their friendship over the last few years has been invaluable, both socially and intellectually. I especially appreciate their willingness to listen to problems and to help me develop, refine, and clarify solutions to these problems. Kristen also spent a great deal of her time in helping me revise this thesis. Without her help, many of my ideas would not be presented as clearly. Paul's cheerful disposition and sense of humor have showed me that research can, and should be, enjoyable. His influence is partially responsible for my decision to pursue a doctoral degree.

I am especially grateful for the support of my parents, Shaun and Roseann McMillen, and my sister, Katie. I cannot express how important their dedication and love have been throughout my life.

This work was supported financially through a grant from the National Science Foundation.

Contents

1 **Introduction** **11**

2 **Problem Description** **15**

2.1 Overview 15

2.2 Customers 16

2.3 Suppliers 17

2.4 Assembly and Fulfillment 18

2.5 Market Reports 18

2.6 End of Game and Winner Determination 19

3 **Related Work** **21**

3.1 SCM Techniques Used in Industry 21

3.1.1 The Kanban Heuristic 23

3.1.2 The CONWIP Heuristic 23

3.2 Software Tools for SCM 24

3.2.1 Decision-Support Software 24

3.2.2 Automated Supply Chain Management 25

3.3 Economics and Game Theory 26

3.3.1 Individual Rationality 26

3.3.2 Pareto Efficiency 27

3.3.3 Game Theory 28

3.4 Artificial Intelligence Techniques 29
3.4.1 Game Playing 29
3.4.2 Machine Learning 30
3.4.3 Planning 32
3.4.4 Reasoning under Uncertainty 33
3.5 Multiagent Systems 35
3.5.1 Decision-Making: Voting and Auctions 35
3.5.2 Learning to Play Better Against an Opponent 37
3.5.3 Learning to Benefit from Market Conditions 38
3.6 TAC Classic 39
3.7 Software for the TAC SCM Game 42

4 Analysis **45**

4.1 Interest Rate 45
4.2 Bottlenecks 46
4.2.1 Use of Bottlenecks in *Post Hoc* Game Analysis 49
4.2.2 Bottleneck Analysis of a Typical Game 49
4.2.3 On-Line Bottleneck Prediction 50
4.3 Maximum Score of a Single Agent 51
4.4 Maximum Total Score for a Six-Agent Game 52
4.5 The Cost Efficiency Metric 55
4.6 Agent Design 57
4.7 Procurement 58
4.7.1 Demand Fulfillment 58
4.7.2 Demand Forecasting 59
4.7.3 Supply Availability and Price Estimation 60
4.7.4 Strategic Procurement Behavior 61

4.7.5 Minimization of Inventory Costs 62
4.8 Sales 62
4.8.1 Order Securement 63
4.8.2 RFQ Trend Prediction 64
4.8.3 Customer Order Probabilities 65
4.8.4 Late Deliveries 67
4.8.5 Strategic Sales Behavior 69
4.8.6 Minimization of Inventory Costs 69
4.9 Production 70
4.10 Delivery 71
4.11 Miscellaneous Performance Metrics 71
4.11.1 Product Quality 72
4.11.2 Supply Chain Lead Time 72

5 Implementation 73
5.1 Procurement 73
5.2 Production 75
5.3 Sales 75
5.4 Delivery 76

6 Experimental Results 77

7 Conclusions and Future Work 83

A Source Code for the Supplier Capacity Simulation 85

Bibliography 89

Chapter 1

Introduction

Recently, competitive scenarios have been increasingly utilized as testbeds for the development of multiagent systems. Perhaps the most well-known multiagent competition is the Robot World Cup Initiative (RoboCup), a robotic soccer competition that specifically emphasizes autonomous multiagent collaboration [24]. Another competition sponsored by the RoboCup Federation is RoboCup Rescue [25], in which teams of autonomous or teleoperated robots attempt to find victims in an urban search and rescue course. Researchers interested in autonomous bidding agents started the Trading Agent Competition (TAC) in 2000 [53]. The original TAC game involves agents competing to satisfy clients' travel and entertainment desires in a simulation. Agents are responsible for buying airline tickets, reserving hotels, and finding entertainment packages suitable to each client's desires. The Trading Agent Competition has been an annual event since 2000. A new game for the 2003 Trading Agent Competition (TAC-03) has been proposed by researchers at the Carnegie Mellon University e-Supply Chain Management Laboratory and the Swedish Institute of Computer Science (SICS) [2]. This game involves a supply chain management scenario in which agents attempt to maximize profits by manufacturing personal computers (PCs) to sell to customers. This new supply-chain management game is called TAC SCM; the original game is now referred to as TAC Classic.

Stone [52] discusses multiagent competitions, including RoboCup and TAC. He gives an overview of the rules of these competitions, and notes some similarities and differences between the two. Most importantly, Stone shares his views on the benefits and drawbacks of such competitions. These views are based on his participation in numerous instances of the RoboCup and TAC competitions. He claims that there are many ways in which scientific progress can be hindered by an organized competition, including: obsession with winning, domain-specific solutions, barriers to entry, restrictive rules, and invalid evaluation conclusions. On the other hand, there are also many ways in which scientific progress can be advanced by an organized competition, by providing research inspiration, providing deadlines for creating complete working systems, providing a common platform for testing and exchanging ideas, encouraging continual improvement of solutions, and encouraging flexible software and hardware. Stone concludes that the benefits of RoboCup and TAC outweigh the hazards, but emphasizes that the competition results alone are not scientifically conclusive. He recommends that implementations of novel approaches to the problems posed by RoboCup and TAC be tested not only in competition, but also in controlled, empirical experiments.

There are a number of reasons why we believe the TAC SCM competition is interesting. Agents participating in a TAC game must base their decisions on limited information about the state of the market and the strategies of other agents. Agents must simultaneously compete in two separate but interrelated markets: the market from which the agents must buy their supplies and the market to which the agents must sell their finished products. Agents have a large number of decisions to make in a limited time, so the computational efficiency of the decision-making process is paramount. We believe that effective solutions to the TAC SCM game will be multidisciplinary, as there is substantial related work in the fields of industrial operations research, economics, game theory, artificial intelligence, multiagent systems, computational complexity theory, statistics, and probability.

The main contributions of this thesis include a broad summary of work in related areas and theoretical analysis specific to the TAC SCM game. This analysis suggests metrics by which the performance of TAC SCM agents can be measured and suggestions of strategies that could be used by an ideal agent. We briefly discuss our implementation of a very simple TAC SCM agent. We present experimental results that compare the performance of this agent to simple agents developed by other groups and to the theoretical limits suggested by our analysis. We conclude by discussing directions in which our future work is likely to progress.

Chapter 2

Problem Description

The rules for the 2003 supply chain management game can be found in full at [2]. They are briefly summarized here for the sake of completeness.

2.1 Overview

In each TAC SCM game, six autonomous software agents compete to maximize profits in a computer-assembly scenario. The simulation takes place over a large number of virtual days, each lasting about fifteen seconds of real time. Each agent has a bank account with an initial balance of zero. The winning agent is the one with the highest bank balance when the game is completed. Agents are allowed to go infinitely far into debt, but an interest rate applies to both positive and negative bank balances. The interest rate is chosen randomly at the beginning of the game, and is expressed as an annual rate, though interest is compounded daily.

A Component Catalog and Bill of Materials are sent to each agent at the beginning of the game. The Component Catalog lists each possible computer component, along with the component's base price and a list of suppliers who can produce that component. These components are the raw materials of the TAC SCM supply chain. The Bill of Materials lists sixteen different permutations of components that can be assembled into completed PCs. Each of these computer types is identified uniquely by a stock keeping unit (SKU) number. Each SKU is also assigned a number of processing cycles that determines how much time it takes to assemble that type of computer from raw materials. These PCs are the finished goods of the TAC SCM supply chain. The values specified in the Component Catalog and Bill of Materials do not change over the course of the game.

2.2 Customers

Each agent receives requests for quotes (RFQs) from potential customers each day. Each RFQ specifies which type of computer the customer desires, along with a quantity, a due date, a reserve price, and a penalty.

Each agent may choose to bid on some or all of the day's RFQs. Individual bids are hidden from other agents, but a summary of each day's price ranges for each type of computer is provided to the agents at the start of the next day. At the end of the day, each customer will order from the agent with the lowest bid. Customers will not consider bids above the reserve price.

The penalty is expressed as a percentage of the reserve price, and is immediately deducted from the winning agent's bank account for each day that the agent is late in delivering the finished product. If the product has not been delivered within five days of the due date, the order is cancelled by the customer and the agent receives no payment.

Each agent also receives a list of customer orders that were won in the previous day's customer RFQ bidding. The agent is responsible for fulfilling these orders by their due dates.

2.3 Suppliers

To attain raw materials, an agent must send a request for quotes (RFQ) to appropriate suppliers. Each computer component is produced by one or two suppliers; each supplier provides two different types of components. These correspondences are detailed in each game's Component Catalog. Each RFQ specifies a component type, a quantity, and a due date.

On the next day, the agent will receive a response to each RFQ. If the supplier expects that it can produce the desired quantity on time, it responds with an offer that contains the price of the supplies. If the supplier predicts that it will have a large amount of available supply by that time, this price will be discounted; discounted prices may be as low as 50% of the component's base price. The agent can then accept this offer by placing an order.

If the supplier does not expect that it can produce the desired quantity on time, it responds with two offers:

- An earliest complete offer, in which the supplier responds with a revised due date and a price. This revised due date is the first day in which the supplier believes it will be able to supply the entire desired quantity.
- A partial offer, in which the supplier responds with a revised quantity and a price. The revised quantity is the amount of raw materials which the supplier believes it will have available on the desired due date.

The agent can accept either one of these amended offers, or reject both.

An agent receives a notification whenever a supplier has delivered parts. Suppliers will not deliver fractional parts of orders, nor will they deliver early, even if the desired supplies are available earlier than the supplier expected. Suppliers may deliver late, due to randomness in their production capacities. Late orders are given priority over other orders. When supplies are received, the agent's bank account is decreased by the order price. The agent's inventory of raw materials is unlimited, and there is no penalty for holding a large inventory.

2.4 Assembly and Fulfillment

Once an agent has procured the necessary raw materials, it may begin assembling finished products. The raw materials required for each type of PC are given in the Bill of Materials. The Bill of Materials also lists the number of assembly cycles required to complete a machine. Each agent has a fixed number of these cycles available per day, and must therefore create a daily production schedule. Finished goods are also held in inventory. This inventory can also be infinitely large, and there is no penalty for holding a large inventory.

An agent fulfills a customer order when it schedules finished goods to be shipped out of inventory and to the customer. Each agent has an infinite shipping capacity. Shipped items are received by the customer on the next day. There is no bonus for shipping finished goods early; the penalty for shipping late is described above. The agent's bank account is increased by the order price when the customer receives the goods.

2.5 Market Reports

Each agent also receives periodic reports of the market state. These reports are issued every twenty days. These reports include the following information:

- Aggregate quantities of each component produced by all suppliers in the last 20 days
- Aggregate quantities of each component sold by all suppliers in the last 20 days
- Average price of each type of PC sold in the last 20 days
- Quantity sold of each PC type in the last 20 days

These reports are made available so that the agents can use the data to plan strategies. In addition, people watching the game can view their agent's bank balance, inventory, and deliveries as the game progresses.

2.6 End of Game and Winner Determination

After approximately 200-250 virtual days (which corresponds to roughly an hour of real time), the game ends. The agent with the highest bank account balance is declared the winner.

Chapter 3

Related Work

Since this year's Trading Agent Competition will include the inaugural version of the supply chain management game, there is very little work that has been done by others on this specific game. However, there are substantial bodies of literature in a wide variety of related fields, including traditional supply chain management techniques, supply chain decision-support and automated management software, economics, game theory, artificial intelligence, and multiagent systems. Results from the TAC Classic competition may also apply, to some degree, to the TAC SCM competition.

3.1 SCM Techniques Used in Industry

Researchers in the operations research community have contributed a substantial amount of analysis of the supply chain management strategies actually utilized by businesses and industry.

Lee and Billington [27] discuss commonly-made pitfalls in supply chain management practices. Tayur et al. [59] review quantitative approaches to supply chain modeling. Their discussion primarily focuses on mathematical and computational analysis of supply chain models. They discuss the design and evaluation of supply contracts, the impact of information on decision-making performance, management of product variety, and future research challenges.

Biswas and Narahari [5] classify supply chain problems into three categories based on the expected timeframe of the decisions: strategic, tactical, and operational. Strategic decisions affect the long-term performance of the supply chain, tactical decisions affect medium-term performance, and operational decisions affect short-term performance. Operational decisions are under the additional constraint that they must be made in real-time. They further classify supply chain decisions into four functional categories: procurement, manufacturing, distribution, and logistics. They also give a list of measures that can be used to quantify the performance of a supply chain. These measures are useful as metrics for real-world supply chain situations, but are also useful as measures of the performance of TAC SCM agents. The measures suggested by Biswas and Narahari include:

- **Customer satisfaction.** Biswas and Narahari classify customer satisfaction as a qualitative measure; they note that, in general, customer satisfaction can be a hard metric to quantify.

- **Product quality.** Biswas and Narahari also classify product quality as a qualitative measure.

- **Supply chain lead time.** In general, this is the time required for the supply chain to acquire raw materials, plus the amount of time required to convert raw materials into finished goods, plus the amount of time required to ship finished goods to the customer.

- **Order-to-delivery lead time.** This is the amount of time elapsed between the placement of a customer's order and the customer's receipt of finished goods.

- **Order fill rate.** This is the fraction of customer orders that are immediately met from existing stock.

- **Probability of on-time delivery.** This is the fraction of customer orders that are fulfilled by the due date.

- **Financial costs.** Biswas and Narahari list some sources of supply chain costs, including inventories, transportation, facilities, operations, technology, materials, and labor.

3.1.1 The Kanban Heuristic

One well-accepted supply chain optimization heuristic that is used in industry is the Kanban heuristic. This heuristic was developed at the Toyota Motor Company in the early 1980s [49]. "Kanban" is the Japanese word for "card;" the Kanban heuristic utilizes the circulation of (real or simulated) cards to control rates of production. Each machine in the supply line starts with some number of cards; a machine is not allowed to process goods unless it possesses at least one card. When it is done processing, the machine passes one card to the machine immediately upstream. The last machine in the supply line receives a card whenever a unit of finished product is consumed by customer demand. This machine then creates a new unit of finished product, using the results of the upstream machine as raw materials. It then passes this new card to the upstream machine. This card will flow back upstream, eventually informing all the machines that they should produce one additional part. The Kanban heuristic is primarily used as an inventory-control tactic; the size of the inventories of finished and partially-completed goods is bounded by the number of cards in the supply chain at any time.

3.1.2 The CONWIP Heuristic

Another supply-chain optimization heuristic is the constant work-in-progress (CONWIP) strategy [50]. Like the Kanban heuristic, CONWIP attempts to limit the amount of in-progress work of the supply chain. Kanban achieves this by capping work-in-progress at each workstation in the supply chain; CONWIP caps work-in-progress of the supply chain as a whole. This can lead to a more flexible assignment of work, especially when there is variability in the supply chain. For instance, if a machine malfunctions in the Kanban system, all upstream machines will eventually be prevented from doing any work. Under CONWIP, these machines might be able to continue processing for a longer period of time.

3.2 Software Tools for SCM

A variety of software tools have been developed to aid in the supply chain management process. Most of these tools fall under the category of decision-support software: they simulate supply chains and allow users to see the effects of changing various parameters. Some researchers in academia and industry have also begun to develop tools to automate increasingly complex supply chain decisions.

3.2.1 Decision-Support Software

Biswas and Narahari [5] present an object-oriented modeling system called DESSCOM that is intended to provide decision support for supply chain problems. They present a case study of a real-world liquid petroleum gas supply chain and show how their system facilitates decision-making in many areas of the supply chain.

Jain et al. [23] present a system that simulates supply chains for semiconductor manufacturing. They analyze the supply chain at different levels of detail, and show experimentally that a well-detailed model of the supply chain provides more accurate decision support than an abstracted model.

Ettl and Schwehm [14] present an analytical method for modeling Kanban systems. They use this method along with a general-purpose genetic algorithm to determine allocation of kanbans and supply chain network configuration. The goal of their algorithm is to minimize inventory costs while maintaining a desired throughput rate. They present results from a set of experiments that compare the results of the genetic algorithm to the results of an exhaustive search that finds the globally optimal configuration. The relative difference in inventory levels between the optimal solution and their approximate solution is consistently under 5%. Since the underlying problem has been shown to be NP-hard, they claim that their heuristic approach is justified and efficient.

There are also a large number of commercial software tools for supply chain simulation, modeling, and analysis. These include the IBM Supply Chain Simulator [7], the i2 Technologies RHYTHM decision-support system [22], and the SAP Advanced Planner and Optimizer [46].

3.2.2 Automated Supply Chain Management

SAP AG offers a product called mySAP SCM [45] that combines supply chain modeling, design, and optimization tools with intelligent agents. Their vision is to develop a self-adapting supply chain management system that "provides total visibility of the order, automates order management, and monitors product uses by customers across the network, replenishing when necessary, without having any manual intervention, other than exceptions." [44] They claim that adaptive software agents improve the effectiveness of their system in two ways:

- Agents provide support for automating information exchange, allowing instant propagation of important information to interested parties.

- Agents can actively monitor the supply chain system for critical events. They can then form intelligent responses to many of these events, enabling a faster response to many exceptional conditions.

Walsh [63] investigates market protocols for supply chain automation. He models supply chains as task dependency networks. He discusses a novel auction protocol that allows agents in the supply chain to negotiate about resource exchanges. He gives theoretical and empirical results that show that this auction protocol produces better solutions to supply chain problems than other commonly-used protocols (such as contract nets) when resource contention is an issue.

The MAGNET system [9] is intended as a testbed for multiagent negotiation. One of the major motivations behind this research is the automation of supply chains. MAGNET allows self-interested agents to negotiate over complex, coordinated tasks with precedence and time constraints. These negotiations are carried out in an auction-based market environment. Collins [10] investigates algorithms for determining winners in MAGNET combinatorial auctions, and the issues involved in implementing a customer agent for this environment.

3.3 Economics and Game Theory

There are a variety of interesting theories in the field of economics that can be used to analyze the TAC SCM game. The following sections discuss individual rationality, on which much of modern economic theory is based, Pareto efficiency, which allows one to decide how efficient an economic system is, and game theory, which is useful for reasoning about multiplayer games and the characteristics of multiagent systems.

3.3.1 Individual Rationality

Much of the work that has been done in the fields of economics and game theory assumes that participants in an economic system are individually rational. In simple terms, individual rationality implies that each participant (or agent) in the system has some preferences, which can be expressed as a function mapping possible world states to utility values. A rational agent will then act in a manner such that its personal utility is maximized. McCain [32] gives two reasons why the assumption of individual rationality is often used in the field of economics.

1. Individual rationality narrows the scope of the problem: it is substantially easier to

predict the behavior of a rational agent than it is to predict the behavior of an irrational agent.

2. Individual rationality provides a way to assess the efficiency of an economic system. If an economic system causes a reduction in the rewards available to one agent, without producing compensating rewards to others, the system is flawed in the sense that the sum of all the agents' utilities is not maximized.

There are some who have argued that assumptions of individual rationality are not justified. Tversky and Kahneman [60] studied cognitive anomalies: situations in which individuals exhibit demonstrably irrational behavior.

In [33], McFadden argues that cognitive anomalies are due to errors in perception that arise from the way the human mind stores and retrieves information and to errors in process that lead to inconsistent formulation of problems. He discusses how these effects influence individuals' economic behavior and the implications of these anomalies to contemporary economic analysis.

3.3.2 Pareto Efficiency

McCain suggests that the efficiency of an economic system can be assessed if it is assumed that agents are individually rational. A commonly-used criterion for evaluating outcomes of an economic system is Pareto efficiency [38]. An outcome is said to be Pareto efficient if there is no other outcome in which some individual is better off and no individual is worse off. A similar measure is Pareto domination: an outcome o_1 is said to Pareto dominate an outcome o_2 if someone is better off in o_1 than in o_2 and no one is worse off in o_1. An economic system or mechanism is said to be Pareto efficient if the mechanism guarantees that it will produce an outcome which is Pareto efficient. Sandholm [43], in discussing mechanism design, notes that Pareto efficiency is a measure of global good that does not require inter-agent utility comparisons. This is a useful because two different agents may exhibit the same preferences (i.e., have the same preference ordering over possible outcomes) even if their utility values for each of these outcomes vary substantially.

3.3.3 Game Theory

There are variety of principles from the field of game theory that could be useful for analysis of the TAC SCM game. Some games are said to have dominant strategies. This indicates that some agents playing the game are best off when following a certain strategy regardless of the strategies used by the other agents. If all the agents have dominant strategies, they will follow these strategies (if we assume that each agent's actions are rational.) A game in which all agents are following the dominant strategy is said to be in dominant strategy equilibrium. However, in many games, an agent's best strategy depends on the strategies chosen by the other agents. If this is the case, a dominant strategy equilibrium will not occur.

Another equilibrium concept is that of the Nash equilibrium [36]. The Nash equilibrium is a generalization of the dominant strategy equilibrium. A game is said to be in Nash equilibrium if, given the strategies that all other agents are currently following, there is no incentive for any agent to change the strategy it is currently following. The main problem with the concept of Nash equilibrium is that games are not guaranteed to have a single Nash equilibrium. Some games may not have a Nash equilibrium [17], while others have multiple Nash equilibria [26]. In the latter case, it is not generally possible to decide which equilibrium the agents should (or will) play.

Concepts of equilibrium are useful because they allow for the analysis of agents' strategies in games. A game or mechanism is typically considered stable if it exhibits a Nash or dominant strategy equilibrium. However, Sandholm [43] notes that the Nash equilibrium does not guarantee stability if the agents are able to coordinate and change strategies simultaneously. Therefore, an environment which allows this sort of collusion is not guaranteed to be stable.

Additional results from the field of game theory are discussed below in the section on game playing (3.4.1).

3.4 Artificial Intelligence Techniques

It is nearly certain that most TAC SCM agents will utilize techniques from the field of artificial intelligence. The AI community has developed many algorithms and strategies which may be useful for approaching aspects of the TAC SCM game. Specific subfields of artificial intelligence that might be relevant to TAC SCM include game playing, machine learning, and planning.

3.4.1 Game Playing

Russell and Norvig [42] give a good introduction to the basics of game-playing. They present descriptions and pseudocode for several commonly-used algorithms and techniques. Minimax search is one such algorithm; it was investigated by Von Neumann and Morgenstern [37] as a technique for mathematically evaluating the value of a game position. The basic minimax technique generates the entire game tree, evaluates the terminal states of the game, and propagates these values up to a node representing the current game state, under the assumption that both players will play perfectly. The straightforward application of minimax search is typically not practical in interesting domains, because it is not computationally feasible to generate the entire game tree. Shannon [48] added the idea of a search-depth cutoff and evaluation functions to the minimax technique; this allows for informed, but imperfect play. Computational time can be further reduced by using a technique known as alpha-beta [6], which prunes branches from the search tree that will not affect the final evaluation. Search techniques based on minimax assume that the rules of the game, the results of each move, and the full state of the game are perfectly known by the players. When these assumptions turn out to be incorrect, more sophisticated algorithms are needed.

If an element of chance determines the available moves or the results of each move, such as in backgammon, the game tree needs to include chance nodes to account for this randomness. This modification to minimax, called expectimax, was proposed by Donald Michie [34]; a big problem with the inclusion of chance nodes is that they add additional complexity to the search tree, which can make it impractical to search deeply. Ballard proposed a modification to expectimax that allows alpha-beta pruning on expectimax trees [3].

If the full state of the game is not perfectly known (as happens in many card games, such as poker), standard game-playing algorithms based on minimax are not likely to be feasible. In a fairly simple game such as blackjack, there is a limited amount of hidden information, so it is possible to use chance nodes to model an agent's uncertainty about the game state. For a more complex game, the amount of hidden information may render this approach computationally infeasible.

In the field of generic game-playing, the rules of the game are not necessarily previously known–either to the agent or to its programmers. This is another challenge for standard game-playing algorithms such as minimax. Pell proposes such a system, called Meta-Game Playing (Metagame) in [39]. In Metagame, the rules are given to the agent at runtime, and the agent is expected to use the rules as constraints for forming a successful strategy.

Levinson also discusses the topic of generic game-playing [29]. Levinson details a system called MorphII, which is a generic game-player written in the C++ programming language. It is capable of both blind learning and informed learning. Results of the system on a variety of simple games are presented and analyzed.

3.4.2 Machine Learning

Russell and Norvig's book [42] contains a broad discussion of machine learning, including techniques such as neural networks, decision trees, and reinforcement learning. Mitchell presents a more in-depth discussion of these and other topics in the field of machine learning [35]. He defines the field of machine learning as the study of "computer algorithms that improve automatically through experience." He discusses the issues in designing a system that learns and suggests a methodology for framing learning problems such that they are well-posed and can be solved through a modular design.

The most common characterization of machine learning is that of on-line learning, in which the agent learns by observing the results of its actions and attempting to determine which actions lead to good outcomes. It then updates its behavior in an attempt to improve its performance. However, some problems require off-line machine learning, in which the agent takes in a large sample of inputs and associated outputs, known as the training set, and attempts to determine a reasonable function mapping inputs to outputs. The agent then uses the results of this learning phase to guide its actions at run time. This approach is often used in situations in which real-time decisions are essential and the learning algorithm is time-consuming.

Stone [51, 55] has extensively utilized machine learning in the RoboCup soccer simulation league. He advocates a layered learning approach, in which successive layers of learning build on the results of previous layers. Specifically, Stone uses off-line neural networks to train agents in a basic ball interception skill. A more complex skill is that of pass evaluation, in which an agent has to evaluate the probability that a pass to a given teammate will succeed. During the training phase of the pass evaluation task, the potential receivers and defenders are equipped with the ball interception skill developed during the previous phase. A decision tree is utilized to learn the pass evaluation skill, again in an off-line manner. A higher-level skill is that of pass selection, in which agents must use their pass evaluation capabilities to determine whether they should pass the ball and, if so, to which teammate. Stone notes that the utility of a particular pass selection decision depends primarily on the long-term performance of the team as a whole (i.e., whether a series of such decisions eventually leads to a score.) Stone presents a novel reinforcement learning method called TPOT-RL, which is motivated by Q-learning and explicitly intended for multiagent domains. TPOT-RL is used in an on-line manner to improve the effectiveness of the team over the course of a game. Stone verifies the efficacy of the layered learning approach empirically, through controlled simulation environments and the actual RoboCup competition. Teams from Carnegie Mellon University based on this approach won both the simulation league and the small-robot league in 1998.

Mahadevan and Theocharous [31] utilize reinforcement learning to optimize a simulated manufacturing scenario. They use an average-reward reinforcement learning algorithm, called SMART [30], to maximize delivery of finished product to consumers while keeping inventory levels low. They compare SMART with the Kanban heuristic, and show that both strategies are effective in maximizing delivery of finished product, filling 98% of customer demand. However, SMART outperforms the Kanban heuristic at minimizing inventory levels in a sample scenario by 15-30%.

Additional literature that discusses the utilization of machine learning concepts is can be found in Sections 3.5.3, 3.5.2, and 3.6.

3.4.3 Planning

In problem-solving and game-playing, the actions taken by an agent at a specific point significantly affect the actions available to the agent (and possibly its opponents) later on. This is reasonable in some domains, but many real-world problems do not contain such stringent dependencies. Planning algorithms can take advantage of this flexibility to achieve better performance. The Partial-Order Planner (POP) algorithm presented by Russell [42] utilizes a restricted version of situation calculus known as STRIPS [15] to plan actions. As its name implies, POP represents plans as a partial order of actions. Analysis shows that the POP algorithm is both sound and complete, which means that it will always find a solution if one exists and that any solution POP proposes will be a correct solution.

However, the power of the POP algorithm is limited by the expressiveness of the STRIPS language. Ideally, planners should be able to represent plans hierarchically, be able to reason about time and resource constraints, and be able to reason about more expressive operators than those allowed by the STRIPS language, such as conditional effects, nondeterministic effects, and universal quantification. Erol et al. [13] present a planning system that supports hierarchical decomposition. Russell presents a planner called HD-POP that is based on this planner. The notion of hierarchical decomposition makes intuitive sense because it seems to be the way that most people plan their actions: by breaking a large problem up into smaller subproblems and then planning the actions that will solve the subproblems. On the other hand, if the large problem is not broken up "correctly," it may be impossible to devise a consistent plan, even if the undecomposed problem does have a consistent solution. The O-PLAN algorithm [4] and its successors (e.g., PlanERS-1 [16] and OPTIMUM-AIV [1]) incorporate representations of time and resource constraints.

Tambe extends planning to the multiagent domain in [58]. He proposes a system, based on the joint intentions framework [28], that explicitly represents team goals, team plans, and models of teamwork. His model of teamwork provides domain-independent rules that outline each agent's commitments and responsibilities toward the team. He gives a procedure which an agent can use to recover from the failure of another team member. This procedure allows agents to change roles during the execution of plans, which increases the probability that these plans can be carried out successfully. He applies this model of teamwork to a helicopter combat simulation and the RoboCup simulation league, and gives an anecdotal examination of a variety of cases in which the failure-detection and recovery capabilities of this system are useful. Tambe also presents results which show that these modifications to the joint intentions framework lead to a reduction in the amount of communication required.

3.4.4 Reasoning under Uncertainty

Planning and reasoning are significantly more difficult in the presence of uncertainty. If an agent is unable to obtain complete and correct information about its environment, it needs to have the ability to deal with this uncertainty in a logical manner. Two main strategies that have been used to deal with uncertainty are conditional planning and execution monitoring. In conditional planning, also known as contingency planning, agents construct a plan that accounts for the possible unexpected events (contingencies) that could arise. The execution monitoring strategy requires that the agent monitor its progress while executing the plan. If the agent determines that the current plan has failed, it must replan, finding a new way to achieve its goals given what it now knows about the state of the world. Russell [42] outlines a possible implementation of a conditional planner and a replanning agent. He also notes that a replanning agent may wish to learn in response to failed expectations, such that its future performance can be improved.

Pollack and Horty [40] discuss the issues involved in multiagent plan management in dynamic environments. Plan management introduces additional concepts not traditionally handled by planning methods, which are typically concerned only with the generation of plans. Plan management supports the notion of commitment to plans: upon creating a plan, an agent will generally attempt to avoid creating new plans that conflict with the plan just created. However, an unexpected, serendipitous consideration might arise, such that the agent would prefer to give up its commitment to the old plan, choosing instead to take advantage of the new situation. In order to notice that such a consideration has arisen, an agent also needs to monitor its environment. Russell also indicates the need for environment monitoring in his discussion of the conditional planner and the replanning agent.

However, replanning agents need to decide which subset of the environmental features they should monitor; in general, it is too costly to attempt to continuously monitor the entire state of the world. Veloso et al. [61] propose the idea of rationale-based monitoring. In this scheme, the planning system keeps track of the rationale for the decisions it makes and monitors only the environmental changes that might affect the planning rationale. They argue that the planning system should also keep track of the reasons why it chose the current plan over other alternate plans. If the environment changes in such a way that these reasons are no longer valid, it may be beneficial to choose one of the alternate plans instead. They give experimental results which show that rationale-based planning generates robust plans and can be done in an efficient manner.

3.5 Multiagent Systems

One common way to view multiagent systems is as distributed artificial intelligence (DAI). In DAI, agents cooperate to solve AI problems in a distributed manner. Researchers in the multiagent community have developed techniques for distributed problem-solving, distributed planning, distributed search algorithms, and distributed learning, among others. Stone [52] observed that some teams used the distributed AI approach in the TAC Classic games; for example, some teams used two separate agents to bid on the different types of auctions. However, many researchers have also dealt with the possibilities that agents may be self-interested, or even competitive (such that an agent may be able to increase its own utility by hindering the utilities of other agents). In order to achieve optimal results under these assumptions, each agent generally must maintain models of the other agents in the system. In both cooperative and competitive setups, it is often necessary for agents to communicate and interact. Huhns and Stephens [21] discuss such multiagent societies, including the motivations behind such societies, properties of these societies, methods of communication, speech act theory, ontologies, and methods to achieve multiagent coordination, cooperation, and negotiation.

3.5.1 Decision-Making: Voting and Auctions

Some multiagent decision-making strategies that have been proposed include voting, auctions, and market economies. Sandholm [43] discusses a variety of such decision-making strategies under the assumption that agents are rational and self-interested. He notes that these strategies can be evaluated according to several different criteria, including social welfare, Pareto efficiency, individual rationality, stability, computational efficiency, and communication efficiency.

Sandholm discusses voting as a social choice mechanism. It is generally considered desirable that the voting protocol will lead to truthful declarations of the agents' preferences.

This may not be the case if an agent can benefit by lying about its preferences. An idea from game theory, known as mechanism design, explores the design of protocols that do lead to truthful declarations. The revelation principle [41] states that, if a negotiation protocol implements some social choice function in a Nash equilibrium, then this function is implementable in Nash equilibrium via a single-step protocol in which the agents reveal their preferences truthfully. This is true even if the original negotiation protocol required multiple steps or if the agents' strategies under the original protocol were not necessarily truthful.

Sandholm's discussion of auctions is particularly relevant to the TAC SCM scenario, since customer orders are allocated by an auction. This is a slightly different form of auction than the traditional auction, because the agent with the lowest value wins. However, the results from auction theory still apply. Auction settings can be classified into three groups, depending on how the agents value the items. In a private value auction, the value of each item is determined solely by the agent's preferences. In a common value auction, the value of each item is determined entirely by the other agents' valuations of the item. A correlated value auction is one somewhere in between, in which an agent's value is partially determined by its own preferences and partially determined by the preferences of others.

Sandholm discusses four commonly-used auction protocols: the English auction, the first-price sealed-bid auction, the Dutch auction, and the Vickrey auction. The TAC SCM scenario uses the first-price sealed-bid auction, in which each agent submits a single bid, and the bids of other agents are not revealed. The highest-bidding agent wins the auction and pays the amount of its bid. Under this auction protocol, an agent's bidding strategy needs to take into account its own valuation of the item and any information or beliefs it may have regarding the valuations of other agents. Obviously, an agent's bid should not be greater than its own valuation of the item, but it can attain the item for less than this valuation if it places a bid that is just a bit higher than the bid of the agent with the second-highest valuation. Under the first-price sealed-bid auction, it is therefore useful for agents to make strategic bids that depend upon the bidding habits of other agents. However, since the bids are sealed, it is often hard to determine the true preferences of the other agents. Furthermore, in any common value or correlated value auction, the best strategy for an agent is to bid less than its own valuation. This is due to the "winner's curse," in which an agent that bid its true valuation and won an auction knows that its valuation was too high (due to the fact that the other agents bid lower). Winning an auction in this setting therefore actually amounts to a loss of money, unless every agent bids less than its true valuation. This makes it even more difficult to determine the true valuations that other agents place on auctioned items.

Sandholm also discusses the efficiency of the allocation of items and the expected revenue of the auctioneer under each of the auction protocols. It turns out that, under reasonable constraints, each of the protocols allocates auction items Pareto efficiently to the bidders who value them most, and that all of the auction protocols give the same expected revenue to the seller.

Sandholm also discusses interrelated auctions, in which a series of different items are auctioned one at a time, and an agent's valuation of a given item depends on whether or not the agent wins some other item. He shows that, under these circumstances, inefficient allocations can occur, and that there can be incentives to counterspeculate.

3.5.2 Learning to Play Better Against an Opponent

Since the success of a TAC SCM agent is likely to depend greatly on the strategies of other agents, it is reasonable to assume that learning the behaviors of other agents may be vital to good performance.

A game-player that tries to analyze and learn the strategy of its opponent is given in Carmel and Markovitch [8]. They discuss the benefits of using a model of the opponent's strategy, and give an algorithm called M* that attempts to exploit the opponent's strategy. M* is a generalization of the standard minimax algorithm that also uses the search depth and evaluation function of the opponent to drive the search. Assuming that the search depth and evaluation function of the opponent are known perfectly, M* always selects a move with a value greater than or equal to the move that would have been returned by minimax. Pruning techniques, such as alpha-beta, which can be applied to minimax trees also apply to M* trees; however, fewer nodes will be pruned by alpha-beta unless the player's evaluation function agrees with that of the opponent. This raises concerns (not discussed in their paper) that the M* search may take significantly longer to choose a move than standard minimax search. Also, M* search assumes that the opponent's search depth and evaluation function are known. Since it is not usually the case that these features of the opponent's strategy are known, Carmel and Markovitch also present and discuss algorithms which can be used to attempt to learn these features by analyzing a history of the opponent's games.

Zeng and Sycara [64] present a decision-making model, called Bazaar, that is explicitly designed to support easy incorporation of learning. Bazaar provides a multi-issue negotiation framework. Within this framework, Zeng and Sycara model learning as a Bayesian belief update process. This Bayesian framework incorporates the knowledge and beliefs that the agent has about the environment and other agents. They give theoretical and experimental results that indicate that this learning strategy is beneficial within the context of multi-issue negotiation. They indicate that they intend to test the application of this framework to more complex domains, such as supply-chain management; however, no literature documenting such an attempt could be found.

3.5.3 Learning to Benefit from Market Conditions

Vidal and Durfee [62] investigate the use of learning models of other agents in a multiagent information economy. They consider both buying and selling agents, and classify these agents into groups based on the sophistication of their models of other agents. A 0-level agent does not model the behavior of other agents whatsoever, instead choosing to base its decisions wholly on the state of the world. A 1-level agent is aware that other agents are participating in the economy, and attempts to take this into account when making its decisions. However, 1-level agents have no idea how the other agents make their decisions, so they assume that all other agents base their decisions solely on the state of the world; that is, they model all other agents as 0-level agents. The most sophisticated agents in Vidal and Durfee's taxonomy are 2-level agents, which model all other agents as if they were 1-level agents. Vidal and Durfee briefly discuss the possibility of recursively defining n-level agents, as agents which model all other agents as (n-1)-level agents, but have not performed any in-depth analysis of any agents more sophisticated than 2-level agents. They claim that there is no obvious way to theoretically analyze how different populations of agents would interact, so they rely on empirical testing to determine the benefits gained by incorporating sophisticated models of other agents. They have performed many such tests, varying the levels of sophistication of both the buying and selling agents. In general, higher-level agents perform better than their lower-level counterparts. Through analysis of these experiments, Vidal and Durfee claim that the rewards of sophisticated modeling start to diminish as the other agents become "smarter." They hypothesize that agent designers will therefore converge to some level of agent sophistication that balances the complication of detailed models of other agents with the benefit to be gained from taking more well-informed actions.

3.6 TAC Classic

Much related work has been done recently (since 2000) on implementations of agents for the TAC Classic game. However, it is not clear to what extent the previous work done on the original TAC Classic game will be useful for development of TAC SCM agents, since the rules of the game are completely different. This section contains a review of the related work on the TAC Classic game in areas in which the results seem applicable to the TAC SCM game as well.

In the TAC Classic game, each player is a travel agent. These travel agents each have eight clients, with randomly-chosen preferences regarding the details of a trip. A complete travel package requires round-trip airline ticket reservations, hotel reservations for the duration of the trip, and tickets to a variety of entertainment events. These three types of components are sold in three different auction settings. The winner of the game is the agent that maximizes the total satisfaction of its clients. A full overview of the game can be found at the TAC 2003 web site [56].

Schapire et al. utilized machine learning in their implementation of an agent (ATTac-2001) for TAC-01 [47]. This agent uses machine learning to predict prices in the TAC Classic auctions. They concentrate on learning the prices for the hotel auctions. To do this, they used a set of data from previously played games to train the agent. The agent uses several features to predict hotel prices, including the length of time remaining in the game, the current asking price for rooms whose auctions have not closed, the selling price for rooms whose auctions have closed, the current prices of airline tickets, the number of players playing the game, and a bit vector representing the identities of the players participating in each game. During the course of the game, many of these values will be known by the agent, but some will be unknown. Schapire et al. therefore classify this problem as a conditional-density-estimation problem. That is, given what the agent knows about the world state, the agent needs to estimate a conditional distribution of prices. They present a novel learning algorithm that reduces this problem to a multiclass, multi-label classification problem and solves this problem using a modification of a boosting algorithm that they had previously developed [11]. The results of the TAC 2001 competition show the efficacy of their approach: ATTac-2001 attained the second-highest raw score in the game as well as the highest score after handicapping for the relative difficulties of pleasing each of the agents' clients in the final round. They also present additional experimental results in a controlled setting that show that this learning strategy performed better at price prediction than a variety of simpler learning strategies. These experiments also show that price prediction errors in the TAC Classic game are inversely related to final scores, with a statistical correlation of -0.88.

Stone et al. [54] discuss the use of an integer linear programming approach that optimally solves the problem of allocation of goods to TAC Classic clients. This method may be useful in determining the daily production schedule of TAC SCM agents. Cormen et al. [12] define a linear programming problem as one in which "... we are given an $m \times n$ matrix A, an m-vector b, and an n-vector c. We wish to find a vector x of n elements that maximizes the objective function $\sum_{i=1}^{n} c_i x_i$ subject to the m constraints given by $Ax \leq b$." An integer linear programming problem is a linear programming problem where the values of x_i must be chosen from the set of integers. Greenwald and Boyan [18] utilized the A^* heuristic search algorithm in their TAC Classic agent (RoxyBot) to efficiently determine allocations of goods that approximate optimality.

M. He and N. Jennings discuss the implementation of SouthamptonTAC, the third-place agent in TAC-01, in [19]. A slightly modified version of this agent took second place in TAC-02 [20]. He and Jennings also observe that the hotel auctions are the most important, yet most difficult part of the TAC Classic competition. Their agent utilizes a fuzzy reasoning approach to predict the likely clearing prices of hotel auctions. They devised a set of hand-coded fuzzy rules that describe the likely increase of the price of a hotel auction (between the current time and the auction's closing time) given what is currently known about the state of the world. This representation allows them to encode such rules as "IF the asking price of Hotel 2 is *high* AND the change in asking price of Hotel 2 in the last minute is *quick* AND the auction for Hotel 1 closed *early* in the game THEN the change in price of Hotel 2 will be *very_big*." Their post hoc analysis of the TAC-02 competition notes that the outcome of a given game depends greatly on the risk-taking behavior of the participants. Some agents are risk-averse, choosing to hold off on buying flight tickets until they have secured appropriate hotel reservations, while other agents are risk-seeking, choosing to buy many flight tickets at the very beginning of the game, when the prices of flights are lower. They classify TAC Classic games as competitive, semi-competitive, and non-competitive, in which the most competitive games contained three or more risk-seeking agents. In this competitive setting, hotel prices will typically be very high, so the flexibility of a risk-averse agent is advantageous. In the non-competitive setting, the risk-taking agents perform better, since they receive lower prices on airline tickets and do not need to pay a premium to secure appropriate hotel reservations. This observation is interesting because it seems to indicate that the best strategy in a TAC Classic game is to simply choose a different strategy than the one used by the majority of the other agents. It remains to be seen whether this will be the case in the TAC SCM game as well.

3.7 Software for the TAC SCM Game

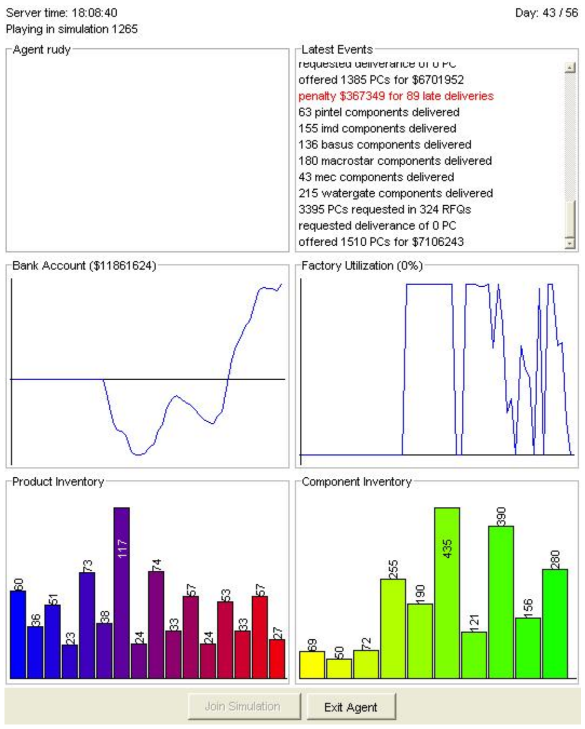
The Swedish Institute of Computer Science (SICS) was primarily responsible for developing the server on which the TAC SCM simulation runs. A connection with this server is required in order to play the TAC SCM game. The server's source code is supposed to be open to the general public, but SICS has not made it available yet. Instead, they have a server running in a public location to which agents can connect. This state of affairs is less than optimal for a variety of reasons:

- Some aspects of the TAC SCM rules are vague in the current specification of the game. Having the source of the server would allow us to quickly determine what is intended in these situations.
- Sometimes the public server is inaccessible, presumably due to network or software problems.
- We would like to be able to test some strategies and tactics without disclosing our agent's behavior to a third party.
- There are variety of parameters which affect the SCM game which should be relatively easy to change, such as the number of days in a game, the number of real-time seconds in a game, the interest rate, the average number of customer RFQs per day, and so on. For the purposes of testing, it would be preferable if we could set these parameters to

known values. This is especially important for parameters such as the interest rate and average number of RFQs per day, since they are chosen randomly at the beginning of the game and substantially affect the outcome of the game.

For these reasons, we hope that the source to the TAC SCM server will be made available to the public soon.

In addition to the server, SICS also provides a software package called AgentWare. AgentWare is written in the Java programming language and is intended as an interface for interaction with the TAC SCM server. AgentWare also provides a graphical interface which shows the status of the agent, including a graph of the agent's bank account, histograms of current inventory levels, and textual output of important game events. A screenshot of this interface is pictured in Figure 3.1. The AgentWare code also includes a default "Dummy Agent" that is primarily intended as an example of how to use the AgentWare server interface. This agent plays the SCM game through the server, following a very simple strategy.


```
Server time: 18:08:40
Playing in simulation 1265
Day: 43 / 56
```

```
requested deliverance of 0 PC
offered 1385 PCs for $6701952
penalty $367349 for 89 late deliveries
63 pintel components delivered
155 imd components delivered
136 basus components delivered
180 macrostar components delivered
43 mec components delivered
215 watergate components delivered
3395 PCs requested in 324 RFQs
requested deliverance of 0 PC
offered 1510 PCs for $7106243
```

Figure 3.1: A screen capture of AgentWare's graphical user interface.

The source code for AgentWare is not yet publicly available from the TAC web site, but the source is available to parties who express a desire to alpha-test the software on the tac-dev mailing list [57]. We therefore possess a copy of the most recent version of the AgentWare code, which we are using in the development of our own TAC SCM agent. A further discussion of the use of AgentWare in our implementation can be found below in Section 5.

Chapter 4

Analysis

This chapter presents a detailed analysis of the TAC SCM game. It presents theorems and equations relevant to the game itself, suggests metrics for the evaluation of agents, and proposes strategies that ideal TAC SCM agents could use to solve various problems. Some of these strategies may require calculations that are too complex (in terms of computation time required) to be feasible under the time constraints of the TAC SCM domain. However, we have chosen to include these strategies in this section for the sake of discussion; further work is needed to determine the actual computational times required by each strategy. We may be able to approximate optimal solutions to computationally expensive problems through the use of heuristic methods.

4.1 Interest Rate

To be precise, most of the calculations done by a TAC SCM agent need to consider the interest rate. The annual interest rate (*interest*) during a TAC SCM game is randomly chosen from a uniform distribution over the range $[10\%, 20\%]$. This interest rate is applied daily. The net effect of interest is to cause transactions occurring at the beginning of the game to be somewhat more important than transactions occurring at the end of the game. When considering the value of a transaction, an agent might want to consider the transaction's value with respect to the current timeframe or with respect to its value at the end of the game (which will be magnified due to the effect of interest). As long as an agent selects one of these conventions and uses it consistently, its decisions should be equivalent.[1] Our group has chosen to represent expected values with respect to the current time. All interest calculations presented in this thesis therefore follow this convention.

A well-known result from the field of economics gives the amount of money A accumulated after n years based on an initial investment of P and an annual interest rate r:

$$A = P \times (1 + r)^n. \tag{4.1}$$

4.2 Bottlenecks

In order to assess the performance of TAC SCM agents, it is useful to be able to determine the bottlenecks of the TAC SCM market. More formally, we can pose the following question: what factors limit the number of PCs produced in a TAC SCM game? There are basically three answers to this question: the bottleneck could be in the number of PCs demanded in customer RFQs, in the production capacity of the agents' factories, or in the amount of PC components available from suppliers.

We can predict the maximal number of PCs on a day with the following equation:

$$\mathit{production}_{PCs} = \min(\mathit{demand}_{PCs}, \mathit{production\_capacity}_{PCs}, \mathit{supplies\_available}_{PCs}). \tag{4.2}$$

We define the *bottleneck* of the TAC SCM game on some day d as the factor which limits the global production of PCs on day d. A *demand bottleneck* is in effect if $\mathit{demand}_{PCs} < \mathit{production\_capacity}_{PCs}$ and $\mathit{demand}_{PCs} < \mathit{supplies\_available}_{PCs}$; we can similarly define the analogous terms *production bottleneck* and *supply bottleneck*.

[1]This result is due to personal correspondence with Alex Babanov.

Equation 4.2 assumes that, if production capacity and available supplies are sufficient, every customer RFQ will receive at least one bid at or below its reserve price. This is a reasonable assumption because customers' reserve prices are at least double the maximum price of the components used to make the PC, so an agent which has the capability to fulfill an RFQ will never lose money by bidding at the reserve price of the RFQ.

To make use of Equation 4.2, we need to clarify what is meant by the quantities $demand_{PCs}$, $production\_capacity_{PCs}$, and $supplies\_available_{PCs}$.

We define $demand_{PCs}$ as the actual daily customer demand for PCs and can calculate this quantity by multiplying the number of RFQs per day by the mean quantity of PCs requested per RFQ:

$$demand_{PCs} = \# RFQs \times \overline{RFQ\_quantity}.^{2} \tag{4.3}$$

We define $production\_capacity_{PCs}$ as the maximum total number of PCs that can be produced daily by agents' production lines. We calculate this value under the assumption that each agent has sufficient supplies for production of these PCs. Each agent has the ability to utilize 2000 assembly cycles per day. The number of assembly cycles $cycles_i$ required to assemble a PC with type i is given in the TAC SCM specification [2]. PCs made of more expensive components have higher cycle times. The cheapest computers only require four assembly cycles, while the most expensive computers require seven. Throughout this section, we will make the assumption that each type of PCs is produced in equal quantities. Under this assumption, we can calculate the number of cycles required to assemble an "average" PC as:

$$\begin{aligned} cycles_{avg} &= \frac{\sum_{i=1}^{16} cycles_i}{16} \\ &= \frac{88}{16} \\ &= 5.5. \end{aligned} \tag{4.4}$$

[2]$\bar{x}$ denotes the mean of a sample chosen from a population with mean μ. $E[x]$ denotes the expected value of a variable x. $E[x] = E[\bar{x}] = \mu$.

We can then calculate $production\_capacity_{PCs}$ by multiplying each agent's daily production capacity by the number of agents and dividing by $cycles_{avg}$:

$$\begin{aligned} production\_capacity_{PCs} &= \frac{2000 \text{ cycles/day} \times \#\, agents}{cycles_{avg}} \\ &= \frac{2000}{5.5} \times \#\, agents. \end{aligned} \tag{4.5}$$

We define $supplies\_available_{PCs}$ as the number of PCs that can be built from the suppiles available in a day, assuming that suppliers are producing at maximal capacity. Every PC requires four supplies: one each from the categories of CPUs, motherboards, memory modules, and hard disks. We will assume that components of each of these categories are produced with equal frequency. This is a reasonable assumption because the TAC SCM rules specify that nominal production capacity is equivalent for each component category. Then $supplies\_available_{PCs}$ is equal to $supplies_{max}/4$, where $supplies_{max}$ is the maximum number of supplies produced in a day.

To calculate $supplies_{max}$, we need to determine the amount of supplies produced daily by each of the eight suppliers. A supplier i has a production capacity $capacity_i$ that is determined on each day d by a mean reverting random walk with a lower bound, as specified in the TAC SCM specification:

$$\begin{aligned} capacity_i(d) = \max(0,\, & capacity_i(d-1) \\ & + random(-0.05, 0.05) \times C_{nominal} \\ & + 0.01 \times C_{nominal} - capacity_i(d-1)). \end{aligned} \tag{4.6}$$

$C_{nominal}$ is the supplier's nominal capacity, which is specified as 500 components per day. $capacity_i(0) = C_{nominal}$, which is used in the calculation of $capacity_i(1)$, the supplier's production capacity on the first day of the game.

Given the definition of $capacity_i(d)$ and the assumption made earlier, we can now calculate

$supplies\_available_{PCs}$ for a given day d:

$$supplies\_available_{PCs}(d) = \frac{\sum_{i=1}^{8} capacity_i(d)}{4}. \tag{4.7}$$

We can now combine the results of Equations 4.2, 4.3, 4.5, and 4.7 to predict the maximal number of PCs produced in a day d:

$$\begin{aligned} production_{PCs} = \min\Bigg(& \#\, RFQs \times \overline{RFQ\_quantity}, \\ & \frac{2000}{5.5} \times \#\, agents, \\ & \frac{\sum_{i=1}^{8} capacity_i(d)}{4}\Bigg). \end{aligned} \tag{4.8}$$

4.2.1 Use of Bottlenecks in *Post Hoc* Game Analysis

Equation 4.8 has the potential to be a powerful tool for *post hoc* analysis of TAC SCM games, since all the needed values are known to the server for each day. It would be interesting to analyze the behavior of agents when the market is constrained by each of the three possible bottlenecks. It may also be interesting to note what happens at the boundary cases when the market is constrained by two of these factors or when the market is in the process of switching from one predominant bottleneck to another.

4.2.2 Bottleneck Analysis of a Typical Game

Let us consider a typical game. In such a game, the initial average number of customer RFQs is 200 per day. For simplicity, we will assume that the number of RFQs per day is always exactly 200, and that suppliers' daily capacity is always the nominal capacity of 500 components per day. We can then substitute these values into Equation 4.8 to determine the expected number of PCs produced per day:

$$\begin{aligned} production_{PCs} &= \min(demand_{PCs}, production\_capacity_{PCs}, supplies\_available_{PCs}) \\ &= \min(200 \times 10.5, \frac{2000}{5.5} \times 6, \frac{\sum_{i=1}^{8} 500}{4}). \\ &= \min(2100, 2182, 1000) \\ &= 1000. \end{aligned} \tag{4.9}$$

This result shows that the most likely bottleneck for a typical TAC SCM game is the amount of available supplies. In fact, the availability of supplies will be the most probable bottleneck as long as the number of RFQs per day is greater than 95. (If $\# RFQs = 95$, then $E[demand_{PCs}] = 95 \times 10.5 = 997.5$.) Since the initial average number of customer RFQs is chosen randomly from a uniform distribution over the range $[80, 320]$, approximately 6.25% of games will start off with a demand bottleneck. A greater number of games might enter a demand bottleneck after some period of time due to fluctuations in the average number of customer RFQs.

Agents' production capacities are not likely to be a bottleneck in any game in which there are more than two agents. However, a single agent could be limited by its production capacity if it acquires substantially more supplies than its opponents.

4.2.3 On-Line Bottleneck Prediction

In order to be of use during game play, Equation 4.8 needs to have predictive power. An agent can then use the equation to predict future market bottlenecks and plan for these contingencies. An agent which has the capability of estimating future values of $\# RFQs$ and $capacity_i$ would be able to use Equation 4.8 to plan for the future. The development of a means of estimating the future values of these parameters has not yet been pursued; this remains as an interesting area for future work.

4.3 Maximum Score of a Single Agent

Given the results presented above, we can now calculate the maximum score that an agent can possibly attain in a TAC SCM game. We will begin by bounding the score that an agent can achieve in a single day.

Theorem 1 (Maximum Daily Score of an Agent).

$$\forall a \in agents,\ daily\_score(a) \leq 2.15 \times 10^6.$$

Proof. Assume that an agent is the only participant in a game. Such an agent is then able to acquire all the available supplies if it desires and can win any customer RFQ by placing a bid at the reserve price specified by the RFQ. Given sufficiently high numbers of RFQs and supplier production levels, such an agent will be limited by the assembly capacity of its factory. This agent must therefore decide which type of PCs to produce. Since customers' reserve prices are a multiple of the base price of the components, an agent looking to maximize its profit under this scenario needs to choose to manufacture the PC that has the highest ratio of base price to production cycles required:

$$PC\_type = \arg\max_i \frac{base\_price_i}{cycles_i}. \tag{4.10}$$

Given the Bill of Materials and Component Catalog used in the TAC SCM specification, there are two values of i which maximize this equation: 5 and 13. Both of these PC types have a base price of 2150 and require 5 assembly cycles to build. The value of this ratio is therefore $2150/5 = 430$.

Given its production capacity of 2000 cycles/day, this agent can manufacture 400 such PCs per day. Assuming a sufficiently large number of RFQs, the agent can bid very near to 3 times the base price of these PCs, attaining a total revenue of 2.58×10^6 per day. By ordering in advance, this agent can receive its supplies at half of the base price, so its total costs will be 0.43×10^6 per day. Its total profit per day is $2.58 \times 10^6 - 0.43 \times 10^6 = 2.15 \times 10^6$.

□

Given Theorem 1, we can now calculate the maximum score of an agent in a TAC SCM game that is d days long. We do this by considering the maximum daily score and the effect of interest.

Theorem 2 (Maximum Final Score of an Agent).

$$\forall a \in agents, final\_score(a) \leq \sum_{i=1}^{d} 2.15 \times 10^6 \times (1 + interest)^{\frac{i}{d}}.$$

Proof. An agent's also depends on the effect of interest. We can use Equation 4.1 to find the amount of end-game profit due to the agent's profit on a day. Assume that there are i days remaining in the game. Then $P = 2.15 \times 10^6$, $r = interest$, and there are $\frac{i}{d}$ years remaining in the game. (Since interest calculations are done at the end of each day, there is still "one day remaining" even on the last day of the game.) The total amount of money accumulated at the end of the game due to the profit P attained on a day i is then:

$$A = 2.15 \times 10^6 \times (1 + interest)^{\frac{i}{d}}. \tag{4.11}$$

In Theorem 2, i represents the number of days remaining in the game; we are therefore summing the maximum profits for each day in the game, which gives us the maximum final score.

□

4.4 Maximum Total Score for a Six-Agent Game

In this section, we attempt to place an upper bound on the sum of agents' scores in a six-agent TAC game. The results from Section 4.2 suggest that, in a typical six-agent game, the bottleneck of the system will be the availability of supplies.

Theorem 3 (Approximate Maximum Total Score of a 250-Day Game).

For nearly all $g \in$ *250-day-long games,* $total\_final\_score(g) \leq 1.3 \times 10^9 \times (1 + interest)$.

Proof. Equation 4.6 is used to determine the available supplier capacity on a day d. We could obtain an upper bound on the quantity of supplies available in a game by assuming that the random number generator will always choose the value 0.05, which leads to the highest possible supplier capacity. However, this is not a very useful upper bound because it is very unlikely that the random number generator will always choose the highest possible value.

We have therefore chosen to determine an upper bound on supplier capacities by performing a simulation of supplier capacities. Each trial simulates the production capacities of the eight suppliers in a TAC game. The length of a TAC SCM game was set to 250 days, since that is the maximum value suggested in the most recent draft of the TAC SCM specification. Each day, a random value in the range $[-0.05, 0.05]$ is generated for each supplier and used to update the production capacity of that supplier according to Equation 4.6. This new capacity is then added to the total quantity produced by the supplier. At the end of the trial, the total quantities of CPUs, motherboards, memory modules, and hard disks are calculated. The maximum production of PCs in a trial is the minimum of these four quantities. This simulation was run one million times. The maximum possible production of PCs over the million runs was found to be 276,494. The results of these simulation runs suggest that 276,494 is a practical upper bound on the number of PCs that can be produced in a 250-day TAC SCM game.

Assuming a sufficiently large number of RFQs, each agent can bid very near to three times the base price of each PC produced and still have sufficient orders to sell all these PCs. Since customers' reserve prices are a multiple of the base price of the components, an agent looking to maximize its profit under this scenario needs to choose to manufacture the PC that has the highest base price:

$$PC\_type = \arg\max_i base\_price_i. \tag{4.12}$$

Given the Bill of Materials and Component Catalog used in the TAC SCM specification, there are two values of i which maximize this equation: 8 and 16. Both of these PC types have a base price of 2350. The total revenue gained by all agents over the course of a game is then $276494 \times 2350 \times 3 = 1.9492827 \times 10^9$. Since the suppliers' production capacities are full, supplies will never be acquired at a discount from the base price. The total supply cost incurred by all agents is therefore $total\_cost = 276494 \times 2350 = 0.6497609 \times 10^9$. The total profit is then 1.2995218×10^9.

We also need to take interest into consideration. For the sake of simplicity, we will assume that the entire profit is achieved on the first day of the simulation; the sum of the agents' scores at the end of the game is then:

$$total\_final\_score = 1.2995218 \times 10^9 \times (1 + interest). \tag{4.13}$$

□

The source code of the simulation used to approximate maximum PC production can be found in Appendix A. Further simulation trials were performed on 200-day-long games (because 200 days is the minimum suggested in the most recent draft of the TAC SCM specification) and 56-day-long games (because the publicly-accessible TAC SCM server is currently configured for 56-day-long games). The maximum number of PCs produced in a million trials of the 200-day-long game was 226739; the maximum number of PCs produced in a million trials of the 56-day-long game was 63540. Using reasoning similar to that given above, we can determine that the practical maximum score in a 200-day-long game is approximately bounded by $1.066 \times 10^9 \times (1 + interest)$ and that the practical maximum score in a 56-day-long game is approximately bounded by $2.986 \times 10^8 \times (1 + interest)$.

4.5 The Cost Efficiency Metric

The only significant financial cost in the TAC SCM game is the cost of raw materials, which need to be attained by requesting quotes from the appropriate suppliers. Penalties from unfulfilled customer orders can also considered as costs. One useful performance metric is what we call cost efficiency: the proportion of income to costs.

$$cost\_efficiency = \frac{revenue + interest}{supply\_costs + penalties}. \tag{4.14}$$

A higher cost efficiency directly translates into higher profit over the course of a game. Specifically, an agent's cost efficiency can be related to its profit (final score) by the following equation:

$$profit = (cost\_efficiency - 1) \times (supply\_costs + penalties). \tag{4.15}$$

This equation allows us to view the solution to the TAC SCM game as an optimization problem: we want to find the balance of cost efficiency, supply costs, and penalties such that this profit equation is maximized. (Since this is a multiagent competition, we are also interested in doing whatever we can to minimize the value of this equation for the other agents.) An agent with high cost efficiency orders raw materials inexpensively, sells finished goods at high prices, and incurs few penalties. This metric allows us to evaluate agents by comparing agents' cost efficiencies to theoretical limits.

Theorem 4 (Maximum Cost Efficiency). *The maximum cost efficiency of any agent in the TAC SCM game is 7.2.*

Proof. There are four quantities that govern cost efficiency: revenue, interest, supply costs, and penalties. All of these quantities are non-negative. To attain the maximum cost efficiency, we therefore want to maximize revenue and interest while minimizing supply costs and penalties. We assume that the supply costs are strictly positive. (If not, the agent will be unable to manufacture any PCs and is therefore not very interesting.) The prices of supplies can be discounted from their base prices by a factor of as much as 50% (see Section 2.3). An agent that seeks to minimize its supply costs would therefore only order supplies when they were available at a 50% discount. Such an agent would also aim to incur no penalties, which could be achieved by only bidding on RFQs that can be immediately fulfilled from existing inventory. Given these supplies, the agent must then act to maximize its revenue and interest. The reserve price specified in a customer RFQ for PCs of type pc_i is chosen from a uniform distribution over the interval $[2 \times base\_price(pc_i), 3 \times base\_price(pc_i)]$. An agent that maximizes cost efficiency would therefore only bid on RFQs with reserve prices very close to the upper boundary of reserve prices and would always bid the reserve price of these RFQs. Furthermore, so as to maximize interest, the agent would only bid on RFQs at the very beginning of the game. If the cost of the raw materials bought by such an agent is c, then the agent can sell the finished goods at a maximum price of $6c$ (because all the raw materials were purchased at half the base price). The parameter representing the interest rate during a game is chosen randomly with a maximum value of 20%. Given these values, we can calculate the maximum cost efficiency of any agent:

$$\forall a \in agents,\ cost\_efficiency(a) \leq \frac{6c + 6c \times 20\%}{c + 0} = \frac{7.2c}{c} = 7.2. \tag{4.16}$$

Additionally, if we know the interest rate parameter for a specific game, we can substitute this value into the formula above to obtain the maximum cost efficiency of any agent participating in that game.

□

Theorem 5 (Minimum Cost Efficiency). *The minimum cost efficiency of any agent in the TAC SCM game is 0.*

Proof. Any agent which orders some supplies or incurs some penalty, but which does not sell any finished products, will attain a cost efficiency of 0. □

For various reasons, no practical agent will attain the theoretic maximum cost efficiency. Successful agents will need to respond to RFQs throughout the majority of the game, not just at the very beginning. The effect of interest will therefore be reduced. Also, depending on the strategies of the other agents, an agent which only bids near the high end of the reserve price range may never receive any customer offers. It is also unlikely that an agent will be able to attain parts at 50% discount, since the discount is based on available capacity and available capacity is expected to be low (see Section 4.2).

A metric similar to cost efficiency that is more commonly used in economics is *profit margin*, which is defined as $\frac{profit}{total\_revenue}$ and is bounded above by 100%. We can calculate the profit margin at the end of a TAC SCM game as:

$$profit\_margin = \frac{final\_score}{revenue + interest}. \tag{4.17}$$

4.6 Agent Design

The principle of modular design is pervasive in the field of software engineering. In the actual implementation of a TAC SCM agent, we need to identify appropriate modules that can be developed and tested independently of each other. A TAC SCM agent is responsible for making decisions in four main areas: procurement of raw materials from suppliers, sales of finished goods to customers, production of finished goods from raw materials, and delivery of finished goods to customers. These reponsibilities suggest a natural decomposition of an agent into four high-level functional components: procurement, sales, production, and delivery. The next four sections investigate the detailed responsibilities of each of these components and suggest metrics that can be used to measure the performance of these components.

4.7 Procurement

The ideal procurement component would have the following characteristics:

1. It would maintain a level of supplies sufficient to meet customer demand and allow for maximum factory utilization.

2. It would forecast future demand and place requests for supplies far in advance, thus maximizing supplier discounts.

3. It would provide estimates of future product supply and prices to other components.

4. It would engage in strategic behavior, preventing other agents from obtaining their needed supplies.

5. It would keep inventory levels as low as possible.

6. It would ensure that no unused components remained in raw materials inventory at the end of the game.

These characteristics suggest strategies that can be pursued by procurement components and metrics by which the performance of procurement components can be evaluated. These strategies and metrics are discussed in the following sections.

4.7.1 Demand Fulfillment

Biswas and Narahari [5] define order fill rate as the fraction of customer orders that are immediately met from existing stock. It is not clear that this will be a useful metric for the TAC SCM game, since there is no incentive to ship orders immediately. However, it may be useful to measure the fraction of customer orders that could be immediately met from existing stock if there were such an incentive. An agent which does well with respect to this metric would be able to fulfill orders with short lead times with a low probability of incurring penalties. Note that this metric is not entirely a measure of inventory management performance, as it also involves the production component. The value of this metric is also highly correlated with the behavior of the sales component, since the sales component is responsible for controlling the level of customer orders.

A simpler metric is that of potential factory utilization: if the production component has enough raw materials to produce finished PCs at maximum utilization, the procurement component is clearly doing a good job of obtaining required components. Note that the potential factory utilization can be 100% even if the factory is not actually producing at 100% utilization. With this measure, we can measure the demand fulfillment capabilities of the procurement component independently of other components.

4.7.2 Demand Forecasting

The proposed demand fulfillment metrics measure the ability of the procurement component to meet present demand. We are also interested in measuring the ability of the procurement component to account for future demand. This is especially important since the TAC SCM specification describes supply price discounts, which allow advance supply orders to be discounted at a factor of up to 50%. The supplier price formula given in the TAC SCM specification is as follows:

$$P(d+i) = base\_price \times \left(1 - \delta p \times \frac{C_{available}(d+i)}{C_{nominal} \times i}\right), \tag{4.18}$$

where $P(d+i)$ is the price offered for a component with due date $d+i$ on any day d, δp is the price discount factor (specified as 50%), $C_{nominal}$ is the nominal capacity of the supplier (500 components per day), and $C_{available}$ is the amount of currently free capacity the supplier has between days d and $d+i$. Therefore, an agent which can accurately predict its supply needs may be able to obtain a large proportion of its supplies at half of the base price. An obvious measure of an agent's demand forecasting capabilities is the percentage of the base price that the agent actually pays for supplies during the course of the game. Of course, this metric can be misleading if the agent orders a large quantity of supplies that are never used. Section 4.7.5 provides metrics which are useful for analyzing the costs due to excess inventory.

4.7.3 Supply Availability and Price Estimation

Another important responsibility of the procurement component is the estimation of future availability and prices of supplies. These estimates provide information that is useful for the decisions made by the sales and production components.

Each agent is allowed to send up to ten RFQs to each supplier every day. Since an agent will not generally need to use this many RFQs for supply acquisition, it can use the excess RFQs as supplier queries. When the supplier makes offers on these query RFQs, the agent has no intent to accept these offers. Rather, the agent can use these offers to determine the availability of the supplier.

If the supplier expects its production capacity to be full between the current date and the due date of the RFQ, it will respond with two offers: an earliest complete offer and a partial offer. The agent can then examine the revised due date in the earliest complete offer to determine when the supplier will next have available production capacity. Since this supplier has little free capacity available, the procurement component can also estimate that the prices offered by the supplier will be very close to the base price.

If the supplier responds with a single offer, the agent knows that the supplier currently expects to have free production capacity. The agent can also update its supply-price estimates to reflect the offer price provided in the offer. Furthermore, the agent can use this price along with the supplier pricing formula (Equation 4.18) to deduce the supplier's level of expected production capacity.

The procurement component can also use the periodic reports of market state to estimate the production levels of suppliers. Although market reports are only produced every twenty days, they provide one item of information that can not be ascertained through sending supplier queries: the quantities of each component produced and sold by suppliers. (Supplier queries can only be used to tell whether suppliers are busy, not which components the suppliers have actually been producing.)

4.7.4 Strategic Procurement Behavior

There are significant incentives for procurement components to engage in strategic behavior. Since availability of raw materials is likely to be the limiting factor in most TAC SCM games (see Section 4.2), the agent which secures the largest proportion of the available supply should have a marked advantage over its opponents. An agent can monopolize the supply for some component by simply placing an extremely large order on the first day of the game. If the agent is lucky (luck is required since suppliers consider incoming RFQs in a random order), it will be able to monopolize the production capacity of the supplier for a large fraction of the game. Any agent which can manage to monopolize the supply of some PC component is guaranteed to be able to win all bids for RFQs that require that component, and can do so while bidding at the customer's reserve price. Furthermore, an agent which monopolizes an entire class of components (e.g., all CPUs, all memory modules, etc.) can prevent all other agents from being able to manufacture any PCs at all.

Given the significant incentives for strategic behavior in inventory management, successful agents will most likely engage in such behavior. In addition, they will need to be able to detect and respond to the strategic behaviors of other agents. One way to detect supplier monopolization is through the supply availability estimation technique detailed in Section 4.7.3.

4.7.5 Minimization of Inventory Costs

In the TAC SCM game, there is no extra cost associated with keeping large inventories of raw materials or finished goods. However, to take advantage of the interest rate, an agent should not hold items in inventory for a long period of time. Doing so prevents the agent from obtaining interest on the revenue gained by selling the items, effectively causing a depreciation in the value of these items. The procurement component should therefore attempt to minimize the amount of time that raw materials are kept in inventory.

At the end of the game, any unused raw materials or finished goods represent a loss because the agent could have achieved greater profit by not ordering these excess items. (This is not strictly true if part of our motivation for ordering these items was to prevent other agents from possessing them.) We can express this loss as the sum of two separate quantities: $excess\_rm\_loss$, which denotes the sum of the order prices of all excess raw materials, and $excess\_fg\_loss$, which denotes the sum of the order prices of the raw materials used to manufacture all excess finished goods. The procurement component should attempt to minimize the value of $excess\_rm\_loss$.

4.8 Sales

The ideal sales component would have the following characteristics:

1. It would secure customer orders to match existing inventory and projected future inventory.

2. It would forecast the trend underlying daily RFQ quantities.

3. It would learn the mapping from offer prices to the probability that a customer will accept the offer.

4. It would provide estimates of future PC demand and order prices to other components.

5. It would minimize losses due to late or missed deliveries.

6. It would engage in strategic behavior.

7. It would keep inventory levels as low as possible.

8. It would ensure that no unused PCs remained in finished goods inventory at the end of the game.

These characteristics suggest strategies that can be pursued by sales components and metrics by which the performance of sales components can be evaluated. These strategies and metrics are discussed in the following sections.

4.8.1 Order Securement

The order securement component of an agent is responsible for deciding which customer RFQs to bid on and at what prices. In this section we characterize the behavior of an ideal order securement component.

Define the expected profit of a single offer x as the expected profit of the order o that results if the customer accepts the offer, multiplied by the probability that x is accepted:

$$E[profit(x)] = E[profit(o)] \times P(acceptance(x)). \tag{4.19}$$

The expected profit of an order o is:

$$\begin{aligned} E[profit(o)] =& offer\_price(o) \times (1 + interest)^{\frac{days - due\_date(o)}{days}} \\ & - supply\_costs(o) - E[late\_loss]. \end{aligned} \tag{4.20}$$

Equation 4.26, developed in Section 4.8.4, can be used to determine $E[late\_loss]$, the expected loss due to late delivery.

The order securement responsibility of an ideal sales component can then be characterized as follows. Given a set of RFQs R, 16-vectors Q_{min} and Q_{max} representing desired minimum and maximum sales levels for each PC type, and a confidence level $c \in [0 \ldots 1]$, return a set of offers O that satisfies the following constraints, or fail:

1. Consider the set C of customer orders that are received as a result of the offers O. Let Q_{actual} be a 16-vector representing the sum of the quantities ordered in C for each PC type. For $i = \{1, \ldots, 16\}$, assert that $Q_{min_i} \leq Q_{actual_i} \leq Q_{max_i}$ with confidence $\geq c$.

2. O maximizes expected profit from customer orders. More formally, O is the minimal set that maximizes the quantity $\sum_{x \in O} E[profit(x)]$.

It is assumed that the values for Q_{min}, Q_{max}, and c are provided by some external source. These values are probably chosen based on expected demand and order prices, expected supply and supply costs, and the current status of raw materials inventory, finished goods inventory, and the agent's factory utilization. If the order securement module is unable to find an O that satisfies the constraints, the external source may want to consider relaxing the acceptable sales levels or desired confidence level.

Given the large number of RFQs per day and the fact that agents only have fifteen seconds to make a day's decisions, an optimal solution to the order securement problem is probably computationally infeasible. Future work will investigate the development of heuristics that approximate optimality and can be used to solve these constraints efficiently.

4.8.2 RFQ Trend Prediction

The number of RFQs per day is determined by a Poisson distribution over the average number of customer RFQs (RFQ_{avg}):

$$\# RFQ = poisson(RFQ_{avg}). \tag{4.21}$$

RFQ_{avg} is chosen at the beginning of the game from a uniform distribution over the interval $[80, 320]$, and is updated daily by a trend. This trend is initially zero and is updated daily by a bounded random walk:

$$RFQ_{avg} = RFQ_{avg} \times trend, \tag{4.22}$$

$$trend = \max(T_{min}, \min(T_{min}, trend + random(-0.01, 0.01)), \tag{4.23}$$

where $T_{min} = 0.95$ and $T_{max} = 1/0.95$. Given a sample of actual numbers of RFQs per day, an agent may be able to determine the most probable values of RFQ_{avg} and $trend$. These values could then be used to predict future demand.

4.8.3 Customer Order Probabilities

To attain an acceptable solution to the order securement problem, an agent must be able to accurately estimate the probabilities that customers will accept offers. There are a variety of factors that may influence this probability, including the offer price, the lead time of the RFQ, the penalty specified in the RFQ, availability of supplies, and the strategies of other agents.

The Effect of Order-to-Delivery Lead Time

In the TAC SCM game, customers' RFQs specify due dates that are three to twelve days in the future. This could be viewed as the customer's order-to-delivery lead time preference. It is reasonable to assume that, all else being the same, RFQs specifying lower lead times will command higher prices. This is because agents that win bids with very short lead times are essentially taking on a higher risk of incurring penalty charges. This effect will therefore be especially pronounced when the penalty percentage is high. These agents are also committing themselves to a schedule with more stringent constraints.

Conversely, if all else is the same, an RFQ with an early due date allows an agent to get paid earlier. Due to the effect of interest, this will translate into a higher profit. It is not yet clear whether this effect will be significant.

Shipping finished goods early does not result in any benefit, so there is no incentive to ship earlier than the day before the due date. There is actually an incentive to avoid shipping early: the agent might be able to put in a successful bid on a high-priced, low-lead-time RFQ and sell the (already completed) PCs to another customer at a premium. The agent could then use remaining supplies and assembly capacity to fulfill the original customer's order within a reasonable time.

Since there is no incentive to ship finished goods early, order-to-delivery lead time is not an interesting metric for evaluation of TAC SCM agents; the probability of on-time delivery is of far greater importance. However, determining the relationship between lead time and order price is essential for the evaluation of customer RFQs.

Machine Learning in Order Securement

There are a variety of factors that influence the probabilities of customer orders. These factors are partially dependent on one another and not necessarily known to an agent's developers *a priori*. Given the importance of RFQ evaluation, it is very likely that successful agents will use on-line machine learning to aid in this process.

In the simplest case, we can assume that the order probability of an offer o is based only on the offer price and the values specified in the RFQ: quantity, lead time, reserve price, and penalty. We can then create a 5-dimensional matrix $Q = offer\_price \times quantity \times lead\_time \times reserve\_price \times penalty$. Each entry in Q contains two values: the number of offers the agent has made on RFQs with the associated parameter values and the number of times that these offers have been accepted by customers. We can seed each slot in Q with a small number of fake "offers" and a randomly-chosen number of acceptances, and update the appropriate slots every time a bundle of offers is sent or a bundles of orders is received. To calculate $P(acceptance(o))$, we look up the appropriate slot in Q and divide the number of orders by the number of offers. We may want to keep Q small by partitioning the values of offer prices, reserve prices, and penalties into ranges.

The biggest problem with this approach is that this method considers all offers equally. Since the state of the market will change over the course of the game, it is likely that the results of the most recent offers are much more relevant. To solve this, we could use a reinforcement learning approach, such as Q-learning, that assigns rewards to an agent each time an offer is accepted. This reward can also be correlated with the profit gained by the order such that the agent is enticed to continue making offers on profitable categories of RFQs.

The periodic market reports produced by the server may also be useful in learning the order acceptance rate.

4.8.4 Late Deliveries

An acceptable solution to the order securement problem must also consider the losses associated with late deliveries. This section develops results that can be used to determine the expected loss due to late delivery of any customer order.

Customer Satisfaction

In the TAC scenario, customers' satisfaction levels are actually fairly easy to quantify. This is at least partially due to the fact that customers have no brand loyalty or preferred suppliers since customers always choose the lowest bidder. We can consider a customer to be completely satisfied if a complete order is received by the due date. If an order is late, the customer is unsatisfied by a quantifiable amount: the daily late-delivery penalty. This penalty is given in the original customer RFQ and ranges from 5% to 15% of the customer's reserve price (chosen from a uniform distribution). If an order is incomplete or more than five days late, the customer is maximally unsatisfied and cancels the order without paying. The penalties incurred over the last five days still apply. We can therefore write an equation that explicitly quantifies the customer's satisfaction level. A reasonable measure of customer satisfaction is the fraction of the agreed order price which the customer actually pays:

$$satisfaction = \begin{cases} 1 & \text{if delivery is on time} \\ \frac{price - penalty \times n}{price} & \text{if delivery is } 1 \leq n \leq 5 \text{ days late} \\ \frac{-5 \times penalty}{price} & \text{if delivery is more than 5 days late} \end{cases} \tag{4.24}$$

Note that the satisfaction of the customer can be negative if the delivery is more than five days late or if the daily penalty is greater than 20% of the order price.

Losses Due to Late Delivery

If an agent delivers an order n days late, it will lose some money due to late delivery. Each customer RFQ specifies a daily penalty p which is immediately subtracted from an agent's bank account if the order is not delivered on time. In addition, the agent needs to consider the interest charged on these penalties over the last $n - 1$ days and the loss of interest that could have accrued if the agent had delivered the order on time. We can therefore express the loss $late\_loss_n$ due to being n days late as the sum of these three quantities, if $n \leq 5$. We will assume the loss for $n > 5$ is the same as the loss for $n = 5$. This is a simplification since the order does not ever get delivered and the agent therefore essentially loses the order price and all future interest on the order price. On the other hand, the components needed for that order still exist in inventory, so the agent can use them to fulfill another order. The true loss calculation in this case is therefore not straightforward; further work is needed to theoretically or experimentally determine this loss.

$$late\_loss_n = \begin{cases} n \times p + \left(int_d \times p \times \sum_{i=1}^{n-1} i\right) + int_d \times order\_price \times n & \text{if } 0 \leq n \leq 5 \\ \sim late\_loss_5 & \text{if } n > 5, \end{cases} \tag{4.25}$$

where int_d denotes the daily interest rate.

Probability of n-Day-Late Delivery

In the TAC SCM game, it makes sense to measure the probability $P(late_n)$ of n-day-late delivery, where $0 \leq n \leq 5$, and the probability $P(late_\infty)$ of being more than 5 days late, which is $1 - \sum_{i=0}^{5} P(late_i)$. $P(late_0)$ is the probability of on-time delivery. These quantities are essential for accurate prediction of the expected loss due to late delivery. If we know (or can accurately estimate) these probabilities, then we can use Equation 4.25 to calculate the expected loss due to late delivery $E[late\_loss]$ of any order:

$$E[late\_loss] = \left(\sum_{i=1}^{5} P(late_i) \times late\_loss_i \right) + P(late_\infty) \times late\_loss_\infty. \tag{4.26}$$

This quantity can be used to more accurately estimate the expected profit of a bid. This is another area where machine learning could be utilized: the probability of on-time delivery likely depends on the lead time of the order, the amount of available (and projected) supplies, and the quantity of PCs required by other outstanding orders.

4.8.5 Strategic Sales Behavior

There are also incentives for sales components to behave strategically. The RFQ bidding process is a first-price, sealed-bid, correlated-value auction. As discussed in Section 3.5.1, agents can achieve greater utility in these auctions by making strategic bids that depend on the bidding habits of other agents. An ideal sales component would therefore model other agents' bidding behavior and make strategic bids.

4.8.6 Minimization of Inventory Costs

The sales component should also do its part to keep inventory levels low and to ensure that the loss due to excess finished goods ($excess\_fg\_loss$) is minimized. See Section 4.7.5 for further discussion of this topic.

4.9 Production

The responsibility of the production component can be stated as follows: given a set R of raw materials, produce a set F of finished goods with maximal expected utility. It is assumed that some outside source determines the utility of producing PCs of each type.

If we assume that the utility $\mathit{utility}_i$ of producing a single PC of type PC_i does not depend on the production of other PCs, we can state the production problem as an integer linear programming problem.[3]

Let b be an 11-vector such that $b_i = \mathit{quantity}(\mathit{component}_i)$ for $1 \leq i \leq 10$ and $b_{11} = \#\mathit{assembly\_cycles\_available}$. Let c be a 16-vector such that $c_i = \mathit{utility}_i$. Let A be an 11×16 matrix such that $A_{i,j} =$ the quantity of $\mathit{component}_i$ needed to manufacture PC_j for $1 \leq i \leq 10$ and $A_{11,j} =$ the number of assembly cycles needed to manufacture PC_j.

We then wish to find a 16-vector x that maximizes the sum $\sum_{i=1}^{16} c_i x_i$ subject to the 11 constraints given by $Ax \leq b$. The values of x must be chosen from the set of non-negative integers. This is a full formulation of an integer linear programming problem.

Given this formulation, we can use standard integer linear programming tools to find an optimal value for x. Unfortunately, general integer linear programming problems are NP-complete, so we are not guaranteed to be able to find a solution efficiently. However, integer linear programming was used successfully by Stone et al. [54] to solve the TAC Classic allocation problem. Also, the size of this problem has a constant bound, since A is 11×16. We therefore hope that this approach can be used in the TAC SCM game as well. If not, a heuristic searching strategy, such as that used by Greenwald and Boyan [18] in their implementation of RoxyBot, may achieve near-optimal performance in a computationally efficient manner.

[3]A definition of this term is presented in Section 3.6.

4.10 Delivery

The responsibility of the delivery component can be stated as follows: given a set F of finished goods, produce a set D of deliveries with maximal expected utility. We can define the utility $utility_i$ of delivering the finished products required for order o_i as:

$$utility_i = \begin{cases} -\epsilon & \text{if } today + 1 < due\_date \leq today - 5 \\ late\_loss_n(o_i) & \text{if not delivering } o_i \text{ today would cause } o_i \text{ to be } n \text{ days late.} \end{cases} \tag{4.27}$$

As discussed in Section 4.8.3, there is no incentive for delivering early, and there is some incentive to not deliver early, so $utility_i = -\epsilon$ if the order is not yet due. Also, $utility_i = 0$ if the order is already at least five days late, because the customer will cancel the order and we will get no money even if we choose to deliver today.

Since each customer order only requires one type of PC, we can consider delivery for each type of PC separately. We will choose to take this approach. The delivery component will therefore solve sixteen delivery subproblems each day.

Let O_k be the set of outstanding customer orders that require PCs of type PC_k, and let n be the size of O_k. Let W be the quantity of PC_k available in finished goods inventory. Let v_i be the utility of fulfilling the ith order in O_k and w_i be the quantity specified by the ith order in O_k. We want to choose some subset of O_k such that $\sum v_i$ is maximized and $\sum w_i \leq W$. This is equivalent to the 0-1 knapsack problem [12], which can be solved with a dynamic programming algorithm that runs in $O(nW)$ time.

4.11 Miscellaneous Performance Metrics

Supply chain performance metrics suggested by Biswas and Narahari [5] were presented in Section 3.1. Most of these metrics have already been included in our analysis. Two metrics that have not yet been discussed are product quality and supply chain lead time.

4.11.1 Product Quality

The TAC SCM scenario makes no distinction between product qualities. Since all goods produced by all agents are of equivalent quality, product quality is not a useful metric for evaluating agents.

4.11.2 Supply Chain Lead Time

In general, this is the time required for the supply chain to acquire raw materials, plus the amount of time required to convert raw materials into finished goods, plus the amount of time required to ship finished goods to the customer. Since the time required to ship finished goods is a constant (one day) in the TAC SCM game, we are only interested in considering the time required to attain and process raw materials.

Chapter 5

Implementation

This chapter discusses the current status of our implementation of a TAC SCM agent. To date, much of our implementation has focused on low-level issues such as communication with the TAC server and the development of a software framework for the TAC agent. The abilities of the agent presented here are therefore rather limited, especially when compared to those of the ideal agent explored in the previous chapter. Nevertheless, this agent performs all the tasks required of a TAC SCM agent and does so in a reasonable manner. It therefore serves as a useful benchmark which we can use to test the performance of more advanced agents. Furthermore, our agent has been implemented in a modular, component-based manner, such that more advanced modules can be developed and added to the agent easily.

5.1 Procurement

The current implementation of our agent could be characterized as supply-driven. By this we mean that all aspects of the agent's processing depends on the acquisition of supplies. It is therefore not surprising that the performance of our agent depends heavily on the performance of the procurement component. We have implemented a *bounded inventory manager* component that attempts to keep the level of each type of raw materials within

```
MANAGE_INVENTORY(V: vector of current raw materials inventory levels,
                 Min: vector containing desired minimum inventory levels,
                 Max: vector containing desired maximum inventory levels):
for i <- 1 to length[V]
  if V[i] < Min[i]
    ORDER_COMPONENTS(V, i, Max[i] - V[i])
  end
end

ORDER_COMPONENTS(V: vector of current raw materials inventory levels,
                 c: component ID,
                 n: quantity to order):
supplier <- pick random supplier for component i
for i <- 1 to 5
  send RFQ to supplier requesting n / 5 units of c to be delivered tomorrow
end
V[c] <- V[c] + n
```

Figure 5.1: The algorithm used by the bounded inventory manager.

user-determined bounds. It does this using the simple algorithm presented in Figure 5.1. This algorithm essentially implements a bang-bang control scheme.

For each RFQ it sends out, the bounded inventory manager always accepts a supplier offer. If the desired quantity of supplies is not available by the due date, the bounded inventory manager always accepts the earliest complete offer. Whenever extra supply for some component is needed, the agent chooses a random supplier that produces that type of component. It then sends five requests to this supplier, each specifying 1/5 of the desired quantity. This is done in case the supplier isn't able to deliver the full order on the due date. If this is the case, the five requests might allow the agent to receive some fraction of the order earlier than if it had just sent a single RFQ.

One desirable characteristic of the bounded inventory manager is that it allows us to determine an *a priori* upper bound on the loss due to excess raw materials inventory at the end of the game. (See Section 4.7.5 for further details.)

Theorem 6 (Maximum *excess_rm_loss* for an Agent with a Bounded Inventory

Manager).

$$excess\_rm\_loss \leq \sum_{i=1}^{10} base\_price(i) \times Max[i].$$

Proof. In the worst case, an agent with a bounded inventory manager will have $Max[i]$ units available for every component i at the end of the game. The maximum price the agent can pay for a unit of component i is $base\_price(i)$. The worst-case total price for all the components remaining in raw materials inventory is therefore $\sum_{i=1}^{10} base\_price(i) \times Max[i]$. □

5.2 Production

The *simple production manager* creates a production schedule given available supplies. It does this by iterating over the possible PC types and building a unit of each PC for which the required raw materials are available. It repeats this process until its production capacity is maximized or the raw materials inventory has been depleted to the point that it is no longer possible to create another PC.

5.3 Sales

The *simple sales manager* uses the finished goods inventory to make offers on RFQs. For each RFQ, it checks if the PCs necessary to fulfill the order are available in inventory. If so, it places a bid for the RFQ at 90-100% of the RFQ's reserve price. It then removes some quantity of finished goods from consideration such that it does not overcommit available inventory. This quantity is determined by multiplying the number of PCs requested in the RFQ by the probability P that the customer will accept this offer. P is currently a single scalar value hard-coded into the simple sales manager; a value of 0.5 is currently used. Experimental results have shown that this quantity seems to provide a good tradeoff between overcommitment and buildup of excess finished goods inventory.

5.4 Delivery

The *simple delivery manager* takes in a list of outstanding customer orders and produces a delivery schedule. It does so by considering each order and checking whether there is currently available finished goods to fulfill the order. If so, it adds the order to the delivery schedule and decreases the quantity of finished goods inventory by the appropriate amount.

Chapter 6

Experimental Results

We have experimentally tested our current implementation of a TAC agent by running it on the publicly-accessible TAC SCM server. This server is currently configured for 56-day-long games, which is substantially shorter than the 200- to 250-day games suggested by the TAC SCM specification. This allows for easier testing of agents since the play of an entire game takes only 15 minutes. Our agent played a total of 260 games over a period of three days. In each game, the agent played against five other agents. The strategy used by our agent is that described in Chapter 5, with minimum and maximum inventory bounds of 750 and 1000 for each of the four types of CPUs and 1500 and 2000 for each other type of component. In 54 of these games, one or two of our agent's opponents were agents developed by other TAC SCM teams. The remaining opponents were all dummy agents. These dummy agents are created automatically by the server when a game starts that has fewer than six participants. The dummy agents perform all the tasks required of TAC SCM agents using a very simple strategy. The results of these experiments are summarized in Figures 6.1, 6.2, 6.3, and 6.4. For the purposes of these experiments, our agent is named `rudy`, after one of our team members. We intend to choose a different name at some point in the future.

These results show that even our simple agent is currently performing well with respect to other teams' agents. Our agent was first place in every game but one of the two games

Agent Name	Games Played	Standing	Final Score
rudy	260	1.004	44.1 million
utexas	2	1.500	15.1 million
Dummy-2	260	3.935	8.86 million
Dummy-4	259	3.965	8.47 million
Dummy-5	206	3.971	8.54 million
Dummy	260	3.973	8.91 million
temp	2	4.000	-1.03 million
Dummy-3	260	4.012	8.96 million
BlueLight	46	4.652	2.71 million
omalley	1	6.000	-48.6 million
viswanath	4	6.000	-64.0 million

Figure 6.1: Summary of game outcomes. The number of games played by each agent is included in this table along with the agent's mean final standing (where 1 denotes the winner) and mean final score. In this figure and the others in this chapter, agents are presented in order of their mean standing.

Agent Name	Revenue	Interest	Supply Costs	Penalties
rudy	60.3×10^6	0.174×10^6	36.4×10^6	0.677×10^6
utexas	45.7×10^6	-0.0229×10^6	28.4×10^6	2.17×10^6
Dummy-2	43.0×10^6	0.0723×10^6	21.7×10^6	12.5×10^6
Dummy-4	43.1×10^6	0.0723×10^6	21.9×10^6	12.8×10^6
Dummy-5	41.9×10^6	0.0723×10^6	21.4×10^6	12.1×10^6
Dummy	43.0×10^6	0.0736×10^6	21.6×10^6	12.5×10^6
temp	27.0×10^6	0.0294×10^6	15.0×10^6	13.0×10^6
Dummy-3	43.2×10^6	0.0750×10^6	21.8×10^6	12.5×10^6
BlueLight	25.8×10^6	0.00759×10^6	15.3×10^6	7.80×10^6
omalley	36.5×10^6	-0.231×10^6	38.5×10^6	46.3×10^6
viswanath	0.0×10^6	-0.393×10^6	16.5×10^6	47.1×10^6

Figure 6.2: Summary of Income and Expenses. Each agent's mean revenue, interest, supply costs, and penalties are presented in this table.

Agent Name	Orders	On-Time	Late	Missed	% On-Time
`rudy`	2110	2105	0.86	4.2	99.8%
`utexas`	1013	859	125	29	89.2%
`Dummy-2`	1243	540	462	242	51.6%
`Dummy-4`	1260	548	455	256	52.5%
`Dummy-5`	1216	538	440	238	53.3%
`Dummy`	1248	544	457	247	52.1%
`temp`	866	193	403	270	11.1%
`Dummy-3`	1252	544	465	243	52.1%
`BlueLight`	860	543	62	255	47.5%
`omalley`	2459	604	274	1581	24.6%
`viswanath`	1795	0	0	1795	0.0%

Figure 6.3: Summary of delivery statistics. Each agent's mean number of orders, on-time deliveries, late deliveries, missed deliveries, and percentage of on-time deliveries are presented in this table.

Agent Name	Factory Utilization	Cost Efficiency	Profit Margin
`rudy`	89.4%	2.21	54.7%
`utexas`	52.5%	1.47	32.0%
`Dummy-2`	51.3%	1.55	25.8%
`Dummy-4`	51.3%	1.56	25.4%
`Dummy-5`	50.0%	1.59	26.6%
`Dummy`	51.2%	1.57	25.8%
`temp`	31.5%	0.48	-1.3%
`Dummy-3`	51.5%	1.57	26.2%
`BlueLight`	31.2%	0.94	-134.3%
`omalley`	50.0%	0.43	-132.5%
`viswanath`	1.8%	0.0	0.0%

Figure 6.4: Summary of performance metrics. Each agent's factory utilization, mean cost efficiency, and profit margin are presented in this table.

against `utexas`. In this game, our agent placed second. The results strongly suggest that only our agent and `utexas` are currently more sophisticated than the dummy agents.

The results highlight one of the main advantages of our supply-driven approach: our agent is consistently able to meet deadlines. Even though our agent obtained roughly twice as many orders as other agents, its on-time delivery percentage of 99.8% was substantially higher than that of any other agent. The penalties incurred by our agent are an order of magnitude lower than the penalties incurred by other agents.

Our average factory utilization of 89.4% indicates that the main bottleneck in our agent's performance was often the production capacity of its factory. In a typical game, our agent would receive its first batch of supplies on the fifth day of the game, and was able to produce PCs at nearly 100% utilization for the remainder of the game. Since the expected bottleneck of the TAC SCM game is the amount of available supplies (see Section 4.2), this result seems to indicate that other agents are under-utilizing suppliers' production capacities.

Of the 260 games played, the two most interesting are those in which we competed with `utexas`, since `utexas` seems to be the only agent with a level of sophistication roughly equal to our own. In the first of these two, `utexas` was the winner, with a final score of $8,095,390. Our agent came in second, with a score of $815,358. The four dummy agents performed especially poorly; their scores ranged from -$6,110,901 to -$10,314,096. `utexas` obtained 599 customer orders; we obtained 581 orders. The dummy agents also obtained approximately 600 orders each. These figures are substantially less than the averages presented in Figure 6.3. This seems to indicate that this was a game in which customer RFQs were the main bottleneck. We believe that the reason for our loss in this game was overproduction. Our factory utilization was 81%, while `utexas` was able to fulfill practically the same number of orders with a factory utilization of only 32%. Our agent therefore built up excess finished goods that were never sold to customers. Our supply costs were approximately $5.5 million more than those of `utexas`. This figure accounts for the majority of the difference between our scores. This game demonstrated one of the primary weaknesses of our current implementation: our agent will continue to produce PCs and order supplies even when it is unable to sell its current finished goods inventory at an appropriate rate. When a customer RFQ bottleneck occurs, a better strategy for our agent would be to slow down the rate of production or respond to customer RFQs with a lower offer price. One possible way to slow the rate of production is through the use of a Kanban system. However, the static bidding strategy of our agent does not currently handle this contingency.

In the second game against `utexas` there was no customer RFQ bottleneck. Our agent was therefore able to produce at 87% capacity and fulfill 2095 orders to `utexas`' 1426. This allowed us to achieve nearly $15 million more revenue than `utexas`. However, `utexas` was also assessed over $4 million in penalties. Since our supply costs were roughly identical to those of `utexas`, these two factors led to our victory by a margin of approximately $20 million.

We can calculate the maximum score of a single agent in a 56-day game by using Theorem 2. This value is approximately $129.4 million. The mean final score of our agent is $44.1 million; this is approximately 34.1% of the theoretical maximum. Since our agent is competing in the presence of other agents, we consider this to be a very high percentage. It would be interesting to test the performance of our agent when it is the only player in a game; such a test has not yet been performed.

The mean cost efficiency of our agent in these trials was 2.21, which is also somewhat high, since our agent is making no attempt to obtain supplies at discounted prices. Since it buys nearly all of its supplies at full price, the maximum cost efficiency our agent can possibly obtain is approximately 3.6.

The results of our experimental work seem to indicate that our current implementation is relatively advanced compared to those of most other teams. However, our loss to `utexas` has demonstrated that our agent's strategy is not robust. Several months remain until the TAC SCM finals, so we must continue to improve our agent by devising and implementing more sophisticated strategies.

Chapter 7

Conclusions and Future Work

The primary contribution of this thesis is its extensive analysis of the TAC SCM game. We presented a concise description of this game in Chapter 2. Results from a variety of fields are applicable to this domain; we presented a broad survey of work related to the TAC SCM competition in Chapter 3. We presented a detailed analysis of the TAC SCM game in Chapter 4, including proofs of theoretical results, metrics by which the performance of agents can be measured, and suggestions of strategies that may be of use in the implementation of sophisticated agents. We described the preliminary implementation of a simple supply-driven agent in Chapter 5. Experimental results presented in Chapter 6 seem to indicate that this implementation is currently more sophisticated than those of most other teams. These experiments also illustrated one particular condition in which our agent behaves particularly poorly. We presented a brief comparison of our simple agent's performance to the theoretical results presented in Chapter 4.

Our agent is far from complete. We still need to implement the more sophisticated strategies suggested by the results of Chapter 4 and to experimentally determine the effectiveness of these strategies. In addition, the problems faced by the procurement and sales components have not been fully analyzed. For instance, we are currently unsure how to represent the loss due to being more than five days late in delivery. Although we have enumerated some capabilities that an ideal TAC SCM agent would have, we do not know for certain which of these capabilities are feasible. It seems that machine learning, statistical inference, and search algorithms may be especially useful in the development of the procurement and sales components. Additional theoretical work is needed to determine which of these capabilities are worth further exploration. Experimental work will then be needed to determine the performance of newly-developed strategies.

We intend to perform a *post hoc* analysis of the TAC SCM game after the competition is completed. We can then compare our results to those of other teams and attempt to determine what strategies led to exceptional results.

A potential area for long-term future work is the development of a real-world SCM system based on the principles derived during our analysis and implementation of the TAC SCM agent.

Appendix A

Source Code for the Supplier Capacity Simulation

The source code for the supplier capacity simulation used in Section 4.4 is presented here. This simulation was written in the Java programming language.

```
public class SupplierCapacitySimulation {
  public static void main(String[] args) {
    /* SIMULATION PARAMETERS */
    // Number of trials to run
    int trials = 1000000;
    // Maximum number of PCs that could have been produced in each
    // trial - this is the output of a trial
    long[] maxPCs = new long[trials];

    /* TAC SCM PARAMETERS */
    // Number of days in a TAC SCM game
    int days = 250;
    // Nominal supplier capacity
```

```
int cNominal = 500;
// Total capacity of each of the 8 suppliers
long[] cTotal = new long[8];
// Current capacity of each of the 8 suppliers
int[] cCurrent = new int[8];

System.out.println("Commencing " + trials + " trials...");

for (int i = 0; i < trials; i++) {
  // Initialize total capacity to 0 and current capacity to nominal
  // capacity
  for (int j = 0; j < 8; j++) {
    cTotal[j] = 0;
    cCurrent[j] = cNominal;
  }

  // Simulate the days
  for (int j = 0; j < days; j++) {
    // Update each supplier's current capacity, then add the new
    // current capacity to the supplier's total capacity.
    for (int k = 0; k < 8; k++) {
      double randomValue = Math.random() * 0.1 - 0.05;
      cCurrent[k] = (int)
        Math.max(0,
                 cCurrent[k] +
                 randomValue * cNominal +
                 0.01 * (cNominal - cCurrent[k]));
      cTotal[k] += cCurrent[k];
    }
  }

  // Determine the total number of CPUs, motherboards, memory
  // modules, and hard disks produced in this trial
  long cpus = cTotal[0] + cTotal[1];
  long motherboards = cTotal[2] + cTotal[3];
  long memory = cTotal[4] + cTotal[5];
  long disks = cTotal[6] + cTotal[7];

  // The maximum number of PCs that could be produced is
  // min(cpus, motherboards, memory, disks).
  maxPCs[i] = Math.min(Math.min(cpus, motherboards),
                       Math.min(memory, disks));
}

// Find the results. We are interested in the highest, lowest, and
// mean values of maxPCs.
long highestValue = 0;
long lowestValue = Long.MAX_VALUE;
long totalPCsProduced = 0;
for (int i = 0; i < trials; i++) {
  totalPCsProduced += maxPCs[i];
  if (maxPCs[i] > highestValue) {
    highestValue = maxPCs[i];
  }
```

```
      if (maxPCs[i] < lowestValue) {
        lowestValue = maxPCs[i];
      }
    }
    double meanValue = totalPCsProduced / trials;

    // Now find the standard deviation.
    double variance = 0.0;
    for (int i = 0; i < trials; i++) {
      variance += Math.abs(maxPCs[i] - meanValue);
    }

    // Print it out!
    System.out.println("Minimum PC production was: " + lowestValue);
    System.out.println("Maximum PC production was: " + highestValue);
    System.out.println("Mean PC production was: " + meanValue);
    System.out.println("Standard deviation was: " + Math.sqrt(variance));
  }
}
```

Bibliography

[1] M. Aarup, M. M. Arentoft, Y. Parrod, J. Stader, and I. Stokes. OPTIMUM-AIV: A knowledge-based planning and scheduling system for spacecraft aiv. In M. Fox and M. Zweben, editors, *Knowledge Based Scheduling*. Morgan Kaufmann, 1994.

[2] R. Arunachalam, J. Eriksson, N. Finne, S. Janson, and N. Sadeh. The TAC supply chain management game (Draft version 0.6), March 2003.

[3] B. W. Ballard. The *-minimax search procedure for trees containing chance nodes. *Artificial Intelligence*, 21(3):327–350, 1983.

[4] C. Bell and A. Tate. Using temporal constraints to restrict search in a planner. In *Proceedings of the Third Alvey IKBS SIG Workshop*, 1985.

[5] S. Biswas and Y. Narahari. Object oriented modeling for decision support in supply chain networks. In *Proceedings of POMS-99, International Conference on Operations Management for Global Economy*, December 1999.

[6] A. L. Brudno. Bounds and valuations for shortening the scanning of variations. *Problems of Cybernetics*, 10:225–241, 1963.

[7] S. J. Buckley and J. E. Smith. Supply chain simulation. Technical report, Georgia Tech Logistics Short Course, June 1997.

[8] D. Carmel and S. Markovitch. Learning models of opponent's strategy in game playing. Technical report, Technion-Israel Institute of Technology, 1993.

[9] J. Collins, W. Ketter, and M. Gini. A multiagent negotiation testbed for contracting tasks with temporal and precedence constraints. *International Journal of Electronic Commerce*, 7(1):35–57, 2002.

[10] J. E. Collins. *Solving Combinatorial Auctions with Temporal Constraints in Economic Agents*. PhD thesis, University of Minnesota, June 2002.

[11] M. Collins, R. E. Schapire, and Y. Singer. Logistic regression, adaboost and bregman distances. In *Computational Learing Theory*, pages 158–169, 2000.

[12] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, 1990.

[13] K. Erol, J. Hendler, and D. S. Nau. HTN planning: complexity and expressivity. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*. AAAI Press, 1994.

[14] M. Ettl and M. Schwehm. Determining the optimal network partition and kanban allocation in JIT production lines. In J. Biethahn and V. Nissen, editors, *Evolutionary Algorithms in Management Application*, pages 139–152. Springer-Verlag, 1995.

[15] R. E. Fikes and N. J. Nilsson. STRIPS: a new approach to the application of theorem proving to problem-solving. *Artificial Intelligence*, 2(3-4):189–208, 1971.

[16] J. J. Fuchs, A. Gasquet, B. Olalainty, and K. W. Currie. PlanERS-1: an expert planning system for generating spacecraft mission plans. In *First International Conference on Expert Planning Systems*. Institute of Electrical Engineers, 1990.

[17] D. Fudenberg and J. Tirole. *Game Theory.* MIT Press, 1995.

[18] A. Greenwald and J. Boyan. Bid determination in simultaneous auctions: A case study. In *Proceedings of the Third ACM Conference on Electronic Commerce*, 2001.

[19] M. He and N. R. Jennings. SouthamptonTAC: designing a successful trading agent. In *Proceedings of the Fifteenth European Conference on Artificial Intelligence*, 2002.

[20] M. He and N. R. Jennings. SouthamptonTAC: an adaptive autonomous trading agent. *ACM Transactions on Internet Technology*, To appear, 2003.

[21] M. N. Huhns and L. M. Stephens. Multiagent systems and societies of agents. In G. Weiss, editor, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, chapter 2. MIT Press, 1999.

[22] i2 Technologies. http://www.i2.com, 2003.

[23] S. Jain, C.-C. Lim, B.-P. Gan, and Y.-H. Low. Criticality of detailed modeling in semiconductor supply chain simulation. In *Proceedings of the 1999 Winter Simulation Conference*, 1999.

[24] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa. RoboCup: The robot world cup initiative. In W. L. Johnson and B. Hayes-Roth, editors, *Proceedings of the First International Conference on Autonomous Agents (Agents'97)*, pages 340–347, New York, 1997. ACM Press.

[25] H. Kitano, S. Tadokor, H. Noda, I. Matsubara, T. Takhasi, A. Shinjou, and S. Shimada. Robocup-rescue: Search and rescue for large scale disasters as a domain for multi-agent research. In *Proceedings of the IEEE Conference on Systems, Men, and Cybernetics*, 1999.

[26] D. M. Kreps. *A Course in Microeconomic Theory.* Princeton University Press, 1990.

[27] H. L. Lee and C. Billington. Managing supply chain inventory: pitfalls and opportunities. *Sloan Management Review*, 33(3):65–73, 1992.

[28] H. J. Levesque, P. R. Cohen, and J. Nunes. On acting together. In *Proceedings of the National Conference on Artificial Intelligence.* AAAI Press, 1990.

[29] R. Levinson. General game-playing and reinforcement learning. *Computational Intelligence*, 12(1):155–176, 1996.

[30] S. Mahadevan, N. Marchalleck, T. Das, and A. Gosavi. Self-improving factory simulation using continuous-time average reward reinforcement learning. In *Proceedings of the Fourteenth International Machine Learning Conference*, pages 202–210. Morgan Kaufmann, 1997.

[31] S. Mahadevan and G. Theocharous. Optimizing production manufacturing using reinforcement learning. In *Proceedings of the FLAIRS Conference 1998*, pages 372–377, 1998.

[32] R. A. McCain. Strategy and conflict: An introductory sketch of game theory (course material), 1997.

[33] D. McFadden. Rationality for economists. *Journal of Risk and Uncertainty*, 19:73–105, 1999.

[34] D. Michie. Game-playing and game-learning automata. In L. Fox, editor, *Advances in Programming and Non-Numerical Computation*, pages 183–200. Pergamon, 1966.

[35] T. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[36] J. Nash. Equilibrium points in n-person games. In *Proceedings of the National Academy of Sciences*, volume 36, pages 48–49, 1950.

[37] J. V. Neumann and O. Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, 1944.

[38] V. Pareto. *Manual of Political Economy*. Augustus M. Kelley, 1906.

[39] B. D. Pell. *Strategy Generation and Evaluation for Meta-Game Playing*. PhD thesis, University of Cambridge, August 1993.

[40] M. Pollack and J. F. Horty. There's more to life than making plans. *AI Magazine*, 20(4):71–84, 1999.

[41] A. Postlewaite. Implementation via Nash equilibria in economic environments. In L. Hurwicz, D. Schmeidler, and H. Sonnenschein, editors, *Social Goals and Social Organization: Essays in Memory of Elisha Pazner*, chapter 7, pages 205–228. Cambridge University Press, 1985.

[42] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice-Hall, 1995.

[43] T. Sandholm. Distributed rational decision making. In G. Weiss, editor, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, chapter 5. MIT Press, 1999.

[44] SAP AG. Adaptive supply chain networks. White paper, 2002.

[45] SAP AG. mySAP supply chain management. White paper, 2003.

[46] SAP AG. SAP advanced planner and optimizer. http://www.sap.com, 2003.

[47] R. Schapire, P. Stone, D. McAllester, M. Littman, and J. Csirik. Modeling auction price uncertainty using boosting-based conditional density estimation. In *Proceedings of the Nineteenth International Conference on Machine Learning*, 2002.

[48] C. Shannon. Programming a computer for playing chess. *Philosophical Magazine*, 41(4):256–275, 1950.

[49] S. Shingo. *A Study of the Toyota Production System from an Industrial Engineering Viewpoint.* Productivity Press, 1989.

[50] M. Spearman, D. Woodruff, and W. Hopp. CONWIP: An alternative to kanban. *International Journal of Production Research*, 28(5):879–894, 1990.

[51] P. Stone. *Layered Learning in Multi-Agent Systems.* PhD thesis, Carnegie Mellon University, 1998.

[52] P. Stone. Multiagent competitions and research: Lessons from RoboCup and TAC. In *The RoboCup 2002 International Symposium*, 2002.

[53] P. Stone and A. Greenwald. The first international trading agent competition: autonomous bidding agents. *Electronic Commerce Research*, 2002.

[54] P. Stone, M. L. Littman, S. Singh, and M. Kearns. ATTac-2000: an adaptive autonomous bidding agent. In J. P. Müller, E. Andre, S. Sen, and C. Frasson, editors, *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 238–245, Montreal, Canada, 2001. ACM Press.

[55] P. Stone and M. Veloso. A layered approach to learning client behaviors in the RoboCup soccer server. *Applied Artificial Intelligence*, 12, 1998.

[56] Swedish Institute of Computer Science AB. TAC Classic game description. http://www.sics.se/tac/page.php?id=3, 2003.

[57] The tac-dev Mailing List. tac-dev@sics.se.

[58] M. Tambe. Implementing agent teams in dynamic multi-agent environments. *Applied Artificial Intelligence*, 12, 1998.

[59] S. Tayur, R. Ganeshan, and M. Magazine, editors. *Quantitative Models for Supply Chain Management.* Kluwer Academic Publishers, 1999.

[60] A. Tversky. On the elicitation of preferences: Descriptive and prescriptive considerations. In B. Bell, R. Kenney, and H. Raiffa, editors, *Conflicting Objectives in Decisions.* Wiley, 1977.

[61] M. M. Veloso, M. E. Pollack, and M. T. Cox. Rationale-based monitoring for planning in dynamic environments. In *Artificial Intelligence Planning Systems*, pages 171–180, 1998.

[62] J. M. Vidal and E. H. Durfee. Learning nested agent models in an information economy. *JETAI*, 10(3):291–308, 1998.

[63] W. E. Walsh. *Market Protocols for Decentralized Supply Chain Formation*. PhD thesis, University of Michigan, 2001.

[64] D. Zeng and K. Sycara. Bayesian learning in negotiation. *International Journal of Human Computer Systems*, 48:125–141, 1998.