Abstract of “Select Problems at the Intersection of Computer Science and Economics” by Victor Naroditskiy, Ph.D., Brown University, May 2010.

We apply computer science techniques to try to solve a selection of problems that arise in economics and electronic commerce. The problems we address and our results are summarized below.

The first problem is from the field of Mechanism Design. The goal is to find a procedure for allocating identical items among agents with private values in the manner that maximizes the total utility of the agents. We approach this problem computationally: solutions are found algorithmically rather than through mathematical derivations. Our computational approach yields a nearly optimal solution greatly improving prior results. In the case with 3 agents and 2 items, we were able to find a provably optimal solution.

Next, we address a game-theoretic problem of finding Nash Equilibria in auctions. We investigate when a computational procedure finds an equilibrium in first and second price auctions with discrete bids and values.

The rest of the thesis is devoted to automated decision making in electronic commerce domains. Three domains are considered: sponsored search, supply chain management, and simultaneous auctions. The last two domains are studied in the context of the SCM and Travel divisions of the Trading Agent Competition (TAC).

Our contributions to automated decision making are both practical and theoretical. On the practical side, the bidding strategy we designed for sponsored search auctions is currently being used by a large advertiser. Our work on TAC Travel culminated in winning the competition in 2006. In the TAC SCM competition, the agent we built was among the top 5 out of over 20 agents almost every year of the competition. For theoretical contributions, we characterized optimal strategies for bidding in simultaneous auctions when prices are known and complemented this analysis with an empirical comparison of different strategies. We identified that bidding decisions in TAC SCM can be modeled as a non-linear knapsack problem and proved the asymptotic optimality of a greedy algorithm for solving a class of non-linear knapsack problems.

Select Problems at the Intersection of Computer Science and Economics

by
Victor Naroditskiy
Sc. M., Brown University, 2005

A dissertation submitted in partial fulfillment of the
requirements for the Degree of Doctor of Philosophy
in the Department of Computer Science at Brown University

Providence, Rhode Island
May 2010

This dissertation by Victor Naroditskiy is accepted in its present form by
the Department of Computer Science as satisfying the dissertation requirement
for the degree of Doctor of Philosophy.

Date _____ _____

Amy Greenwald, Advisor

Recommended to the Graduate Council

Date _____ _____

Geoffroy de Clippel, Reader
Department of Economics

Date _____ _____

David Pennock, Reader
Yahoo Research

Date _____ _____

Lyle Ramshaw, Reader
HP Labs

Date _____ _____

Pascal Van Hentenryck, Reader

Approved by the Graduate Council

Date _____ _____

Sheila Bonde
Dean of the Graduate School

Contents

Chapter 1

Introduction

In this thesis, we apply computer science techniques to try to solve a selection of problems that arise in economics and electronic commerce. Roughly speaking, our contributions can be divided into two parts. In the first part, we use a computational approach to find solutions to problems from economic theory. In the second part, we design and implement agents for making complex decisions in various electronic commerce domains. The thesis is organized into chapters most of which are based on previously published joint work with other authors. The title of each chapter has a footnote indicating the authors and, if applicable, the original publication.

The first part of the thesis focuses on applying computer science techniques to problems in economics; specifically, in mechanism design (Chapter 2) and game theory (Chapter 3). Mechanism design and game theory are concerned with problems where multiple self-interested participants take actions to optimize utility based on the *incentives* provided to them. The interplay between incentives and participants' behavior is at the core of the disciplines.

Computer science optimization problems, on the other hand, used to pay no attention to incentives. Most classic computer science problems (e.g., knapsack, traveling salesman) are not concerned with multiple self-interested agents. Instead, they are characterized by a single decision maker seeking to maximize an objective function assuming all factors that affect the objective are not influenced by the decisions being made.

In the last decade, computer scientists have become interested in economics. On the one hand, the concept of incentives has been applied to computer science problems (e.g., shortest path problem on a graph where edges report their weights [82]). On the other hand, computer scientists have searched for computationally-efficient solutions to classic economics problems (e.g., computationally efficient VCG-like combinatorial auctions [29]).

In this thesis, we employ computer science techniques to investigate economics problems in a different way. The problem we study (in Chapter 2) is not a computer science problem but a problem from economic theory, so we are not introducing incentives into a computer science problem. Further, we are not concerned with finding a more computationally efficient solution to this problem as the problem has never before been solved. Instead, we apply computational techniques to solve an open problem in economic theory.

Specifically, in Chapter 2, we focus on designing a procedure for allocating items among participants

in the manner that maximizes the total utility of all participants. This *Allocation Problem* may arise when conference rooms are allocated among employees or a company wants to distribute a limited number of free tickets to its employees. We approach this problem computationally: the solutions are discovered algorithmically rather than through mathematical derivations.

In Chapter 3, we address another problem from economics. There, we computationally study the problem of finding Nash equilibria in auctions. While Nash equilibria are known to exist, they can be described analytically only for a very small class of auctions. We investigate when a computational procedure finds an equilibrium in simple auctions for which no analytical characterization exists.

In the second part of the thesis we use computer science to automate decision making in various electronic commerce domains. The term *agent* will denote a piece of software that makes decisions autonomously. In contrast to the game-theoretic focus of the first two chapters, our focus from now on will concern decision-theoretic problems.

We begin with a problem in the domain of sponsored search. Sponsored search is a way for search engines to monetize search activity and an opportunity for advertisers to provide highly targeted ads. For example, when a person searches for a laptop on Google, the search results page displays relevant links along with a few *sponsored links* (i.e., ads) usually located above and to the right of the (non-sponsored) search results. Leading search engines sell ad space in auctions and provide advertisers with the ability to place bids automatically. Large advertisers spend millions of dollars a year on sponsored search ads. The bidding strategy has a tremendous effect on advertising cost and revenue. In Chapter 4, we design an effective strategy for buying sponsored ads.

Making effective automated decisions in real-world domains requires a lot of learning and, inadvertently, some trial and error. The errors are bound to be expensive if made in live systems. The Trading Agent Competition (TAC) provides a cost-free venue for designing and evaluating automated strategies in certain problem domains. In this thesis, we develop strategies for two TAC divisions. We automate supply chain decisions in the context of the Supply Chain Management (SCM) division and study the problem of bidding in simultaneous auctions in the Travel division.

A supply chain consists of three participants: suppliers, manufacturers, and consumers. Suppliers produce raw materials that are sold to manufacturers who convert raw materials into finished products bought by consumers. Supply chain management is concerned with coordination of raw material purchases and product sales. Supply chain transactions add up to trillions of dollars a year. Historically, procurement agreements have been based on personal contacts among manufacturers and suppliers. A limitation to this way of doing business is that shopping for new contracts is relatively expensive as it requires personal negotiation. The proliferation of information technology we have seen in the last few decades is likely to change the way supply chain transactions are made. In all likelihood, electronic marketplaces for procurement contracts will soon become widespread providing manufacturers and suppliers with an easy and inexpensive way to enter into agreements. A similar change has already taken place on the consumer side with more and more purchases being made online rather than in bricks-and-mortar stores.

An electronic marketplace offers participants more options for buying and selling raw materials. Taking

full advantage of the extra flexibility requires manufacturers to choose from a large number of possible procurement contracts. Each desired raw material can be procured in various quantities and with various lead times. In the presence of multiple suppliers providing the same raw material and interdependencies among raw materials, choosing the best procurement contracts is a daunting task. The task is complicated even further as the appropriate parameters of a procurement contract depend on demand for products as well as future prices, both of which are uncertain. Computer science can provide algorithms that greatly improve human decision making in the procurement domain.

Developing strategies for participating in an electronic marketplace is an interesting and important problem. However, because it is too expensive to experiment with real contracts, automated agent development has to be carried out in a simulated environment. Fortunately, TAC SCM provides such an environment for the supply chain domain. TAC SCM participants design strategies for computer manufacturers that buy raw materials from suppliers and sell assembled computers to customers. The effectiveness of a strategy is evaluated based on its performance relative to strategies of the other participants. Chapters 5, 6, and 7 discuss solutions to some of the problems that arise when automating supply chain decisions. Chapter 5 addresses the problem of choosing which products to manufacture (schedule for production) given limited factory capacity and uncertain demand for products. Chapters 6 and 7 tackle a more difficult problem where sales decisions are made in combination with scheduling decisions.

Another domain where we automate decision making is bidding in simultaneous auctions. The problem of buying complementary and substitutable items from different sellers is common in the real world (eBay is the most notable example). To illustrate the problem, consider a buyer interested in buying a cell phone and a head set. The items are complementary; i.e., the buyer values the cell phone/head set package more than the sum of the values of each item alone. Not being allowed to submit a single bid for the package, the buyer needs to decide how to split the total value of the package into two separate bids. If the buyer bids too high on both items, she risks spending more than the value of the package. If she places bids that are not high enough, she risks winning only one of the items for the price above the value from having that item alone. Strategies for bidding in simultaneous auctions are discussed in Chapter 8.

We continue investigation of bidding strategies in the Travel division of TAC. TAC Travel provides a simulated market environment consisting of simultaneous and sequential auctions selling complementary and substitutable items. We describe the details of the competition and the strategy of our winning agent in Chapter 9.

A fundamental feature of most electronic commerce problems is that decisions have to be made without having accurate information about the future. Consider the case of bidding in multiple auctions. Bids must be submitted before closing prices of auctions are known. Indeed, closing prices cannot be known beforehand as they are determined by the bids. However, optimal bidding decisions depend on the bids submitted by the other agents making it necessary for a successful agent to predict those bids as accurately as possible. The predictions are inherently imperfect as the other agents do not make their bids known before submitting them and their bids also depend on their predictions of the bids of the competitors. Robustness to imperfect predictions is a crucial feature of a successful agent. Chapter 10 is devoted to an investigation of the robustness of various strategies to imperfect prediction.

Our contributions to automated decision making are both practical and theoretical. On the practical side, the bidding strategy we designed for sponsored search auctions is currently being used by a large advertiser. Our work on TAC Travel culminated in winning the competition in 2006 (Chapter 9). Theoretically, we characterized optimal strategies for bidding in simultaneous auctions when prices are known (Chapter 8) and complemented that analysis with an empirical comparison of different strategies (Chapter 10). We proved the asymptotic optimality of a greedy algorithm for solving a class of non-linear knapsack problems (Chapter 7).

In the realm of economics, we demonstrated that computational techniques can be used synergistically with theoretical economics. Our results on a computational search for Nash equilibria in auctions (Chapter 3) are very preliminary, but point to potential avenues for further investigation. Our computational approach to solving the Allocation Problem (Chapter 2) yielded solutions that greatly improved prior results. For a certain case, we found a provably optimal solution. We plan to apply the algorithms used to solve the Allocation Problem to other mechanism design problems with similar incentive structures. More generally, we believe that a computational approach is a fruitful direction for solving a variety of economics problems.

Chapter 2

The Allocation Problem: Destroy to Save[1]

We study the problem of how to allocate m identical items among $n > m$ agents, assuming each agent desires exactly one item and has a private value for consuming it. We assume the items are jointly owned by the agents, not by one uninformed center, so an auction cannot be used to solve our problem. Instead, the agents who receive items compensate those who do not.

This problem has been studied by others recently, and their solutions have modified the classic VCG mechanism. This approach guarantees strategy-proofness and allocative efficiency. Further, in an auction setting, VCG guarantees budget balance, because payments are absorbed by the auctioneer. In our setting, however, where payments are redistributed to the agents, some money must be burned in order to retain strategy-proofness.

While strategy-proofness is necessary for truthful implementation, allocative efficiency (allocating the m items to those that desire them most), is not always an appropriate goal in our setting. Rather, we contend that maximizing social surplus is. In service of this goal, we study a class of mechanisms that may burn not only money but destroy items as well. Our key finding is that destroying items can save money, and hence lead to greater social surplus.

More specifically, our first observation is that a mechanism is strategy-proof iff it admits a threshold representation. Given this observation, we restrict attention to specific threshold and payment functions for which we can numerically solve for an optimal mechanism. Whereas the worst-case ratio of the realized social surplus to the maximum possible is close to 1 when $m = 1$ and 0 when $m = n - 1$ under the VCG mechanism, the best mechanism we find coincides with VCG when $m = 1$ but has a ratio approaching 1 when $m = n - 1$ as n increases.

[1] Based on [28].

2.1 Introduction

Suppose six roommates jointly own a car that seats five people. They decide to take a weekend trip to the countryside. While they all would like to go, there is not room for all of them. Some really need the fresh country air while others would not mind staying home. The roommates don't necessarily know one another's desires, but each of them knows her own true value of getting out of the city. How should they decide who gets to go?

More generally, we study a class of resource allocation problems, in which n agents commonly own $m < n$ identical items that they wish to distribute among themselves, assuming each agent wants exactly one item, and has a *value* for that item which is known to her alone. As further examples, one could think of the allocation of free tickets for a sport event among club members, or seats on an overbooked plane. This kind of problem is often discussed in the literature on social choice where the goal is fairness, often under the assumption that the agents' values are commonly known. Yet this assumption is rarely satisfied, and self-interested agents will misreport their private values if doing so would be profitable. This is why our primary focus is incentives, instead of fairness.

Significant progress has been made in the field of mechanism design on the general topic of incentives since the seventies. Most research efforts have been devoted to understanding what is achievable in the presence of informational constraints (e.g., revelation principles in mechanism design). Professors Hurwicz, Maskin and Myerson received the 2007 Sveriges Riksbank Prize in Economic Sciences in Memory of Alfred Nobel for their groundbreaking contributions in this direction. Much less is known, however, about how to select a mechanism that is socially optimal, among those that are incentive compatible. In other words, the extension of social choice theory to problems characterized by asymmetric information remains an important avenue of further exploration.

In this work we will focus on *strategy-proof* mechanisms, which require that it is a dominant strategy for each agent to report her value truthfully. This requirement is less permissive, but more robust than Bayesian implementation. In particular, agents are more likely to play a dominant strategy than a strategy that is optimal only when other agents play their part of the truthful equilibrium. Perhaps even more importantly, dominant-strategy implementation does not require any assumptions about the distribution of the agents' values (or their beliefs), nor their attitude towards risk.

As for measuring the appeal of various strategy-proof mechanisms, we will apply a worst-case measure. More specifically, if one fixes the agents' profile of values, one can compute the ratio of the total social surplus realized by the mechanism over the maximal total social surplus that could be achieved, should these values be publicly known. Since these values are not known, nor is their probabilistic distribution, the appeal of a strategy-proof mechanism will be measured by the minimum of this ratio over all possible value profiles. We call this ratio a *social surplus index*, and we use it to determine a mechanism's worst-case (i.e., guaranteed) level of social surplus: reaching a level $\alpha \in [0, 1]$ means that a mechanism realizes at least a proportion α of the maximal social surplus for *every* possible profile of the agents' values.

We assume that the agents can make monetary payments, and further that they have *quasi-linear* utilities. Our problem is different from those studied in auction theory. There, any monetary payments go to the auctioneer, who is usually assumed to have no private information. The presence of such "residual claimants"

makes it easy to achieve budget balance. In our problem, however, the objective is to redistribute as much as possible of the payments among the agents themselves without negatively impacting the incentives (agents may have an incentive to misreport their values to receive higher compensation).

In addition to strategy-proofness, we also impose the following natural constraints 1) *feasibility*: no more than m items can be allocated, and monetary deficits are not allowed (i.e., no external subsidy), 2) *individual rationality*: each agent's total utility is nonnegative, and 3) *anonymity*: the allocation and payment decision applied to each agent does not depend on her identity. The question we are interested in can now be stated formally:

> *Find a mechanism that maximizes the worst-case social surplus index among all those that are strategy-proof, feasible, individually rational, and anonymous.*

Recently, two sets of authors (Moulin [78] and Guo/Conitzer [49]) solved the above question under the additional assumption that the items be allocated to the m agents who value them most, at each possible profile of values. Both their solutions (derived independently) involved a class of mechanisms called VCG[2] mechanisms, which has received special attention in the economic literature because they admit a simple functional form (cf. Green and Laffont's [41] characterization). Using a VCG mechanism guarantees an efficient allocation of the m items available, but not necessarily a good level of overall efficiency (as measured for instance by the worst-case social surplus index), because allocative efficiency may come at the cost of "burning" quite a bit of money to meet the incentive constraints (when $m \geq 2$). So it may be better, in terms of overall efficiency, to destroy some items in order to save money. Indeed it is. It is not difficult to check that it is impossible to guarantee a strictly positive ratio using a VCG mechanism when $m = n - 1$. On the other hand, applying the best VCG mechanism after destroying one item would secure a strictly positive ratio.

Still, applying a VCG mechanism after destroying some fixed number of items is not the best strategy for optimizing overall efficiency. As a first step towards solving the general question, we offer a characterization of all strategy-proof mechanisms in terms of *threshold mechanisms*: i.e., an agent receives an item if and only if her reported value is larger than a threshold value *that may depend on other agents' reports*.[3] Although we don't believe that this result has been stated explicitly in papers discussing the very same model as ours, it is reminiscent of previous characterizations of VCG mechanisms (see, e.g., Green and Laffont [41]) and other strategy-proof mechanisms in related models ([40]).

Though helpful in understanding the question, this characterization result does not immediately allow us to solve it, because the feasibility and individual rationality constraints are nontrivial. The allocation function of the VCG mechanism is constant once the agents' values are ordered decreasingly (the agents with the m highest values receive the items). In that context Guo and Conitzer [49] and Moulin [78] manage to find the optimal payment function. The allocation function of the more general mechanisms that we study is not constant making the search for the optimal mechanism more difficult. We haven't found yet a way of solving the general question. But we have managed to identify specific classes of threshold and compensation

[2] VCG stands for Vickrey, Clarke, and Groves, who independently defined and studied some of these mechanisms in various contexts.

[3] This is a key distinction between our work and [50], where destroying the same number of items regardless of reported values was considered.

functions that allow us to partition the set of value profiles into regions for which the resulting constrained optimization problem is linear in values. We then solve this LP problem numerically.

Our approach is designed to achieve the right balance between tractability, and showing that one can obtain a significant improvement of overall efficiency if one does not rely on the technical convenience of VCG mechanisms. Perhaps most striking is the case where $m = n - 1$. As already pointed out, VCG mechanisms cannot guarantee any strictly positive ratio in this case. Further, applying the best VCG mechanism after destroying a fixed number of items does not guarantee a ratio larger than $1/2$ (see numerical computations in Guo and Conitzer [50]). Our method of "contingent destruction" will identify a mechanism that guarantees a ratio $1 - \frac{2}{n^2-n}$, which rapidly approaches 1 as n increases.

We conclude this introduction by discussing some related literature. Enhancing VCG mechanisms with payment redistribution has been studied in various settings. Bailey [8] proposes a way to redistribute some of the VCG tax in a public good domain. Cavallo [22] designs a redistribution mechanism for single-item allocation problems, and provides a characterization of redistribution mechanisms for more general allocation problems. As already mentioned, Guo and Conitzer [49] and Moulin [78] independently discover the optimal VCG redistribution mechanism for the allocation domain studied here. In [51], Guo and Conitzer derive a linear redistribution VCG mechanism to maximize the expected social surplus when the distribution of agents' values is known. Porter et al. [91] study the problem of allocating undesirable goods (e.g., tasks) to agents in a fair manner.

Most related to our work is the work of Guo and Conitzer [50]. Starting from the same observation as ours that applying a VCG mechanism after destroying a fixed number of items may increase the worst-case social surplus index, they study mechanisms where the number of items destroyed may be a random variable. Introducing lotteries implies that one must take into account the agents' attitude towards risk. Guo and Conitzer's analysis requires the agents to be risk neutral. Also, the feasibility and individual rationality constraints hold only in expectation. Perhaps most importantly, the lottery that determines how many items to destroy does not vary with the players' reports. The key insight we offer in the present work is that one can improve upon the optimal VCG redistribution mechanism without using lotteries, if one applies *contingent destruction* rules. If one is willing to use lotteries, then it may be of interest to combine the insights from our two papers, making Guo and Conitzer's random variables vary with reported values.

Other directions have also been followed when allowing for lotteries. Faltings [34], for instance, proposes a mechanism that picks an agent at random, and makes him the recipient of the VCG payments. The mechanism, which applies to domains more general than our allocation domain, achieves budget balance. However, if one applies this mechanism to our allocation domain, one sees that the resulting allocation is not efficient (unless the chosen recipient happens to value the item less than those who are allocated an item).

This chapter unfolds as follows. Section 2.2 formally states the problem we are studying. We characterize strategy-proof mechanisms for the allocation domain in Section 2.3. A computational method of searching for an optimal mechanism in a restricted setting is proposed in Section 2.4. Numerical results in this setting are presented in Section 2.5.

2.2 Definitions

An *allocation problem* is a triple $\langle n, m, v\rangle$, where n is the number of agents, $m < n$ is the number of (identical) items available to allocate, and $v \in \mathbb{R}^n_+$ represents the agents' satisfaction from consuming one item (agents do not care for consuming multiple units). We restrict attention to value profiles v such that $v_1 \geq v_2 \geq \ldots \geq v_n \geq 0$. This is without loss of generality since our problem involves only anonymous mechanisms. Monetary compensations are possible, and utilities are quasi-linear. The space of possible values is then $V = \{v \in \mathbb{R}^n_+ \mid v_1 \geq v_2 \geq \ldots \geq v_n \geq 0\}$. An *allocation* is a pair $(a, t) \in \{0, 1\}^n \times \mathbb{R}^n$, where $a_i = 1$ if and only if agent i gets one item, and t_i represents the amount of money that agent i receives (this number can be negative, of course, in which case agent i pays that amount). Hence, the total utility of agent i when implementing the allocation (a, t) is $a_i v_i + t_i$, if her value for the item is v_i. A *mechanism* is a pair of functions $f : \mathbb{R}^n_+ \to \{0, 1\}^n$ and $t : \mathbb{R}^n_+ \to \mathbb{R}^n$. Thus, it determines an allocation for each possible report from the agents regarding their value for the item. The vector $v_{-i} \in \mathbb{R}^{n-1}$ denotes values of the agents other than agent i and the vector v can be written as (v_i, v_{-i}). We focus on mechanisms that satisfy the following constraints:

- *Feasibility*: no more than m items should be allocated, and the sum of payments to the agents should be less than or equal to zero, for all value vectors v. In other words,

$$\sum_{i=1}^{n} f_i(v) \leq m \text{ and } \sum_{i=1}^{n} t_i(v) \leq 0 \quad \forall v \in V$$

- *Strategy-proofness*: It is a dominant strategy for each agent to report her value truthfully. Formally, $\forall v \in V, i, v'_i$

$$f_i(v_i, v_{-i})v_i + t_i(v_i, v_{-i}) \geq f_i(v'_i, v_{-i})v_i + t_i(v'_i, v_{-i})$$

- *Individual Rationality*: It is in each agent's interest to participate in the mechanism, for all value vectors v, i.e.

$$f_i(v)v_i + t_i(v) \geq 0 \quad \forall v \in V, i$$

We now define the index that we will use to measure the overall efficiency of a mechanism (f, t) that is implemented truthfully (an equivalent index was used in [78, 49, 50]). If the true value vector is v, then the (utilitarian) surplus realized by the mechanism is equal to $\sum_{i=1}^{n}[v_i f_i(v) + t_i(v)]$. This absolute number is less interesting than knowing how far it is from the first-best solution, i.e. the maximal surplus one could achieve if the agents' values were known. In order to have an index that is unit-free (i.e. homogenous of degree zero), it is natural to consider a ratio. Finally, since the agents' values are not known, nor their probabilistic distribution, it is natural to consider the worst-case index. To summarize, the index that we will use to measure the performance of a mechanism (f, t) that is truthfully implemented is given by the following number:

$$\min_{v \in V} \frac{\sum_{i=1}^{n}[f_i(v)v_i + t_i(v)]}{\max_{a \in \mathcal{F}(m)} \sum_{i=1}^{n} a_i v_i},$$

where $\mathcal{F}(m) = \{a \in \{0,1\}^n | \sum_{i=1}^n a_i \leq m\}$. Finding a mechanism whose index is α means that a proportion α of the maximal total surplus is achieved, *independently* of what the true values are. Following the convention $v_1 \geq v_2 \geq \ldots \geq v_n \geq 0$, the denominator becomes $\sum_{i=1}^m v_i$ and we write the index as

$$\min_{v \in V} \frac{\sum_{i=1}^n [f_i(v) v_i + t_i(v)]}{\sum_{i=1}^m v_i}$$

The formal content of the question stated in the Introduction can thus be summarized by the following optimization problem:

$$\max_{(f,t)} \quad \min_{v \in V} \frac{\sum_{i=1}^n [f_i(v) v_i + t_i(v)]}{\sum_{i=1}^m v_i} \tag{2.1}$$

$$\sum_{i=1}^n f_i(v) \leq m \quad \forall v \in V \tag{2.2}$$

$$\sum_{i=1}^n t_i(v) \leq 0 \quad \forall v \in V \tag{2.3}$$

$$f_i(v_i, v_{-i}) v_i + t_i(v_i, v_{-i}) \geq f_i(v_i', v_{-i}) v_i + t_i(v_i', v_{-i}) \quad \forall v \in V, i, v_i' \tag{2.4}$$

$$f_i(v) v_i + t_i(v) \geq 0 \quad \forall v \in V, i \tag{2.5}$$

2.3 A Characterization of Strategy-Proofness: the Threshold Mechanisms

The allocation domain places strong restrictions on value functions of the agents. Specifically, an agent's value is zero in all outcomes where the agent is not allocated an item and the private value $v_i > 0$ in all outcomes where the agent is allocated an item. We use this restriction on the values to characterize strategy-proof mechanisms in the following proposition.[4]

Proposition 1 *An allocation mechanism (f,t) is strategy-proof if and only if it is a "threshold mechanism," meaning that, for each $i = 1, \ldots, n$, there exist a threshold function $\tau_i : \mathbb{R}_+^{n-1} \to \mathbb{R} \cup \{\infty\}$ and a compensation function $c_i : \mathbb{R}_+^{n-1} \to \mathbb{R}$ such that*

$$\begin{cases} f_i(v) = 1 \text{ and } t_i(v) = c_i(v_{-i}) - \tau_i(v_{-i}) & \text{if} \quad v_i \geq \tau_i(v_{-i}) \\ f_i(v) = 0 \text{ and } t_i(v) = c_i(v_{-i}) & \text{if} \quad v_i < \tau_i(v_{-i}), \end{cases}$$

or

$$\begin{cases} f_i(v) = 1 \text{ and } t_i(v) = c_i(v_{-i}) - \tau_i(v_{-i}) & \text{if} \quad v_i > \tau_i(v_{-i}) \\ f_i(v) = 0 \text{ and } t_i(v) = c_i(v_{-i}) & \text{if} \quad v_i \leq \tau_i(v_{-i}). \end{cases}$$

[4] We thank Yves Sprumont for pointing out this simple result to us.

Remark The threshold mechanisms are easy to interpret. Each agent faces a personalized price (the threshold) that is determined by the reports of the other agents. She gets the good if and only if her reported value is (strictly) larger than this price, and must pay it in exchange. The collected money can be redistributed to some extent to the agents via the compensation function. The VCG mechanisms form a special case, where i's threshold is the m^{th} largest component of v_{-i}.

Proof The sufficient condition is straightforward to check. So we provide an argument only for the necessary part. Fix i and the reports v_{-i} from the other agents. Strategy-proofness implies that

$$f_i(v) = f_i(v'_i, v_{-i}) \Rightarrow t_i(v) = t_i(v'_i, v_{-i}), \tag{2.6}$$

for all v_i, v'_i. It is easy to write (f, t) as a threshold mechanism if $f_i(v) = 0$, for all v_i, or $f_i(v) = 1$, for all v_i. Suppose thus that there exists v_i, v'_i such that $f_i(v) = 1$ and $f_i(v'_i, v_{-i}) = 0$. For any such pair, strategy-proofness implies that $v_i + t_i(v) \geq t_i(v'_i, v_{-i}) \geq v'_i + t_i(v)$. Hence $v_i \geq v'_i$. The space of agent's values $v_i \in \mathbb{R}_+$ can be partitioned in two intervals based on the mapping to either $f_i(v_i, v_{-i}) = 0$ or $f_i(v_i, v_{-i}) = 1$: there exists a threshold, denoted $\tau_i(v_{-i})$, such that $f_i(v) = 1$ if and only if $v_i \geq \tau(v_{-i})$ (or $f_i(v) = 1$ if and only if $v_i > \tau(v_{-i})$). Given (2.6), let $t_i^1(v_{-i})$ (resp. $t_i^0(v_{-i})$) be the payment made by i when she receives (resp. does not receive) the item. Strategy-proofness implies that $\tau(v_{-i}) + \epsilon + t_i^1(v_{-i}) \geq t_i^0(v_{-i}) \geq \tau(v_{-i}) - \epsilon + t_i^1(v_{-i})$, for each $\epsilon > 0$. Making ϵ tend to zero, we conclude that $t_i^1(v_{-i}) = t_i^0(v_{-i}) - \tau(v_{-i})$, and the result follows by taking $c = t_i^0$. ∎

If we add anonymity to strategy-proofness in Proposition 1, the mechanism will change only in dropping indexes i from τ and c. For notational convenience from now on we will restrict our attention to generic profiles v where all components are distinct. This restriction is introduced without loss of generality as we can extend the mechanism to all value profiles (including vectors with equal components) by using uniform lotteries[5] to break ties, as is usually done in papers on auctions.

This characterization of strategy-proofness is reminiscent of other well-known results for VCG and other more general strategy-proof mechanisms (see [77, 73]). A result similar to Proposition 1 appears in [40] in a slightly different context.

We restrict our attention to the first class of mechanisms identified in Proposition 1 (the one with $v_i \geq \tau_i(v_{-i})$) and restate the constrained optimization problem (2.1)-(2.4) using the threshold characterization:

The first constraint is the feasibility constraint with respect to the items being allocated, while the second constraint is the feasibility constraint with respect to money (the sum of all compensations or rebates should be no more than the sum of the money collected from the agents that get an item). The third constraint is the individual rationality constraint (remember that an agent's value must be larger than the threshold when she gets an item, and so the IR constraint is trivially satisfied for her as well).

[5] Suppose for instance that agent i should receive an item, and that more than m other agents have the same value as i. Anonymity would then come in conflict with feasibility. A uniform lottery will then be used to determine which subset of agents will receive the item, among all those that have the same value. Even so, the way agents react to risk is irrelevant because all the outcomes of the lottery are equivalent in terms of utility. Specifically, the lottery is between receiving the item worth v_i at the price p_i and receiving compensation c_i such that $c_i = v_i - p_i$.

$$\max_{(c,\tau)} \quad \min_{v \in V} \frac{\sum_{i|v_i \geq \tau(v_{-i})}(v_i - \tau(v_{-i})) + \sum_{i=1}^{n} c(v_{-i})}{\sum_{i=1}^{m} v_i}$$
$$\#\{i|v_i \geq \tau(v_{-i})\} \leq m \quad \forall v \in V$$
$$\sum_{i=1}^{n} c(v_{-i}) \leq \sum_{i|v_i \geq \tau(v_{-i})} \tau(v_{-i}) \quad \forall v \in V$$
$$c(v_{-i}) \geq 0 \quad \forall v \in V, i$$

We now propose a last formulation of our optimization problem. We remove the minimization over v by introducing a variable $r \in \mathbb{R}$ denoting the best ratio that holds for any profile of values. The resulting optimization program is:

$$\max_{r,c,\tau} r \tag{2.7}$$
$$\sum_{i|v_i \geq \tau(v_{-i})} (v_i - \tau(v_{-i})) + \sum_{i=1}^{n} c(v_{-i}) \geq r \sum_{i=1}^{m} v_i \quad \forall v \in V \tag{2.8}$$
$$\#\{i|v_i \geq \tau(v_{-i})\} \leq m \quad \forall v \in V \tag{2.9}$$
$$\sum_{i=1}^{n} c(v_{-i}) \leq \sum_{i|v_i \geq \tau(v_{-i})} \tau(v_{-i}) \quad \forall v \in V \tag{2.10}$$
$$c(v_{-i}) \geq 0 \quad \forall v \in V, i \tag{2.11}$$

Figure 2.1: The problem of finding an optimal mechanism stated as an optimization problem.

A solution to the mathematical program above is the optimal mechanism for the allocation domain. However the program is hard to solve for different reasons. Firstly, maximization is over arbitrary functions c and τ, and there is little hope in optimizing over the space of arbitrary functions. Secondly, the program has an infinite number of constraints as the set of possible value vectors $v \in V$ is infinite. We address these problems in the next section where we make assumptions about the form of the functions c and τ, show that it is sufficient to consider a finite number of constraints, and solve the resulting problem computationally.

2.4 A Simpler Problem

There are infinitely many constraints in our optimization problem, since they are indexed by the value profiles v. Our main insight for dealing with this difficulty comes from the simple observation that linear constraints are satisfied by all the elements of a convex polytope if and only if they are satisfied by its extreme points. So we will restrict attention to the threshold and compensation functions that add some linearity to the general problem. The main difference with [78, 49] is that the number of items allocated, and thereby the constraints in the optimization problem, will vary with v when considering non-VCG mechanisms. So we will have to decompose the

For the allocation problem with m items and n agents

1. Choose k and p to select a threshold function satisfying Assumption 1
2. For each such threshold function τ_{kp}
   (a) find the optimal compensation function c_{kp} that satisfies Assumption 2
3. Choose the mechanism (τ_{kp}, c_{kp}) that obtains the highest percentage of the optimal utility

Figure 2.2: Algorithm for finding an approximate solution to the mechanism design problem

- $i \in \{(p+1) \dots m\}$

  if $v_i \geq kv_p$**:** $f_i = 1$

  $$t_i = -\max(kv_p, v_{m+1}) + \begin{cases} av_{-i} & \text{if } kv_p \geq v_{m+1} \\ bv_{-i} & \text{otherwise} \end{cases}$$

  otherwise: $f_i = 0,\ t_i = \begin{cases} av_{-i} & \text{if } kv_p \geq v_{m+1} \\ bv_{-i} & \text{otherwise} \end{cases}$

- $i \in \{(m+1) \dots n\}: \quad f_i = 0,\ t_i = \begin{cases} av_{-i} & \text{if } kv_p \geq v_m \\ bv_{-i} & \text{otherwise} \end{cases}$

By the definition of the threshold function $\tau = \max(kv^p_{-i}, v^m_{-i})$, there are $m - p + 1$ possible allocations (the first p agents get the items, the first $p+1$ agents get the items, ..., the first m agents get the items) determined by the position of kv_p among $v_p \dots v_m$. The compensation function c is resolved to one of the two linear functions (av_{-i} or bv_{-i}) when the position of kv_p relative to v_m and v_{m+1} and the position of kv_{p+1} relative to v_{m+1} are determined. Each region below is defined to have a constant number of allocated items and a linear compensation function (i.e., resolved to either av_{-i} or bv_{-i}).

$$\begin{aligned}
&\forall j \in \{p \dots m\},\ j' \in \{\max(p+1, j) \dots m\} \\
&\quad V_{j,j'} = \{v \in V | v_1 \geq \dots \geq v_p \geq \dots \geq v_j \geq kv_p \geq v_{j+1} \geq \\
&\quad \dots \geq v_{j'} \geq kv_{p+1} \geq v_{j'+1} \geq \dots \geq v_m \geq \dots \geq v_n\} \\
&\forall j \in \{p \dots m\} \\
&\quad V_{j,m+1} = \{v \in V | v_1 \geq \dots \geq v_p \geq \dots \geq v_j \geq kv_p \geq v_{j+1} \geq \\
&\quad \dots \geq v_m \geq \dots \geq v_n \text{ AND } v_{m+1} \geq kv_{p+1}\} \\
&V_{m+1,m+1} = \\
&\{v \in V | v_1 \geq \dots \geq v_n \text{ AND } v_{m+1} \geq kv_p \text{ AND } v_{m+1} \geq kv_{p+1}\}
\end{aligned}$$

The collection of regions above partitions the space $\{v \in V |\ v_1 \geq v_2 \geq \dots \geq v_n \geq 0\}$. We group constraints by the regions and state the optimization problem in Figure 2.3. Notice that on each region the constraints are

of the form $dv \geq 0$ for some $d \in \mathbb{R}^n$, which means that they are satisfied at λv $(\forall \lambda > 0)$ as soon as they are satisfied at v. Hence we can assume without loss of generality that $v_1 = 1$ and focus on polytopes of vectors $(v_2, \ldots, v_n) \in \mathbb{R}^{n-1}$ characterized by $(n+2)$ inequalities:

$$
\begin{aligned}
& 1 \geq v_2 \\
& v_2 \geq v_3 \\
& \ldots \\
& v_j \geq k v_p \\
& k v_p \geq v_{j+1} \\
& \ldots \\
& v_{j'} \geq k v_{p+1} \\
& k v_{p+1} \geq v_{j'+1} \\
& \ldots \\
& v_{n-1} \geq v_n \\
& v_n \geq 0
\end{aligned}
$$

Extreme points of these polytopes have the property that $n-1$ of these inequalities are binding. It is easy to check that this is possible only is the variables $v_2, \ldots, v_n$ take the values $1, k, k^2, 0$. Therefore all of the extreme points are of the form $(1, \ldots, 1, k, \ldots, k, k^2 \ldots, k^2, 0 \ldots, 0)$. Making sure the constraints hold on all such vectors guarantees that the constraints hold everywhere on $V_{j,j'}$. Now the linear program in Figure 2.3 can be stated with a finite number of constraints.

Example As an example consider the allocation problem with $n = 3$, $m = 2$ and the threshold function with $k = .5, p = 1$: $\tau = \max(.5v^1_{-i}, v^2_{-i})$. The threshold function for agent 1 is $\max(.5v_2, v_3) < v_1$. So agent 1 is always allocated an item. The threshold for agent 2 is $\max(.5v_1, v_3)$. Agent 2 is allocated an item only when $v_2 > .5v_1$. Agent 3 is never allocated an item as the threshold for agent 3 is $\max(.5v_1, v_2) > v_3$.

The compensation function is linear when in addition to the allocation the position of $.5v_1$ and $.5v_2$ relative to v_3 is determined. Taking $v_1 = 1$ we can represent this on a 2-dimensional graph (Figure 2.4). The space is divided into 5 regions, with each region having a linear compensation function and a fixed allocation. To make sure the constraints hold for all $\{v \in V |\ v_1 \geq v_2 \geq v_3\}$, we just need to enforce each region's constraints on its extreme points. For example, the extreme points of the right bottom region after adding $v_1 = 1$ as the first component are (1,.5,0), (1,.5,.25), (1,1,.5), (1,1,0).

2.5 Results

We find mechanisms for different values of n and m using the computational approach described in Figure 2.2. The class of threshold functions we consider is given by all pairs (k, p) where k takes values in $\{0, .025, .05, \ldots, .975\}$ and p in $\{1, 2, \ldots, m-1\}$. We used CPLEX 11.2.0 as a linear program solver.

$$\max_{a,b\in\mathbb{R}^{n-1},r\in\mathbb{R}} r$$

$\forall j \in \{p \dots m\},\ j' \in \{\max(p+1, j) \dots m\},\ v \in V_{j,j'}$

$$\sum_{i=1}^{j} v_i - \sum_{i=1}^{p} k v_{p+1} - \sum_{i=p+1}^{j} k v_p + \sum_{i=1}^{n} c(v_{-i}; a, b) \geq r \sum_{i=1}^{m} v_i$$

$$\sum_{i=1}^{n} c(v_{-i}; a, b) \leq \sum_{i=1}^{p} k v_{p+1} + \sum_{i=p+1}^{j} k v_p$$

$$c(v_{-i}; a, b) \geq 0 \quad \forall i$$

$\forall j \in \{p \dots m\},\ \forall v \in V_{j,m+1}$

$$\sum_{i=1}^{j} v_i - \sum_{i=1}^{p} v_{m+1} - \sum_{i=p+1}^{j} k v_p + \sum_{i=1}^{n} c(v_{-i}; a, b) \geq r \sum_{i=1}^{m} v_i$$

$$\sum_{i=1}^{n} c(v_{-i}; a, b) \leq \sum_{i=1}^{p} v_{m+1} + \sum_{i=p+1}^{j} k v_p$$

$$c(v_{-i}; a, b) \geq 0 \quad \forall i$$

$\forall v \in V_{m+1,m+1}$

$$\sum_{i=1}^{m} v_i - \sum_{i=1}^{m} v_{m+1} + \sum_{i=1}^{n} c(v_{-i}; a, b) \geq r \sum_{i=1}^{m} v_i$$

$$\sum_{i=1}^{n} c(v_{-i}; a, b) \leq \sum_{i=1}^{m} v_{m+1}$$

$$c(v_{-i}; a, b) \geq 0 \quad \forall i$$

Figure 2.3: Linear program with constraints grouped by regions $V_{j,j'}$.

Figure 2.5 illustrates the results we generate for each setting of n, m, and p. The value for the parameter k is varied along the horizontal axis. For each value of k, the corresponding threshold function is $\tau = \max(kv^p_{-i}, v^m_{-i})$, and we can solve the linear program in Figure 2.3 to find an optimal compensation function c_{kp}. The ratio for each mechanism (τ_{kp}, c_{kp}) is plotted for the corresponding k value. We refer to the resulting graph as the *performance curve*. We scan the values of k for the one that has the highest ratio. In Figure 2.5, the best ratio is for $k = .20$. Notice that the shape of the curve suggests that there is only one peak. We try other values of k around $.175$ to find the peak at $k = \frac{1}{6}$. In all of our results we noticed that the performance curve as a function of k is single-peaked.

The threshold function with $k = 0$ corresponds to the efficient allocation function and the mechanism we find for $k = 0$ is the best VCG mechanism. The ratio of the best VCG mechanism appears at $k = 0$ and as argued before is zero when $n = m + 1$.

For any fixed values of n and m we found that a mechanism with p set to $m-1$ achieves the highest ratio. This setting of p means that at most one item is destroyed. This result is consistent with the one obtained by Guo and Conitzer [50] for randomized VCG mechanisms. They find that the best mechanism randomizes between destroying one item and not destroying any items. The performance curves for different values of p are shown in Figure 2.6. Notice that the highest ratio is obtained on the graph for $p = m - 1 = 8$ ($k = .1$).

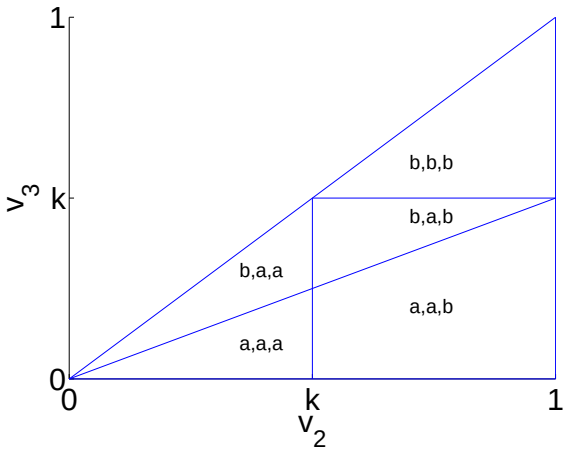


Figure 2.4: $(v_1 = 1)$ Regions where the number of allocated items remains constant and the compensation function is linear for 3 agents and 2 items. Each region is labeled with the coefficients used in the compensation function for each agent, e.g. (b,a,a) means that the compensation functions for agents 1,2, and 3 are bv_{-1}, av_{-2}, and av_{-3} respectively. One item is allocated to the left of the vertical line $v_2 = k$ and two items to the right.

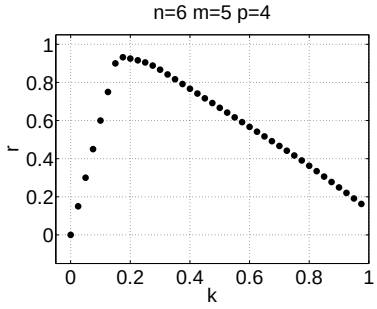


Figure 2.5: Performance curve

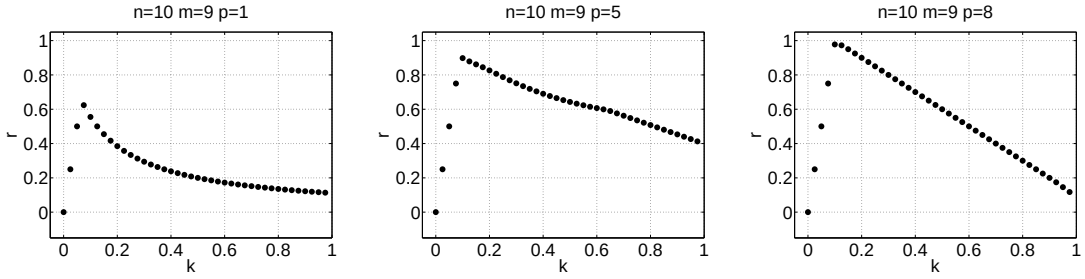


Figure 2.6: Performance curves for different settings of p

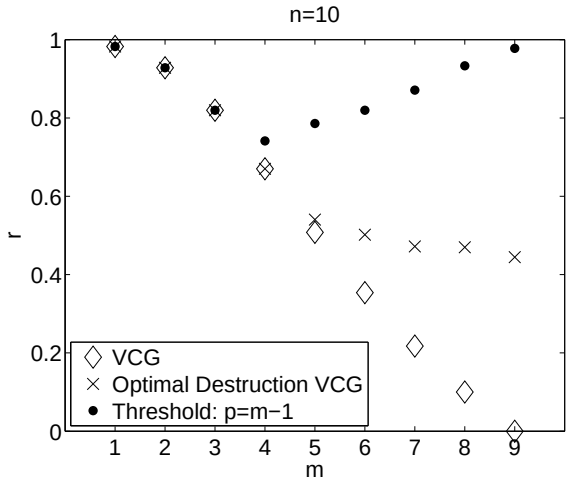


Figure 2.7: Performance of the mechanisms as a function of the number of items.

The mechanisms we find provide the most improvement when the number of items is close to the number of agents. In the extreme case when $n = m + 1$ our mechanism achieves the ratio of[6] $1 - \frac{2}{n^2-n}$, while the VCG mechanisms have the ratio of 0. Our ratio becomes closer to the VCG ratio as the number of items becomes smaller and approximately around $m = \frac{n}{2}$ the ratios and the mechanisms coincide. Figure 2.7 shows this trend for 10 agents and varying number of items.

In our threshold algorithm the parameter p is set to $m - 1$ allocating at least $m - 1$ items. Also plotted are the ratios achieved by the best VCG mechanism as well as the ratio achieved by the mechanism that first destroys a fixed number of items and then applies an optimal VCG mechanism (see deterministic burning mechanism in [50]). All mechanisms coincide when the number of items is 4 or fewer.

We now illustrate the kind of mechanisms we find. Recall the problem where six roommates (i.e., agents) need to distribute five seats in the car (i.e., items). For six agents and five items our mechanism is given by the following parameters: $k = \frac{1}{6}$, $p = 4$, $a = (0, 0, 0, 0, 0)$, $b = (0, 0, 0, -\frac{1}{6}, 1)$. Under the mechanism, the first 4 roommates always go and each of them pays $\frac{1}{6}v_5$. Allocation and payment for agents 5 and 6 is determined as follows:

- if $v_5 \geq \frac{1}{6}v_4$
  - roommate 5 goes and pays $\frac{1}{6}v_4$
  - roommate 6 does not go and gets $(v_5 - \frac{1}{6}v_4)$
- if $v_5 < \frac{1}{6}v_4$
  - roommate 5 does not go and gets 0
  - roommate 6 does not go and gets 0

[6] It is not difficult to check that this ratio is achieved by the following mechanism that meets the requirements of strategy-proofness, feasibility and anonymity: $\tau(v_{-i}) = \max(\frac{1}{n}v_{-i}^{m-1}, v_{-i}^{m})$ and the coefficients $a = (0, \ldots, 0)$ and $b = (0, \ldots, -\frac{1}{n}, 1)$

2.6 An Upper Bound

In previous sections, we described a computational approach for finding the best mechanism from the class of mechanisms satisfying certain simplifying assumptions on the payment and threshold functions. A natural question to ask is how much worse are the mechanisms we find compared to an optimal mechanism that is not restricted by any assumptions. Although, we do not know the ratio of the optimal mechanism, we can find an upper bound on the ratio.

Recall the problem of finding the best mechanism in Figure 2.1. The problem is difficult because optimization is over arbitrary functions and there is an infinite number of constraints: the constraints must hold for each $v \in V$. Consider enforcing the constraints only on a finite subset of points $\hat{V} \subset V$. The ratio achieved by an optimal solution to the problem with only a subset of constraints enforced is at least as high as the ratio achieved by an optimal solution when all constraints are enforced, thus providing an upper bound.

Checking constraints on a finite set of value profiles $\hat{V}$ involves a finite the set of compensations and thresholds. For example, if $\hat{V} = \{v^*, v'\}$ with $v^* = (x^*, y^*, z^*)$ and $v' = (x', y', z')$, then we only need to know the values of c and τ at (y^*, z^*), (x^*, z^*), (x^*, y^*), (y', z'), (x', z'), and (x', y'). An upper bound can be found by solving the optimization problem where values of the threshold and compensation functions are given by the variables $\tau_{v_{-i}}$ and $c_{v_{-i}}$ for all relevant v_{-i}.

We cannot solve the problem even with a finite number of constraints because some of the constraints are nonlinear. To see why this is the case, consider an example with 3 agents. The value of the variable τ_{xz} determines whether or not the last term is present in the feasibility constraint for the value profile (x, y, z)

$$c_{yz} + c_{xz} + c_{xy} - \tau_{yz} - \tau_{xz}\mathbf{1}_{\{y \geq \tau_{xz}\}} \leq 0$$

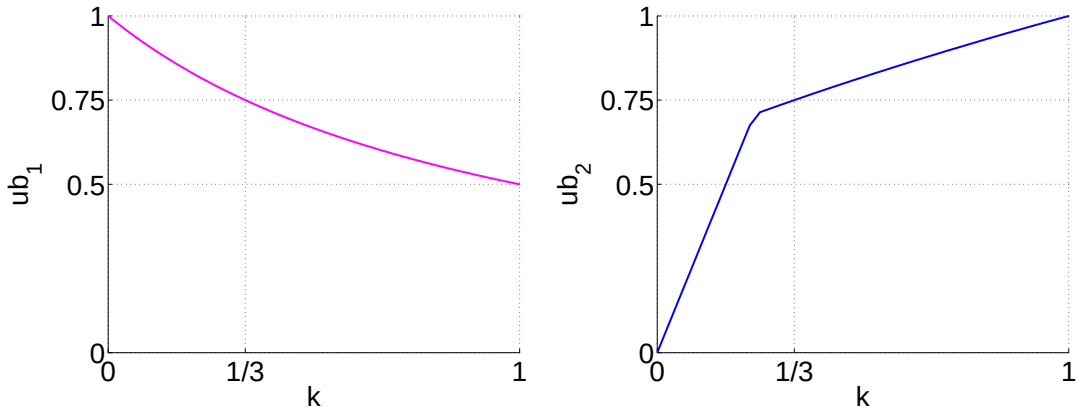
However, if the allocation is determined for all value profiles $v \in \hat{V}$, then the optimization problem is linear and can be solved using linear programming. For instance, the feasibility constraint for the profile (x, y, z) is linear when agents 1 and 2 are allocated

$$c_{yz} + c_{xz} + c_{xy} - \tau_{yz} - \tau_{xz} \leq 0$$

There are multiple threshold values that support a given allocation. While choosing an allocation does not determine the values of the threshold variables, it does place restrictions on the values the thresholds can take. For instance, allocating to agents 1 and 2 when values of the agents are (x, y, z) means that the values of agents 1 and 2 are above the corresponding threshold, but the value of agent 3 is below:

$$x \geq \tau_{yz}$$
$$y \geq \tau_{xz}$$
$$z \leq \tau_{xy}$$

Before we proceed, we assume that the mechanisms we consider satisfy a natural property: the ratio achieved by a mechanism does not depend on units in which utility is measured. For example, the ratio remains the same when we change the units from dollars to pounds. In other words, the ratio must be homogeneous of degree 0. This is achieved when the threshold and compensation functions are homogenous of degree 1.



(a) Upper bound when allocating one item for any value profile $(1, k, \cdot)$.

(b) Upper bound when allocating two items for all value profiles $(1, x, \cdot) \mid x \geq k$.

Figure 2.8: Upper Bounds

In the next section we derive an upper bound for the simplest non-trivial problem: the problem with three agents and two items.

2.6.1 An Upper Bound for Three Agents and Two Items

In a problem with two items, either one or two items must be allocated for any profile of values to achieve a non-zero ratio. First, we characterize how the ratio is affected if one item is allocated when the profile of values is $(1, k, x)$. The maximum social surplus of $1 + k$ is achieved when agents 1 and 2 are allocated and no money is burnt. If only one item is allocated, the highest social surplus is 1. It is achieved when the item is allocated to agent 1 and no money is burnt. The corresponding percentage of the social surplus achieved is $\frac{1}{1+k}$. Since the percentage is at most $\frac{1}{1+k}$ for the profile we considered, the (worst-case) ratio is also at most $\frac{1}{1+k}$. In other words, $\frac{1}{1+k}$ is an upper bound on the ratio when one item is allocated for a value profile $(1, k, \cdot)$. A graph of the upper bound as a function of the value of the second agent appears in Figure 2.8(a).

The value of the upper bound at $k = \frac{1}{3}$ is .75; i.e., the ratio is at most .75 if one item is allocated for any value profile $(1, \frac{1}{3}, x)$ s.t. $x \leq \frac{1}{3}$. The value of the upper bound at $k = 1$ is .5; i.e., the ratio is at most .5 if one item is allocated for any value profile $(1, 1, x)$ s.t. $x \leq 1$. Allocating one item when the value of the second agent is above $\frac{1}{3}$ results in an upper bound below .75. Therefore, a mechanism that achieves a ratio above .75 must allocate two items whenever the value of the second agent is above $\frac{1}{3}$. More generally, if a mechanism achieves the ratio above $\frac{1}{1+k}$, it must allocate two items for all value profiles with $v_2 \geq k$. Next, we compute an upper bound on the ratio that can be achieved when two items are allocated for all such value profiles.

Knowing the number of items allocated does not tell us which agents get the items. It seems natural that the agents with highest values should be allocated. It is indeed the case for the problem with three agents and two items as we show in the Appendix. Therefore, in the case considered here, the allocation is determined for all value profiles with $v_2 \geq k$. As we argued earlier, the optimization problem posed in Figure 2.1 can be

viewed as a linear program when the allocation is determined for all value profiles. To make the number of variable and constraints finite, we restrict attention to a finite subset of value profiles for which two items are allocated. In particular we pick the profiles where the value of each agent is 1, k, or 0; i.e., we only consider the value profiles $(1,1,1)$, $(1,1,k)$, $(1,1,0)$, $(1,k,k)$, and $(1,k,0)$. Note that $(1,0,0)$ is not included as $v_2 = 0 < k$ and, therefore, only one item is allocated. The optimization problem in Figure 2.1 becomes a linear problem with the variables τ_{11}, τ_{1k}, τ_{10}, τ_{kk}, τ_{k0}, c_{11}, c_{1k}, c_{10}, c_{kk}, c_{k0}. We solve the linear program for each value of k and plot the resulting upper bound in Figure 2.8(b).

The value of the upper bound at $k = \frac{1}{3}$ is $.75$ telling us that the ratio is upper bounded by $.75$ if we allocated two items *for all* profiles with $v_2 \geq \frac{1}{k}$. But we also know that, if one item is allocated *for any* of those profiles the ratio is below $.75$. Therefore, no matter how we allocate the items, the highest ratio we can hope for is $.75$. In the next section, we find a mechanism that achieves this ratio.

2.7 Optimal Solution for Three Agents and Two Items

In this section, we describe a mechanism with the ratio of $.75$. The ratio is the same as the upper bound meaning that the mechanism is optimal and the upper bound is tight. Exact details of our search for the best mechanism may not be very interesting, so instead we provide a high level description of how the mechanism was discovered.

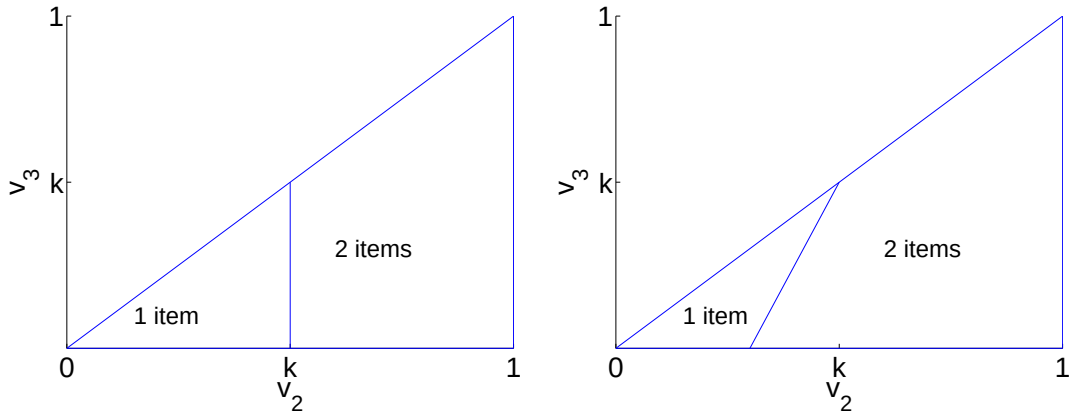
The search for a better mechanism was guided by upper bounds computations. First, we calculated an upper bound for mechanisms that use a threshold function satisfying Assumption 1 but make no assumptions on the compensation function. The values of the compensation variables in the upper bound computation helped us understand how to relax Assumption 2 until the ratio achieved coincided with the upper bound. This required going from 2 sets of linear coefficients allowed by Assumption 2 to 3 sets.

Making further progress involved relaxing the assumption about the threshold function. An illustration of the threshold function satisfying Assumption 1 may be helpful here. A threshold function satisfying Assumption 1 defines allocation regions along a vertical line at $v_2 = k$ as shown in Figure 2.9(a). We noticed that allocating two items for the value profile $(1,k,0)$ and keeping other allocation decisions consistent with the threshold function resulted in the upper bound of $.75$. The observation helped us find a threshold function that supports the upper bound of $.75$. Allocation regions for the new threshold are shown in Figure 2.9(b). Notice that two items are allocated for $(1,k,0)$.

A threshold function similar to the one illustrated in Figure 2.9(b) and a payment function given by 3 sets of linear coefficients constitute a mechanism that achieves the ratio of .75. Formally, optimal threshold and compensation functions are given by

$$\tau(v_{-i}) = \max\left(\frac{1}{4}(v^1_{-i} + v^2_{-i}), v^2_{-i}\right)$$

$$c(v_{-i}) = \begin{cases} \frac{2}{32}v^2_{-i} & \text{if } \frac{1}{9}v^1_{-i} \geq v^2_{-i} \\ \frac{10}{32}v^1_{-i} + \frac{11}{32}v^2_{-i} & \text{if } \frac{1}{3}v^1_{-i} \geq v^2_{-i} \geq \frac{1}{9}v^1_{-i} \\ \frac{4}{32}v^1_{-i} + \frac{20}{32}v^2_{-i} & \text{otherwise} \end{cases}$$



(a) Threshold satisfying Assumption 1. (b) Threshold needed to achieve the upper bound of .75.

Figure 2.9: $(v_1 = 1)$ Allocation Regions

In fact, there are multiple optimal threshold and compensation functions. The ones we presented here were chosen based on the simplicity of the coefficients.

The regions where compensation is linear and allocation is fixed are shown in Figure 2.10. There are 16 regions, 11 more than in Figure 2.4.

2.8 Conclusion

We developed a practical methodology that improves upon previous contributions which were restricting attention to VCG mechanisms for technical convenience. Motivated by our characterization of strategy-proofness in terms of threshold and compensation functions (see Proposition 1), we imposed some restrictions on those functions which guarantee that the optimization problem can be solved via linear programming techniques. The key observation for this simplification is that linear inequalities hold at all points in a polytope if and only if they hold at its extreme points. Though it is possible that more intricate mechanisms would achieve an even greater social surplus, we observed that our approach already significantly improves upon the previous VCG analysis. The reason is that the combination of allocative efficiency, a characteristic feature of VCG mechanisms, and strategy-proofness may come at the cost of "burning" a lot of money. This insight is likely to prove helpful in other contexts as well.

The most striking illustration of the benefits of our approach in our problem is the allocation of $n-1$ items among n agents. No redistribution of VCG payments is possible in that case, and for some value profiles the amount of VCG payments is as high as the sum of the $n-1$ highest values. We find that destroying one item for some profiles of values significantly reduces the amount of payments. For example, the mechanism that destroys one item if the $(n-1)^{\text{th}}$ highest value is less than $\frac{1}{n}$ of the $(n-2)^{\text{th}}$ highest value guarantees that the amount of payment is less than $\frac{2}{n^2-n}$ of the sum of $n-1$ highest values.

Finding the solution to the general optimization problem (2.1) remains an important open question. Taking a step in that direction, we found an optimal solution for the problem with three agents and two items. This

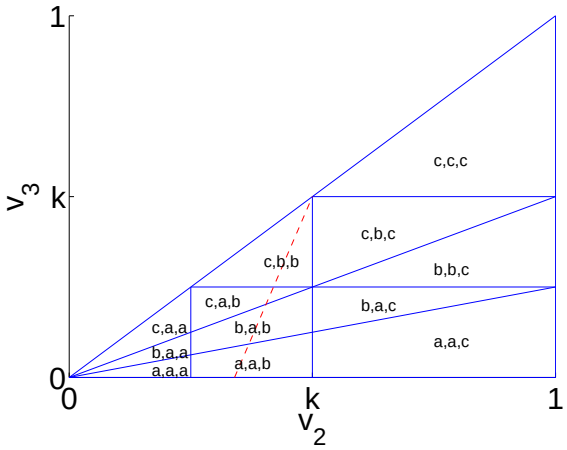


Figure 2.10: $(v_1 = 1)$ Regions where the number of allocated items remains constant and the compensation function is linear for three agents and two items. Each region is labeled with the coefficients used in the compensation function for each agent, e.g. (b,a,c) means that the compensation functions for agents 1,2, and 3 are bv_{-1}, av_{-2}, and cv_{-3} respectively. One item is allocated to the left of the dashed line and two items to the right.

was made possible by developing a technique for upper bounding the ratio of the best possible mechanism.

In the future, we plan to investigate more general allocation settings characterized by allocation of non-identical items, agents desiring more than one item, agents with utilities that depend on whether other agents receive the items (externalities), and common-value models.

2.9 Acknowledgements

Insightful conversations with Hervé Moulin and Yves Sprumont are gratefully acknowledged.

2.10 Appendix

Lemma 1 *In a mechanism that achieves a non-zero ratio in a problem with 3 agents* $f_i(v) = 1$ *implies that* $f_j(v) = 1$ *for all* $j < i$ *and for all* v.

Proof There are three possible ways to allocate an item to agent i without allocating an item to agent $j < i$: allocate to agent 2 while not allocating to agent 1, allocate to agent 3 while not allocating to agent 1, allocate to agent 3 while not allocating to agent 2.

We will use the fact that in a mechanism that achieves a non-zero ratio and is allocation feasible $\tau(x, x) = x$. Suppose $\tau(x, x) \neq x$ and consider the value profile (x, x, x). If $\tau(x, x) > x$, no items are allocated and the ratio is zero. If $\tau(x, x) < x$, three items are allocated and the mechanism is allocation infeasible.

Case 1: Let (x, y, z) denote the value profile where agent 2 is allocated but agent 1 is not. A threshold

supporting this allocation must satisfy:

$$\tau(y,z) > x > y > \tau(x,z)$$

Consider the value profile (y,y,z). Agents 1 and 2 are not allocated because $\tau(y,z) > x > y$. Agent 3 is not allocated because $\tau(y,y) = y > z$. The ratio of the mechanism is zero.

Case 2: Let (x,y,z) denote the value profile where agent 3 is allocated but agent 1 is not. A threshold supporting this allocation must satisfy:

$$\tau(y,z) > x > y > z > \tau(x,y)$$

As in case 1, consider the value profile (y,y,z). Agents 1 and 2 are not allocated because $\tau(y,z) > x > y$. Agent 3 is not allocated because $\tau(y,y) = y > z$. The ratio of the mechanism is zero.

Case 3: Let (x,y,z) denote the value profile where agent 3 is allocated but agent 2 is not. A threshold supporting this allocation must satisfy:

$$\tau(x,z) > y > z > \tau(x,y)$$

Consider the value profile (x,y,y). Agents 1 is allocated because $\tau(y,y) = y < x$. Agents 2 and 3 are allocated because $\tau(x,y) < z < y$. The mechanism allocated 3 items and is therefore allocation infeasible.

We showed that for $n = 3$ a mechanism that allocates to agent i but does not allocate to agent $j < i$ either achieves zero ratio or is allocation infeasible. Therefore any allocation feasible mechanism achieving non-zero ratio for $n = 3$ must allocate to agent $j < i$ if agent i is allocated.

Chapter 3

Using Simultaneous Best Response to Find Symmetric Bayes-Nash Equilibria in Auctions[1]

Finding Nash and Bayes-Nash equilibria in games is a hard problem both analytically and computationally. We restrict our attention to symmetric Bayes-Nash equilibria in auctions and propose a computational method that takes advantage of the symmetry of equilibria and structure of auction games. The method is iterated best-response where all players move simultaneously. We present experimental results for single unit first- and second-price auctions with discrete values and bids. The case of discrete bids and values has not been well studied before.

3.1 Introduction

Bayes-Nash equilibria (BNE) have been derived analytically only for the simplest auction settings [64]. Such settings include single-item first- and second-price auctions with continuous distributions of bidders' values.[2] However, very little research has been devoted to auctions with discrete bids and values. There is a general game solver Gambit (http://gambit.sourceforge.net) which is the state-of-the-art solver for finite normal and extensive form games. Reeves etal [92, 93] report that GAMBIT is only capable of solving relatively small games (e.g., finding BNE in the first-price sealed bid auction with 9 types and 9 values take an hour).

We propose a fast iterated best-response algorithm that takes advantage of

- structure of payoffs in first and second-price auctions
- the symmetry of equilibria

[1] Based on [79].

[2] Some literature refers to values as types or signals.

Experimental results in this chapter are for single auction settings and symmetrically distributed bidders' values. In the future, we plan to search for equilibria in the settings with multiple one-shot auctions where bidders have combinatorial valuations.

3.1.1 Symmetric Pure Strategy BNE

BNE can be interpreted as NE of the game where player's actions are strategies [85]. For example, consider an auction with 2 symmetric bidders, the set of values $V = \{0, 2\}$, and the set of bids $B = \{1, 2\}$. We can represent the Bayesian game in the normal form where strategies represent actions, i.e., an action is a vector of bids for each value:

	1,1	2,1	1,2	2,2
1,1	*			
2,1		*		
1,2			*	
2,2				*

Nash equilibria of the normal form representation of the game are the BNE of the corresponding Bayesian game.

Symmetric pure strategies correpond to cells along the diagonal marked with '*'. Thus, an exhaustive search for a pure strategy symmetric BNE would check each cells along the diagonal. The number of symmetric pure strategy profiles is the same as the number of actions of a player in the normal form game - the number of bids raised to the number of values $|B|^{|V|}$. Note that the number of symmetric strategies does not depend on the number of players.

We can search the space of symmetric strategies using *simultaneous* best response: starting from any cell in the payoff matrix, we calculate the best response of a player and let all players choose the best-response strategy. Because all players select the same best-response strategy, the payoff is on the diagonal.

3.2 Existence

The seminal paper by Nash [80] proves that finite *symmetric* games have a *symmetric* mixed strategy equilibria. However, pure strategy symmetric (or non-symmetric) equilibrium does not have to exist. At this stage, we focus on the games where we can find a pure-strategy symmetric equilibria or show that one does not exist. When it exists, symmetric pure strategy equilibrium is an appealing solution concept from the implementation point of view.

3.3 Model

We focus on a single object independent private-value model. There are n risk neutral bidders with quasi-linear utility functions. Each bidder's value of the object is a random variable distributed according to the same discrete probability distribution function f with support V. The distribution f is common knowledge.

All bidders have the same finite set of bids B resulting in the total number of distinct pure strategies equal to $|B|^{|V|}$. Utility $u_i(s; v_i)$ of bidder i is

$$\begin{aligned} & v_i - q_i && \text{if} \quad s_i(v_i) > \max_{j \neq i} s_j(v_j) \\ & (v_i - q_i)\frac{1}{\#\text{ties}} && \text{if} \quad s_i(v_i) = \max_{j \neq i} s_j(v_j) \end{aligned}$$

where q_i is the bid in the first-price auction and the second highest bid in the second-price auction. In words, the utility of a bidder is her value minus the price when the bidder submits the highest bid. When the bidder ties with #ties other bidders, the winner is chosen uniformly at random from the highest bidders.

3.4 Utility and BNE in Auctions

The strategy profile $s^* = (s_1^*, \ldots, s_n^*)$ is a BNE if for each bidder i and for each of i's values v_i, $b_i = s_i^*(v_i)$ solves

$$\max_{b_i \in B_i} \sum_{v_{-i} \in V_{-i}} u_i(s_1^*(v_1), \ldots, b_i, \ldots, s_n^*(v_n); v_i) f(v_{-i}) \tag{3.1}$$

One key observation is that, in the absence of ties[3], the only information relevant to a bidder's utility in first and second-price auctions is the maximum bid of the other bidders.

$$u_i(s_i(v_i), s_{-i}(v_{-i}); v_i) = u_i(s_i(v_i), \max_{j \neq i} s_j(v_{-i}); v_i)$$

We will refer to the maximum bid of the other bidders as the price. The price distribution g can be derived from the distribution of the other bidders' values.

$$\forall p \in B_i \quad g(p) = \sum_{v_{-i} \in V_{-i} \mid \max_{j \neq i} s_j^*(v_j) = p} f_i(v_{-i}) \tag{3.2}$$

The probability $g(p)$ that the price is p is the sum of the probabilities of all combinations of values of the other bidders that result in the maximum bid equal to p. Rewriting Equation 3.1 with the price distribution we get

$$\max_{b_i \in B_i} \sum_{p \in B_i} u_i(b_i, p; v_i) g(p) \tag{3.3}$$

The strategies $s^* = (s_1^*, \ldots, s_n^*)$ are a BNE if for each bidder i and for each of i's values v_i, $b_i = s_i^*(v_i)$ solves Equation 3.3. Note that $g(p)$ in Equation 3.3 is determined by the strategies of the other bidders.

Given g, Equation 3.3 is more compact than Equation 3.1. The summation in Equation 3.1 has V_i^{n-1} terms. The summation in Equation 3.3 has at most B_i terms. This difference becomes important when there are more than 2 bidders. We will show that in the symmetric case g can be calculated much faster than in Equation 3.2.

[3] We address the issue of ties in Section 3.7

3.5 Iterated Best Response

Strategy profile $s = (s_1, ..., s_n)$ is a symmetric BNE if for all i, s_i is the best response to the price distribution g resulting from $n-1$ bidders playing the same strategy s_i. We search for such profile via an iterated simultaneous best-response procedure starting from some initial distribution of prices.

- initialize g
- repeat until s are a symmetric equilibrium
  - for every $v_i \in V_i$
    - bidder i finds the bid $s_i(v_i)$ that is the best response to g.[4]
  - h is the frequency distribution of bids s_i
  - H is the corresponding cumulative distribution
  - $g(p) = H^{n-1}(p) - H^{n-1}(p-1)$

The strategies $s = (s_1, \ldots, s_n)$ are a symmetric BNE if s_i is a best response to the price distribution resulting from $n-1$ bidders playing s_i. We check this condition at the start of the loop.

In each iteration we calculate bidder i's best response s_i to the price distribution g. The best response is calculated for each value $v_i \in V_i$. At the end of an iteration, g is set to the price distribution resulting from $n-1$ bidders playing s_i. This price distribution is calculated using the cumulative distribution H of bids submitted by bidder i. The cumulative distribution of the maximum of $n-1$ bids distributed according to H is H^{n-1}. For integer prices p the probability function $g(p)$ can be calculated from H: $g(p) = H^{n-1}(p) - H^{n-1}(p-1)$.

3.6 Experiments

We test the procedure in first- and second-price auctions with risk-neutral bidders. The number of bidders ranges between 2 and 11. Bidders have integer values uniformly distributed between 0 and k. The parameter k ranges from 2 to 50. Given a value v and a price distribution, an optimal bid is calculated by comparing profits from bidding each of the $v+1$ bids $0, 1, \ldots, v$. If the bid is equal to the price, we use $\frac{1}{n}$ as an approximation of the probability that the bidder wins the auction.[5] The bidder pays his bid in the first-price auction and the price in the second-price auction. We tried two ways of initializing the price distribution: to zero and to the value distribution.

[4] If multiple bids are best responses, we set $s_i(v_i)$ to the mixed strategy of submitting any of the best-response bids with equal probability.

[5] The probability is exact only when there are two bidder. Section 3.7 describes how the probabilities can be calculated exactly. However, at the time of running the experiments, we used $\frac{1}{n}$ as an approximation. We re-ran some of the experiments with correct probabilities and confirmed that the results reported here still hold.

3.6.1 Second-Price Auctions

The second-price auction has a known dominant strategy of bidding the true value. The iterative procedure converges to the dominant strategy after 2 iterations when the price distribution is initialized to zero and after 1 iteration when the price distribution is initialized to the value distribution.

3.6.2 First-Price Auctions

The procedure converges to an equilibrium in most of our experiments with varying number of bidders and initial price distributions for values of k below 9. It takes under 8 iterations (usually 1 iteration) to reach convergence when the price distribution is initialized to the value distribution. It takes under 27 iterations (5 iterations on average) to reach convergence when the price distribution is initialized to zero. Table 3.1 illustrates the steps of the iterative procedure in the auction with 2 bidders, $k = 4$, and initial belief that the price is zero. Convergence is reached after 3 iterations.

price/value	0	1	2	3	4
price distr	1.0	0	0	0	0
BR strategy	0,1,2,3,4	1,2,3,4	1,2,3,4	1,2,3,4	1,2,3,4
price distr	0.04	0.24	0.24	0.24	0.24
BR strategy	0	1	2	3	4
price distr	0.2	0.2	0.2	0.2	0.2
BR strategy	0	1	2	3	4
price distr	0.2	0.2	0.2	0.2	0.2
BR strategy	0	1	2	3	4

Table 3.1: Sample Run of Iterated Myopic Best Response. "price distr" is the probability distribution of prices resulting from the opponent playing the best-response (BR) strategy to the price distribution from the previous iteration.

Bidding $\frac{n-1}{n}$ of the value is a BNE of a first-price auction when bidders' values are identically distributed according to a continuous uniform distribution [64]. Not surprisingly, the equilibrium strategies we find are similar to the equilibrium strategies for the continuous case. Examples of equilibria are in Table 3.2. The continuous counterparts for these discrete strategies are to bid $\frac{10}{11}$ and $\frac{1}{2}$ of the value respectively.

k	# bidders	equilibrium strategy
8	11	0:0,1:0,2:1,3:2,4:3,5:4,6:5,7:6,8:7
7	2	0:0,1:0,2:1,3:1,4:2,5:2,6:3,7:3

Table 3.2: Equilibria in First-Price Auctions. 3:1, 4:2,... denotes the equilibrium strategy of bidding 1 when the value is 3 and bidding 2 when the value is 4.

In the cases when myopic best response does not converge to an equilibrium, it converges to a cycle.

Surprisingly, when the price distribution was initialized to value distribution, the procedure converged to an equilibrium for any odd value of k.

3.7 Ties

Here we address the issue of ties, which was ignored when we said that only the bid and the price affect one's utility.

$$\Pr(\text{tie and win}) = \sum_{m=1}^{n-1} \underbrace{\frac{1}{m+1}}\underbrace{\binom{n-1}{m}}\underbrace{h^m(b_i)H^{n-m-1}(b_i-1)} \tag{3.4}$$

The first term is the probability of winning with tied with m bidders. The second term is the number of ways to choose m bidders to tie with. The last term is the probability of a tie with exactly m bidders. This computation takes $n-1$ steps assuming that all the terms inside the summation are computed once per iteration of best response. The computation of ties has been previously done in [26].

3.8 Related Work

The related work pertains to symmetric risk-neutral bidders and the independent private value model.

3.8.1 Analytical Results

Existence results on equilibria are summarized in the table. References and comments appear below.

	DB DV	DB CV	CB DV	CB CV
FP	M	P	M	P
SP	M	P	D	D

Table 3.3: Equilibria in Auctions: M - symmetric mixed, P - symmetric pure, D - dominant; DB - discrete bids, DV - discrete values, CB - continuous bids, CV - continuous values

Dominant Strategy Solvable Second Price (SP) with continuous bids and continuous values [106] and SP with continuous bids and discrete values.

Symmetric Pure Strategy Equilibrium First Price (FP) with continuous bids and continuous values [64].

Evenly Spaced Discrete Bids FP ([26]) and SP with continuous values [112].

Note that there is no dominant strategy equilibrium in SP with continuous values. We can illustrate this with an example showing that for some value there is no bid that is dominating all the other bids. Suppose there are 2 bidders, the possible bids are 0 and 1, and the values are uniformly distributed between 0 and 1. A bidder with the value close to one (say, value is .9) prefers bidding 1 if the expected bid of the other bidder is close to zero, but he prefers bidding 0 when the expected bid of the other bidder is above .9.

Arbitrary Discrete Bids SP with continuous values [74].

Symmetric Mixed Strategy Equilibrium FP with continuous bids and discrete values [37]. FP and SP with discrete bids and discrete values (normal form game).

3.8.2 Computational Results

Reeves and Wellman [92] describe a procedure for computing best-response strategies and finding Bayes-Nash equilibrium for two-player infinite games with types drawn from a piecewise-uniform distribution. Sureka and Wurman [99] use iterated tabu best response to search for NE in normal form games. Their work is not restricted to symmetric equilibria and considers smaller normal form games.

3.9 Conclusion and Future Work

While the problem of finding equilibria in general games appears not to be amenable to a single computational solution, auctions impose a specific structure on the payoffs of the participants, making the search for equilibria more manageable. The results in this chapter are rather preliminary but they demonstrate that a computational approach such as the one investigated here can provide insights into understanding equilibria in auctions. An example of a question that can be answered with the help of the simulations is whether the strategies that are part of the cycle contain support for a mixed strategy equilibrium. The answer to the question is no as we computationally found a counter-example. A more general investigation of what can be learned from cycles is a direction for future work. A stochastic stability analysis might prove fruitful. Also, it would be interesting to consider multiple one-shot simultaneous auctions and bidders with combinatorial valuations. Procedures with better convergence properties than best-response dynamics should also be considered. Some of these directions have recently been explored in [39].

Chapter 4

Bidding in Keyword Auctions[1]

We model budget-constrained keyword bidding in sponsored search auctions as a *stochastic multiple-choice knapsack problem* (S-MCKP) and propose a new algorithm to solve S-MCKP and the corresponding bidding optimization problem. Our algorithm selects items online based on a threshold function which can be built/updated using historical data. Our algorithm achieved about 99% performance compared to the offline optimum when applied to a real bidding dataset. With synthetic dataset and *iid* item sets, its performance ratio against the offline optimum converges to one empirically with increasing number of periods.

4.1 Introduction

Sponsored search is an effective way of monetizing search activities where advertisers pay to place their ads on search results pages for specific user keyword queries. Using an automated auction mechanism, search engine companies alleviate themselves from the burden of pricing and placing ads and shift the burden to advertisers. On the advertisers side, large companies spend millions of dollars each year to bid on thousands of keywords, and it is important for them to automate and optimize the bidding process to achieve the best return on investment (ROI). In this work we focus on the bid optimization problem for an advertiser with budget constraints. Formally, we try to address the following problem: for each keyword and each time period, how much should the advertiser bid (i.e., which position to obtain), so as to maximize ROI of the ads given a fixed budget and a fixed time horizon?

For a given keyword, there are multiple slots in the search results page that the auctioneer needs to allocate to different ads from different advertisers taking into account the advertisers' bidding price, ad quality, and other factors. There are different ad ranking and pricing schemes most of which are variations of *rank-by-price* and *pay-per-click* [31, 102, 66], where the advertiser in the i-th position pays the bid of the $(i+1)$-th advertiser whenever its ad is clicked by a user. No matter what ranking and pricing scheme the auctioneer deploys, for a given keyword, an advertiser can bid appropriately to get its ad placed in any ad position. For

[1] This work was done at HP Labs. Published as [114].

each ad slot, the advertiser incurs a *cost* (the fee that the auctioneer charges for each user click), obtains a *revenue* (the expected value-per-click), and a *profit* (the difference between revenue and cost). Naturally, we can model each ad position as an item with associated weight (cost) and value (either revenue or profit). The advertiser (or the agent acting on behalf of the advertiser) has a budget constraint, and it naturally corresponds to the knapsack capacity. Furthermore, one policy most auctioneers enforce is that each advertiser can have *at most one* ad appear on each keyword results page. This corresponds to that at most one item from each item set can be taken in the Multiple-Choice Knapsack Problem (MCKP), a well-known variation of the classic Knapsack Problem (KP). Therefore we can model the budget constrained bidding problem as a MCKP.

Compared to traditional offline setting of knapsack problems, keyword bidding is by nature online and stochastic. As any keyword auction is open to all advertisers with a positive budget, advertisers can join/leave the auction at any time and change their bids arbitrarily. Bidders often exhibit strategic bidding behaviors (e.g., overbidding [30], vindictive bidding[113]) which make the market more dynamic. Furthermore, sponsored search is driven by user queries/clicks. Even though the number of user queries/clicks is statistically stable over long time horizon, periodical spikes/drops are common and quite unpredictable. All these factors contribute to the online and stochastic nature of the underlying bidding problem and this motivates us to work on the stochastic and online version of MCKP.

4.1.1 Our Contributions

In this work we model the budget constrained bidding problem for keyword auctions as the online multiple-choice knapsack problem, design efficient algorithms for S-MCKP and translate it back to solve the budget-constrained bidding problem. Our algorithms are simple, easy to implement, and achieve a performance ratio consistently over 90% with both synthetic and real bidding data.

Even though the Online 0/1 Knapsack Problem is a well-studied problem in Operations Research and Online Algorithms, we are aware of no prior work on S-MCKP. By utilizing previous work on stochastic Online-KP, we design a simple algorithm for S-MCKP with performance ratio approaching one empirically while the number of time periods goes to infinity.

Our algorithms for keyword bidding as well as S-MCKP assume input item sets are independent and identically distributed (iid), however our algorithms do not require any knowledge of the distribution. Our algorithms are based on maintaining a threshold function, and the threshold function can be built in advance using historical training dataset, or can be built from scratch and updated overtime during the execution of the algorithm. The machine learning capability improves the bidding performance and makes our algorithm more attractive to field deployment.

The rest of the chapter is organized as follows. In Section 4.2 we briefly discuss related work. In Section 4.3, we introduce terminology related to online knapsack problems and describe Lueker's algorithm for the stochastic online knapsack problem. In Section 4.4, we describe our algorithm for S-MCKP and prove some properties of the algorithm. Section 4.5 is dedicated to modeling the bidding optimization problem as S-MCKP, and Section 4.6 is devoted to experimental evaluation of our algorithms for S-MCKP and the keyword bidding problem. We conclude in Section 4.7.

4.2 Related Work

Keyword Auctions and Bidding Optimization. Over the past few years, keyword auctions have attracted a lot of attention both from the auctioneer's perspective ([31, 102, 3, 66]) and the advertiser's perspective ([30, 6, 100, 113]). For revenue maximization for the auctioneer with budget-constrained bidders, there are a few papers with various complexities to model keywords, slots, and clicks ([16, 1, 75, 18, 2]).

For bidding optimization for the advertiser, Kitts and LeBlanc [61] describe various bidding heuristics. Borgs et al. [15] propose a bidding strategy which over time equalizes the ROI over all keywords. Rusmevichientong and Williamson [95] discuss how to learn the CTRs for various keywords over time and select keywords accordingly. Most recently, Feldman et al. [35] studied variants of the bidding optimization problem where the objective is to maximize the number of clicks, with possibly complicated interactions among many keywords, and Cary et al. [21] analyzed properties of greedy bidding strategies.

Chakrabarty et al. [23] modeled the budget-constrained bidding optimization problem as Online-MCKP and designed competitive algorithms for Online-MCKP and evaluated the algorithms using both synthetic and real datasets. Their algorithms depend on some input parameters and thus a good bidding performance depends on either knowing these parameters or tuning them appropriately. They emphasize the worst-case performance guarantee while this work focuses on the average-case performance with stochastic input. In addition, the performance of their algorithm on the same real bidding data is between 90%-95%, however ours is around 99%.

Knapsack Problems and Online Algorithms. Variants of knapsack problems were studied extensively in Combinatorial Optimization and Operations Research. For a comprehensive exposition of this topic, see the textbook by Psinger et al. [59]. Online knapsack problems were first studied by Marchetti-Spaccamela and Vercellis [72] and showed that in the general case, there exists no online algorithm achieving any non-trivial competitive ratio. Many special cases of the problem have been studied afterwards, among them the stochastic online knapsack problem [71, 86, 63], and the online partially fractional knapsack problems [83].

Among all this work, Lueker [71] designed a simple threshold based online algorithm for the classic 0/1 knapsack problem, assuming that the weight and value of all items are iid. Under the iid assumption and perfect knowledge of the distribution, Lueker's algorithm achieves an *optimal* performance ratio of $1 - O(\log\log n/\log n)$ against the offline optimum. Instead of assuming that the items are iid, Chakrabarty et al. [23] assumes that all items have their value/weight ratio upper bounded by U and lower bounded by L, for two positive constants U, L. Assuming further that all items are small compared to the knapsack capacity, they designed both deterministic and randomized threshold based algorithms for the online 0/1 knapsack problem achieving an (optimal) competitive ratio $\log(U/L)+1$. They also extend the algorithm to the multiple-choice setting and obtains a competitive ratio of $\log(U/L) + 2$.

4.3 Onlike Knapsack Problems and Lueker's Algorithm

In this section we introduce the Online Knapsack Problem (Online-KP) and describe Lueker's algorithm to solve stochastic Online-KP.

The 0/1 Knapsack Problem (KP) is as follows: given a set of items $\{(w_i, v_i) \mid 1 \leq i \leq n\}$ and a knapsack capacity C, select a subset of items to maximize the total value of selected items while the total weight is bounded by C. Throughout the chapter, for each item i, we call w_i its *weight*, v_i its *value*, and the ratio between value and weight its *efficiency* ($e_i = v_i/w_i$). The Online Knapsack Problem (Online-KP) is the same as the 0/1 KP except that items arrive online one at a time. At each time period t, item t arrives, and the algorithm has to decide whether to select item t or not. The Stochastic Online-KP is the same as Online-KP with an extra assumption that the (weight, value) pair of each item is randomly drawn from the same joint distribution. Naturally, we assume that the knapsack capacity is proportional to the number of items ($C = \Theta(n)$), and all items are small compared to the overall knapsack capacity ($w_t = O(1)$ and $v_t = O(1)$, $\forall\, t$).

Lueker's Algorithm [71] for the Stochastic Online-KP is based on a threshold function that is generated using the distribution of items. All items are assumed to be iid. Only items with efficiency at least the *threshold efficiency* are included in the solution. The algorithm for Online-KP is in Figure 4.1.

Algorithm ALG-Lueker-OKP
Input: items (w_t, v_t) for $t = 1, \ldots, n$;
knapsack capacity C; threshold function g
Output: items to take
1. for each item t from 1 to n
if $e_t \geq g(\frac{C}{n-t+1})$ and $w_t \leq C$
take item t
$C := C - w_t$
2. return items taken

Figure 4.1: Lueker's Algorithm for Online-KP.

The Threshold Function. The main part of the algorithm is the threshold function g which maps the average remaining capacity per time period to an efficiency value, denoted *threshold efficiency*. The threshold efficiency is such that the expected weight of the remaining items with efficiency at least the threshold efficiency is equal to the remaining capacity:

$$C \;=\; \mathrm{E}_{w_i,v_i}\left[\sum_{i=1}^{n} w_i\, \mathbf{1}_{\{\frac{v_i}{w_i} \geq e^*\}}\right] \;=\; \sum_{i=1}^{n} \mathrm{E}_{w_i,v_i}\left[w_i\, \mathbf{1}_{\{\frac{v_i}{w_i} \geq e^*\}}\right]$$

The second equality above uses the linearity of expectation. Since all items are iid, thus

$$\begin{aligned} C &= \sum_{i=1}^{n} \mathrm{E}_{w_i,v_i}\left[w_i\, \mathbf{1}_{\{\frac{v_i}{w_i} \geq e^*\}}\right] \;=\; n\, \mathrm{E}_{w,v}\left[w\, \mathbf{1}_{\{\frac{v}{w} \geq e^*\}}\right] \\ \frac{C}{n} &= \mathrm{E}_{w,v}\left[w\, \mathbf{1}_{\{\frac{v}{w} \geq e^*\}}\right] \end{aligned}$$

Let

$$f(e) \;\equiv\; \mathrm{E}_{w,v}\left[w\, \mathbf{1}_{\{\frac{v}{w} \geq e\}}\right], \tag{4.1}$$

then the threshold function is $g = f^{-1}$, the inverse of f. f maps the efficiency e to the expected item weight among items with efficiency at least e, while g maps the average capacity per item to the efficiency.

4.4 Approximation Algorithms for S-MCKP

In this section we describe our algorithm for S-MCKP. Before we introduce the algorithm, we first define the problem briefly. The Multiple-Choice Knapsack Problem is a generalization of the 0/1 KP: Given a collection of item sets $\{N_t \mid t = 1, \ldots, n\}$ where $N_t = \{(w_{ti}, v_{ti} \mid 1 \leq i \leq n_t\}$ for each t and a knapsack capacity C, select at most one item from each item set to maximize the total value of selected items while the total weight of selected items is bounded by C. The Online MCKP is the online version of MCKP where item set N_t arrives at time t and the algorithm needs to select at most one item from N_t. Stochastic MCKP is the same as Online MCKP with an extra assumption that all item sets are iid random variables. Naturally we assume $C = \Theta(n)$, $w_{ti} = O(1), v_{ti} = O(1)\ \forall\ t, i$.

Our algorithm for the S-MCKP is based on Lueker's Algorithm for Stochastic Online-KP (described in Section 4.3) and an approximation for MCKP [59]. We first describe the approximation for MCKP (Section 4.4.1), then an approximation for the threshold function (Section 4.4.2), and finally the overall algorithm (Section 4.4.3).

4.4.1 Converting Item Sets to Incremental Items

Approximation for MCKP modifies the items from each item set so that taking multiple items is equivalent to taking one original item ([59], p.320). An item i is *dominated* by another item j if $w_j \leq w_i$ and $v_i < v_j$. An item i is *LP-dominated* by items j and k if i is dominated by a convex combination of j and k. Equivalently, if $w_j < w_i < w_k$ and $v_j < v_i < v_k$, then i is LP-dominated by j, k if

$$\frac{v_k - v_i}{w_k - w_i} \geq \frac{v_i - v_j}{w_i - w_j}.$$

The algorithm to remove all dominated and LP-dominated items and generate incremental items is described in Figure 4.2. The algorithm consists of two steps, first sorting items in increasing weight order, then removing dominated and LP-dominated items repeatedly. The second step clearly takes linear time, thus the total running time is dominated by the first step of sorting, thus $O(n \log n)$ time.

Once all dominated and LP-dominated items are removed, and remaining items are sorted in increasing weight order, then for three adjacent items $i - 1, i, i + 1$, we have

$$\frac{v_i - v_{i-1}}{w_i - w_{i-1}} = \frac{\bar{v}_i}{\bar{w}_i} = \bar{e}_i > \frac{v_{i+1} - v_i}{w_{i+1} - w_i} = \frac{\bar{v}_{i+1}}{\bar{w}_{i+1}} = \bar{e}_{i+1}.$$

Thus the efficiency of incremental items are monotone decreasing: $\bar{e}_1 > \bar{e}_2 > \ldots > \bar{e}_\ell$. Taking incremental items $1, \ldots, i$ from the set Q is equivalent to taking item i from the set $\mathbb{N}_t$.

4.4.2 Approximating the threshold function

To compute an approximate solution for S-MCKP, we first convert each item set into a set of incremental items, and try to apply Lueker's Algorithm for Online-KP to these incremental items. Lueker's algorithm requires requires as an input the threshold function, which is not available to us. In this section we discuss how to compute an approximate threshold function using sample item sets, and how to update the threshold function over time.

Algorithm ALG-Gen-Incr-Items
Input: an item set $\mathbb{N}_t = \{(w_{ti}, v_{ti}) \mid i = 1, \ldots, n_t\}$
Output: incremental items
1. sort items according to increasing weights
2. /** remove dominated and LP-dominated items **/
   let Q be a queue with initially one element $(0, 0)$
   for i from 1 to n_t
      push element i into the queue
      (ℓ always denote the last element of Q)
      if $w_\ell = w_{\ell-1}$
         remove from Q either item ℓ or $\ell - 1$
         with smaller value
      while $\ell > 2$ and $\frac{v_{\ell-1} - v_{\ell-2}}{w_{\ell-1} - w_{\ell-2}} \leq \frac{v_\ell - v_{\ell-1}}{w_\ell - w_{\ell-1}}$
         remove item $\ell - 1$ from Q
3. /** create incremental items from items in Q **/
   let $\{(w_i, v_i) \mid 1 \leq i \leq \ell\}$ denote the items in Q
   $\bar{w}_1 = w_1$, $\bar{v}_1 = v_1$
   $\bar{w}_i = w_i - w_{i-1}$, $\bar{v}_i = v_i - v_{i-1}$, $\forall\, 2 \leq i \leq \ell$
4. return $\{(\bar{w}_i, \bar{v}_i) \mid 1 \leq i \leq \ell\}$

Figure 4.2: Algorithm to generate incremental items from an item set

Generating threshold function from a sample. Given a set of training item sets, we can transform them into a collection of incremental items. The distribution of incremental items may not be known or have a closed-form representation, however we can approximate it if we have a reasonably large sample size.

Given a sample set of m incremental items, we can approximate the threshold function given by Eq. 4.1 with the average over all the sample points. Formally, we can use $\tilde{f}$ to approximate f where

$$\tilde{f}(e) \;\equiv\; \frac{1}{m} \sum_{i=1}^{m} w_i \, \mathbf{1}_{\{e_i \geq e\}} \tag{4.2}$$

Assuming that the incremental items are sorted in decreasing order of efficiency, then $\forall\, e \in (e_{i+1}, e_i]$, $\tilde{f}(e)$ is equal to $\tilde{w}_i \equiv (w_1 + \ldots + w_i)/m$. Therefore $\tilde{f}$ is a piecewise constant function, and it can be represented as a sorted list of pairs $\{(e_i, \tilde{w}_i) \mid 1 \leq i \leq m\}$ with $\{e_i\}$ monotone decreasing and $\{\tilde{w}_i\}$ monotone increasing. The threshold function can be computed using the algorithm in Figure 4.3.

Update Threshold Function Online. We can update the threshold function as we are presented with new sets of incremental items. It is convenient to represent the threshold function by a collection of efficiencies $e_1 > e_2 > \ldots > e_k$ sorted in decreasing order and a collection of corresponding weights $w_1 < w_2 < \ldots < w_k$ in increasing order where $w_i = f(e_i)$. Initially the collections can be empty in which case the threshold function is generated using the first item set. See the algorithm in Figure 4.4.

Algorithm ALG-Gen-Threshold
Input: set of incremental items $\{(w_j, v_j) \mid j = 1, \ldots, m\}$
Output: threshold function f
1. sort items in decreasing order of efficiency.
   let $e_j = v_j/w_j, \forall\, j$, then $e_1 \geq e_2 \geq \ldots \geq e_m$.
2. $f(e_1) = \frac{w_1}{m}$
3. $f(e_i) = f(e_{i-1}) + \frac{w_i}{m}, \quad \forall\, 2 \leq i \leq m$
4. return f

Figure 4.3: Algorithm for generating the threshold function.

Algorithm ALG-Update-Threshold
Input: threshold function $\tilde{f} = \{(e_i, w_i) \mid 1 \leq i \leq k\}$,
a set of incremental items $\{(\tilde{w}_j, \tilde{v}_j) \mid 1 \leq j \leq m\}$
Output: updated $\tilde{f}$
1. /** normalize weights **/
   $w_i = w_i \frac{k}{k+m} \qquad 1 \leq i \leq k$
   $\tilde{w}_j = \tilde{w}_j \frac{1}{k+m} \qquad 1 \leq j \leq m$
2. /** update weights **/
   $w_i = w_i + \sum_{\tilde{e}_j \geq e_i} \tilde{w}_j \qquad 1 \leq i \leq k$
3. /** create a list of sorted (e, w) pairs **/
   for j from 1 to m
   if there is no pair in f with efficiency $\tilde{e}_j = \tilde{v}_j/\tilde{w}_j$
   $i = \arg\max_i\{e_i \geq \tilde{e}_j\}$
   add $(\tilde{e}_j, w_i + \tilde{w}_j)$ to the list of new pairs
4. linearly merge the new list and $\tilde{f}$ to get the
   updated $\tilde{f}$

Figure 4.4: Algorithm for updating the threshold function.

4.4.3 An Approximation Algorithm for S-MCKP

We are now ready to describe our algorithm for S-MCKP. For each item set arriving online, we use ALG-Gen-Incr-Items given in Figure 4.2 to generate incremental items for the item set and use the approximate threshold function to select incremental items for the current time period. Since we described how to generate the approximate threshold function and update it in Section 4.4.2, we are now ready to describe the whole algorithm.

Algorithm ALG-S-MCKP
Input: item set $\mathbb{N}_t$ for $t = 1, \ldots, n$;
knapsack capacity C;
(optional) training item sets
Output: items to take
1. (optional) /** generate threshold function f from
training items sets **/
create incremental items from training item sets
using ALG-Gen-Incr-Items
r is the average number of incremental items per set
generate f using ALG-Gen-Threshold with these
incremental items as input
2. for t from 1 to n
create incremental items from item set $\mathbb{N}_t$
usin ALG-Gen-Incr-Items
(optional step)
update f (using ALG-Update-Threshold) and r
$e = f^{-1}(\frac{C}{r(n-t+1)})$
/** $r(n-t+1)$ is the expected number of
remaining incremental items **/
select incremental items with efficiency at least e
$\bar{w}$, $\bar{v}$ are the total weight and value of selected
incremental items
if $\bar{w} \leq C$
take item $(\bar{w}, \bar{v})$.
$C := C - \bar{w}$

Figure 4.5: Algorithm for S-MCKP.

The algorithm for S-MCKP is in Figure 4.5. It consists of two phases, where the first is optional, and it depends on whether training item sets are available. For the second phase, the algorithm decides whether or not to take an item at time period t using the threshold function, and updates the threshold function if necessary.

4.5 Keyword Bidding as S-MCKP

Sponsored search auctions are used by search engine companies to sell ad positions to advertisers on search results page, where popular query terms are treated as "keywords". An auction is set up for each keyword

where advertisers submit bids and compete for different ad positions. The auction mechanism determines how to rank and price ads, using factors like the bidding prices and ad qualities, or even budgets of different advertisers. Among many variations of ad ranking and pricing schemes, most are based on *rank-by-price* and *pay-per-click*. In this mechanism, assuming that bidding prices are sorted in decreasing order ($b_1 \geq b_2 \geq \ldots \geq b_n$), bidder i obtains position i, and is charged a fee $p_i = b_{i+1}$ whenever a user clicks on its ad. [2] No matter what ranking and pricing scheme the auctioneer deploys, for a fixed avertiser and a fixed keyword, the advertiser can obtain any position with an appropriate bidding price. For each ad slot, the advertiser incurs a *cost* (the fee that the auctioneer charges for each user click), obtains a *revenue* (the expected value-per-click), and a *profit* (the difference between revenue and cost). Naturally, we can model each ad position as an item with associated weight (cost) and value (either revenue or profit). Without loss of generality, we focus on profit.

A typical advertiser has a budget for some time horizon (e.g. daily, weekly, quaterly or annually) and wants to purchase a certain set of keywords to maximize its total ROI. The profit of the advertiser is equal to the total amount of expected revenue from search marketing minus the total amount of marketing cost. We can discretize the time horizon into small time periods and assume that the bidding prices of all advertisers do not change over each small time period. Formally, we can model the bidding optimization problem as a multiple-choice knapsack problem as follows. Given multiple keywords $k \in K$, multiple time periods when the advertiser places bids $t \in \{1, \ldots, T\}$, and multiple positions $s \in \{1, \ldots, S\}$, the item set N_t^k consists of items (w_{ts}^k, v_{ts}^k) for all ad positions s. Formally w_{ts}^k and v_{ts}^k are defined as follows:

$$\begin{aligned} w_{ts}^k &= p_{ts}^k \alpha^k(s) X^k(t), \\ v_{ts}^k &= (V^k - p_{ts}^k)\alpha^k(s) X^k(t), \ \ \forall\, s, t, k. \end{aligned} \tag{4.3}$$

Here V^k denote the expected value-per-click for keyword k, $X^k(t)$ denote the number of user queries for keyword k at time period t, and $\alpha^k(s)$ denote the click-through rate (CTR) of position s (the ratio between total user clicks on the ad at s-th slot and the total number of impressions). $p_{ts}^k = b_{t,s+1}^k$, i.e. the cost-per-click is equal to the next highest bid. Since most auctioneers enforce a policy that each advertiser can have *at most one* ad appear on each keyword results page, this corresponds to that at most one item can be taken from N_t^k. If we treat each N_t^k as an item set, then this consists of an instance of MCKP where the knapsack capacity C is equal to the advertiser's total budget B.

4.6 Experimental Results

We run two sets of experiments. The first set evaluates the performance of the algorithm ALG-S-MCKP on synthetic datasets when items are generated from various probability distributions. The second set of experiments uses a real dataset we manually collected from the (now defunct) Yahoo!/Overture view bids webpage.

[2] For the popular rank-by-revenue scheme, the charge of the i-th position is p_{i+1} times a coefficient which is related to the quality scores of ads at both i and $i+1$. We can easily incorporate this case into our consideration. For simplicity, we assume all ads are equally good for this work.

4.6.1 Experiments With Known Distributions

In this set of experiments, we generate items with weights and values drawn independently from one of the following distributions: Uniform with support between 1 and 10, Normal with mean 10, and Exponential with mean 10. The normal distribution was truncated at zero to avoid negative weights and values. The number of items per set received each time period is 5. We express the budget (capacity) as a fraction of the mean weight of an item times the total number of time periods, i.e., $C = \lambda \times n \times \text{mean}(w)$, where n is the total number of time periods, mean(w) is the mean of the weight of a random item. For example, mean(w) is 5.5 when item weights are uniformly distributed between 1 and 10, $B = 5.5 \times \lambda \times n$. The value of λ indicates whether the budget is large compared to the expected overall spending if you pick an item randomly from each set. We tested the algorithms on various problem instances with various budget levels and report the following λ values in our experiments: 0.05, 0.2, 0.5, 0.9, 1.1.

We evaluate the performance of the algorithm based on the ratio of the value obtained by the algorithm and an upper bound on the optimal value of the solution to MCKP. The upper bound of the MCKP can be calculated by solving the fractional version of the MCKP as described in section 4.4.1.

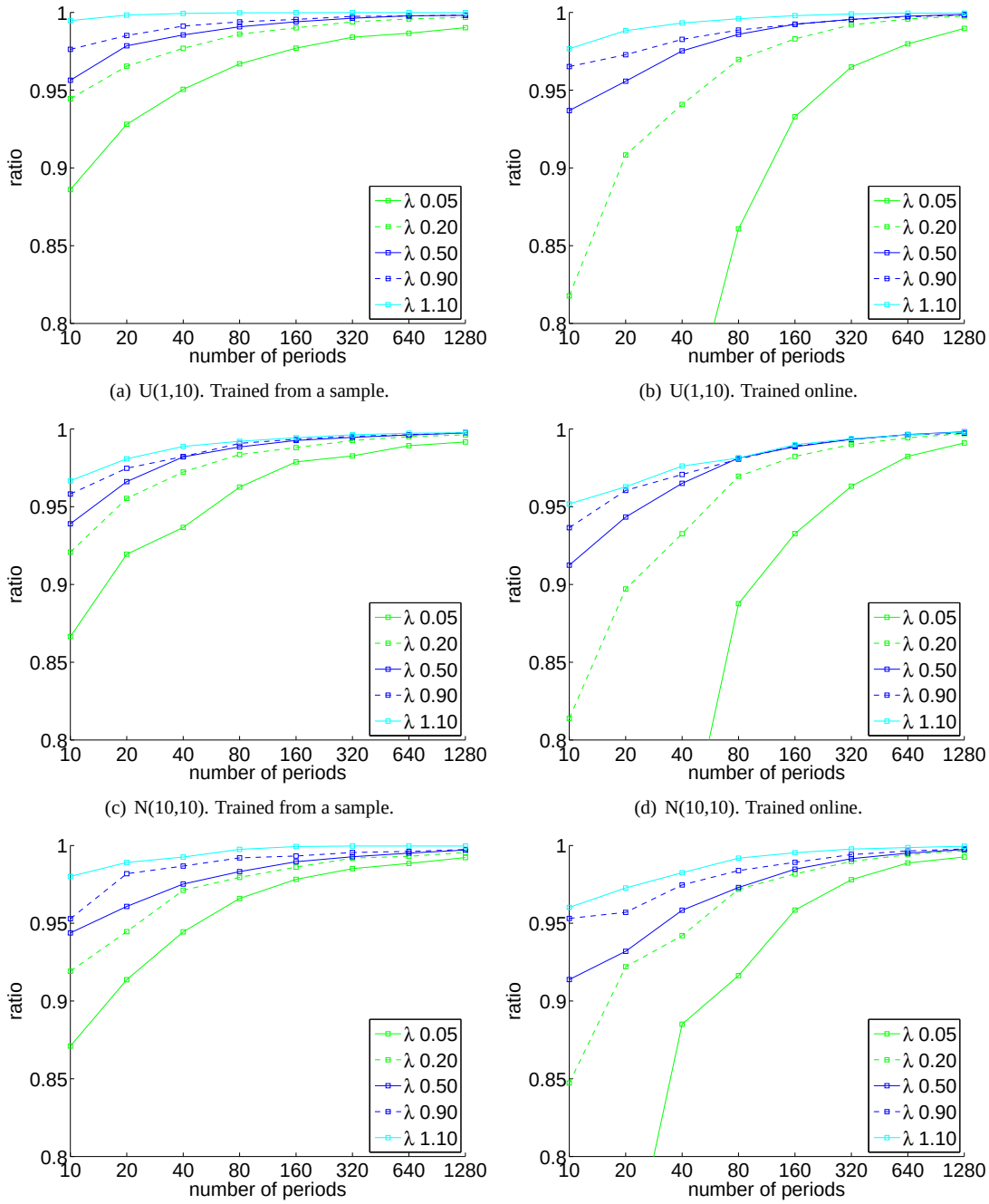
We test two versions of the algorithm. The first one uses the threshold function generated based on 80 sampled item sets and does not update the threshold function. The other version does not generate the threshold function beforehand but uses the items received each time period to generate/update the threshold function. We refer to the first version as offline-training and to the second one as online-training. The results are in Figure 4.6. Each data point on the graphs is the average of 100 runs on random problem instances.

Graphs for the algorithm with offline-training are on the left and graphs for the algorithm with online-training are on the right. For both versions of the algorithm, the performance is almost always within 10% of the optimal when the number of periods is 20 or higher and approaches the optimal as the number of periods increases. The algorithm with online-training performs worse than the algorithm with offline-training when the number of periods is small. During the first few periods the threshold function of the online algorithm is very unreliable (based on very few samples) and the decisions made during early periods are prone to mistakes. These mistakes are especially costly when the budget is small because taking a wrong item may exhaust a significant part of the budget. However as the number of periods increases the performance of the online-training algorithm comes close to that of offline-training. The results are similar across different distributions.

4.6.2 Experiments With Real Bidding Data

For these experiments we use a keyword bidding dataset that was manually collected by Chakrabarty *et al.* [23] from the Overture view bids website over the period of two weeks. The bids are for the single keyword "auto insurance." There are totally 1842 distinct time periods. For each time period, the data contains the bidding prices for all top-40 positions. Each time period is about one minute. The data do not contain any information about the number of clicks.

For one experiment, following [23], we assume that $\alpha(s) = 1 - s/40$ is the CTR of position s and $X(t) = 1$ over all time periods. We use the algorithm that trains online without any pre-training. Each time



(a) U(1,10). Trained from a sample.

(b) U(1,10). Trained online.

(c) N(10,10). Trained from a sample.

(d) N(10,10). Trained online.

(e) E(mean=10). Trained from a sample.

(f) E(mean=10). Trained online.

Figure 4.6: Performance of ALG-S-MCKP with various item distributions

period, the current set of bids is converted into the set of items as described in Eq. (4.3), for three distinct values $V = 8, 10, 12$. The algorithm's solution is within 99% of the offline optimum for all three values of V.

Note that the algorithm performs better than the one described in [23], which achieves a ratio between 90%-95%. Even though both algorithms are based on using some kind of efficiency threshold for item selection, there is an important distinction: our algorithm's bids remain relatively constant over time close to the value which is optimal in expectation, while the algorithm from [23] bids higher at the beginning and gradually reduces its bid, and at some time it starts to increase its bids again based on a sniping heuristic.

For another experiment, we use an exponential function to model the CTR each periods: $\alpha(s) = 0.9^s$, while the number of clicks is $X(t) = 1$ as in the first experiment. The expected value-per-click in this experiment is set to the single value $V = 12$. The performance of the algorithm is shown in Figure 4.7. The algorithm finds a solution that is within 6% of the optimal for all budget levels and number of periods. Unlike the experiments with known distribution, the ratio is not monotonically increasing with the number of periods. Increased number of periods does not result in improved performance because the prices in the data set are not identically distributed each time period.

For instance, the ratio for $\lambda = 1.1$ drops from .97 to .95 when the number of periods increases from 160 to 320. We plot prices of the slots that are targeted by the algorithm during the first 320 periods to explain why this happens. Figure 4.8(a) shows that prices for slots 11-17 are constant during the first 140 periods. Around period 140, prices for slots 11-16 increase and stay at a higher level for most of the periods through 320. Intuitively, it is optimal to target a higher slot in order to get more clicks before period 140, i.e., when the prices are lower. We plot optimal bids and algorithm's bids in Figure 4.8(b) to illustrate this.

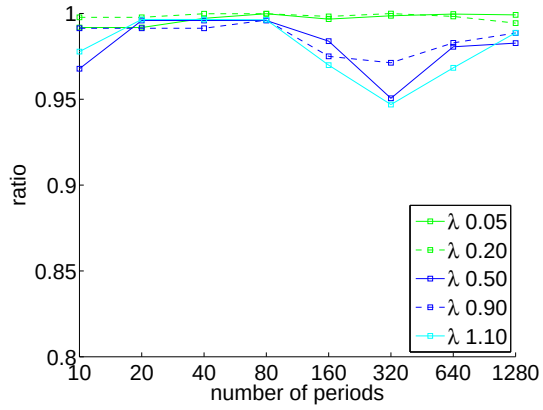
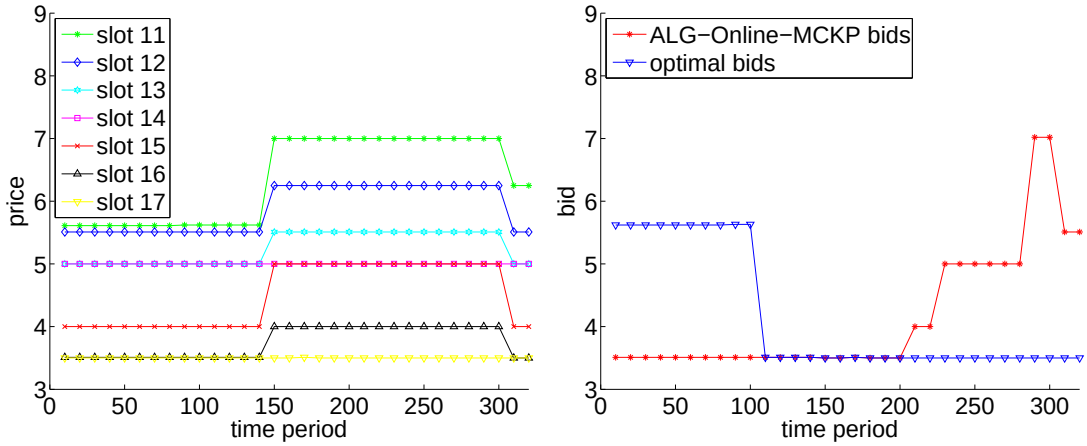


Figure 4.7: Performance of ALG-S-MCKP on keyword bidding data set with 1842 periods, exponential CTR, and fixed number of clicks $X(t) = 1$.

As expected, most optimal bids are higher during the first 140 periods in order to buy a lot of clicks when they are cheap. Optimal bids win slot 11 for \$5.7 for the first 100 periods and are lowered to \$3.5 to win slot 17 in periods 100-300. In contrast, the algorithm has no way of knowing that the prices are going to increase, and its bid (\$3.5 to win slot 16) during the first 200 periods is at the level that is optimal assuming the prices will remain the same as in the first 140 periods. After period 140, the algorithm starts seeing higher prices and spending less as its bid of \$3.5 results in position 17 instead 16 (i.e., fewer clicks). At period 200, the algorithm has seen enough periods with higher prices and has underspent enough to decide to raise its bids.



(a) Prices for Slots 11-17

(b) Bids for $\lambda = 1.1$ and $V = 12$

Figure 4.8: Keyword bidding data set with 320 periods and exponential CTR.

Unfortunately, clicks are more expensive then and the algorithm performs worse than the optimal. However the algorithm's performance is still around 95% of the optimal.

The final experiment is a variant of experiment 2 where the number of clicks each time period is uniformly distributed between 1 and 20. The results are in Figure 4.9. The added randomness from the number of clicks makes the threshold function less reliable when it is based on few samples. This results in mistakes in early periods which are relatively more costly when the total number of periods is small. This is evidenced in a relatively bad performance when the number of periods is 10 and 20. However additional uncertainty about the number of clicks does not prevent the algorithm from perfoming well on problems with more periods. The algorithm is within 7% of the optimal when the number of period is 40 or more.

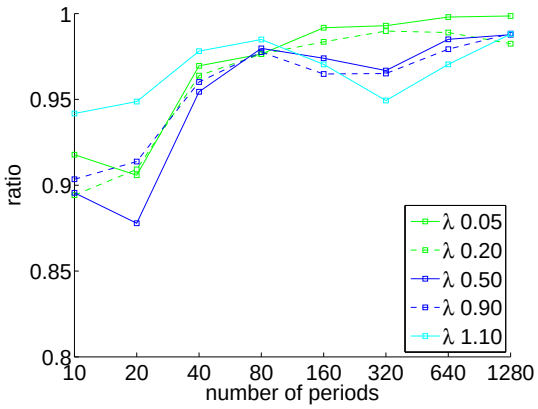


Figure 4.9: Performance of ALG-S-MCKP on keyword bidding data set with 1842 periods, exponential CTR, and random number of clicks.

4.7 Conclusion

We propose an algorithm for S-MCKP that combines an approximation to MCKP with an algorithm for Online-KP. Our algorithm is based on the idea that MCKP can be converted to KP, which can then be solved using the well-known greedy KP approximation, and the solution to KP can be mapped back to the solution to MCKP. Our main contribution is the algorithm that accomplishes this for the online version of MCKP. At the heart of the algorithm is the threshold function for KP which filters out the items of insufficient efficiency. We adapt the process of computing the threshold function to the online setting where no information about the items needs to be available a priori. Instead, the threshold function is updated online. We apply the algorithm to problem instances generated with different distributions and to a real data set. In all of our experiments the performance is within 10% of the offline optimum, and it approaches the offline optimum when the number of periods is sufficiently large.

For future work, one direction is to model trends in data explicitly. The current algorithm assumes that there are no trends in data, as items are identically distributed across time periods. However it works relatively well in the presence of trends because the threshold function is updated over time. Another direction that we are currently pursuing function over time. Another direction that we are currently pursuing is building/deploying a keyword bidding agent based on the above algorithm. Yet another direction is to prove theoretical guarantees about the performance of the S-MCKP algorithm with general assumptions on the distribution of items.

Chapter 5

Production Scheduling in TAC SCM[1]

5.1 Introduction

For many years, researchers in artificial intelligence and operations research have studied difficult problems in combinatorial optimization such as supply chain management, vehicle routing, and airline crew scheduling. The majority of this research has focused on solving deterministic problems; however, in many applications there is inherent uncertainty that is not captured by deterministic models. Moreover, in many optimization settings, stochastic information about the shape of the future is readily available in the form of probabilistic models built from historical data. Recently, computational and technological advances have made it feasible to reason about this stochasticity.

In general, two strategies are adopted when dealing with uncertainty in combinatorial optimization. The first strategy treats problems in an online fashion: algorithms are forced to make decisions in the face of incomplete information and accommodate new information only as it becomes available. Such algorithms typically fall into two categories. The first category includes simple, greedy heuristics for handling new information as it unfolds. The second category includes algorithms for finding optimal solutions given what is known; then, as new information becomes available, the solutions are re-optimized with the new information, respecting any unalterable prior decisions.

The second strategy for dealing with uncertainty in combinatorial optimization focuses on determining ahead of time a solution that is optimal in the expected sense. Stochastic programming is one example of this strategy [14]. At a high level, stochastic programming considers problems in two stages. Decisions must be made in the first stage before pertinent information about the second stage is revealed, but the objectives in the second stage are dependent on the first stage decisions. Given stochastic information available about the second stage outcomes, the goal is to find the first stage decisions that maximize the profits of the first stage plus the expected profits of the second stage.

One computational bottleneck to solving stochastic programs is the calculation of expected profits in the second stage. This calculation typically involves enumerating all possible outcomes of the second stage (also

[1] Based on [28].

known as *scenarios*). In many problems there are combinatorially many scenarios, making it prohibitively expensive to calculate the expected profits of the second stage. One common means of approximating this calculation is the so-called *expected value method* [14].

Unfortunately, the expected value method ignores large portions of the given stochastic information. It has been shown that using additional stochastic information can improve the quality of solutions in dynamic vehicle routing [12, 13], packet scheduling [24], and elevator dispatching [81]. In these sample applications, stochastic information is exploited in widely different ways; however, the unifying theme seen throughout this research is that there are considerable advantages to taking account of stochastic information.

Shapiro, *et al.* [4, 62, 97] recently proposed an alternative approximation technique called *Sample Average Approximation* (SAA), which reduces the number of scenarios. They suggest using only a subset of the scenarios, randomly sampled according to the scenario distribution, to represent the full scenario space. An important theoretical justification for this method is that as the sample size increases, the solution converges to an optimal solution in the expected sense. Indeed, the convergence rate is exponentially fast.

In this paper, we combine these two strategies to handling uncertainty: *we use techniques for finding optimal solutions in the expected sense to solve combinatorial problems in an online setting.* The problem we address is the scheduling component of the Trading Agent Competition in Supply Chain Management (TAC SCM) problem a classic combinatorial optimization problem with uncertainty (see `www.sics.se/tac/`). We formulate this problem as a stochastic program and we use SAA in an online setting to find today's optimal schedule, given predictive information about the future. This optimization procedure forms the heart of BOTTICELLI, one of the finalists in the TAC SCM 2003 competition.

We describe two sets of experiments, using either one or two days of information about the future. In our two day experiments (using one day of information about the future), we show that SAA outperforms the *expected value method*, which solves a deterministic variant of the problem assuming all stochastic inputs have deterministic values equal to their expected values. In our three day experiments (using two days of information about the future), we show that SAA with lookahead outperforms greedy SAA. Our approach generalizes to N days of lookahead, and since our problem setting is one of online optimization, the benefits of two day lookahead accrue rapidly. It remains to show that our approach improves the performance of agents in TAC SCM.

5.2 TAC SCM

In recent years, the amount of time available for making complex managerial decisions in commercial settings has decreased dramatically [70]. Assuming this trend continues, it will become increasingly more important to develop tools that automate the decision making process. TAC SCM is a simulated market economy in which software agents tackle complex optimization problems in dynamic supply chain management.

In TAC SCM, six software agents compete in a simulated sector of a market economy, specifically the personal computer (PC) manufacturing sector. Each agent can manufacture 16 different types of computers, characterized by different *stock keeping units* (SKUs). Building each SKU requires a different combination of components, of which there are 10 different types. These components are acquired from a common pool of

suppliers at costs that vary as a function of demand. After assembly, each agent can sell its PCs to a common pool of customers by underbidding the other agents. The agents are ranked based on their profits over 220 days, each of which lasts 15 seconds.

The TAC SCM simulation proceeds as follows: Each day, customers send a set of *requests for quotes* (RFQs) to the agents. Each RFQ contains a SKU, a quantity, a due date, a penalty rate, and a reserve price—the highest price the customer is willing to pay. Each agent sends an offer to each customer for each RFQ, representing the price at which it is willing to satisfy that RFQ.[2] After the customer receives all its offers, it selects the agent with the lowest-priced offer and awards that agent with an *order*. Either: the winning agent delivers the entire order by its due date, in which case it is paid in full; it delivers the entire order within five days of its due date, in which case it is paid the amount of its offer less a penalty based on the number of late days; or, it cannot deliver the entire order within five days of its due date, in which case the order is canceled, no revenues are accrued, and the maximum penalty is incurred.

In the meantime, the agents themselves are sending RFQs to suppliers, requesting a specific quantity of a component to arrive on a particular day. The suppliers respond to these requests the next day with either partial or full offers, indicating the price per unit at which the RFQ can be satisfied. If an agent receives a partial offer, the supplier cannot deliver the requested quantity of the component on the day on which it was requested, but it can deliver a lesser quantity on that day. Full offers either have a delivery date on the day requested, or a delivery date later than the one requested, in which case they are often accompanied by partial offers. Among these offers, an agent can choose to accept at most one, in which case agent and supplier enter into a contract agreeing that the agent will be charged for the components upon their arrival.

At the end of each day, each agent converts components acquired from suppliers into SKUs according to a production schedule it generates for its finite-capacity, single-machine factory. In addition, it reports a delivery schedule assigning the SKUs in its inventory to customer orders.

Each simulated day represents a decision cycle for an agent, during which time the agents must solve the following four problems: bidding, scheduling, procurement, and allocation.

- The *bidding* problem determines the offer price for each RFQ.
- The *scheduling* problem determines the production schedule for each day.
- The *procurement* problem determines which components to buy from suppliers.
- The *allocation* problem matches SKUs in inventory to orders.

These four problems are highly interconnected. Indeed, an optimal solution to the scheduling problem yields an optimal solution to the procurement and allocation problems, since revenue maximization and cost minimization, which guide scheduling decisions, depend on how inventory is allocated to orders and on what supplies are procured. Moreover, an optimal solution to the bidding problem yields an optimal solution to the scheduling problem, since bidding decisions depend on manufacturing capacity constraints: too few winning bids lead to missed revenue opportunities, while too many winning bids lead to late penalties.

All of these problems involve decisions that must be made today with only stochastic information about tomorrow. *TAC SCM agents face combinatorial, online optimization problems with inherent uncertainty.*

[2] An agent may select to not send an offer for an RFQ, but this is equivalent to issuing an offer price above the reserve price.

Due to an artifact in the design of TAC SCM 2003—namely, negligible component prices on day 1, which led to the placement of essentially infinite orders on day 1 for supplies to be delivered throughout the game—our agent, BOTTICELLI, focused on solving only three of these four problems: bidding, scheduling, and allocation. Here, we present our solution to the scheduling (and allocation) problem.

To solve the scheduling problem, an agent must choose a schedule that accounts for outstanding orders, possible orders (among the existing RFQs), future RFQs, outstanding component orders, future component costs, and current component and SKU inventory. Which orders will materialize among the existing RFQs, the shape of future RFQs, possible supplier defaults on outstanding component orders, and future component costs are all stochastic elements of the scheduling problem. The remainder of this paper is concerned with solving simplified yet representative formulations of this scheduling problem.

5.3 Simple Scheduling

The *simple scheduling* problem is defined as follows: *Given* a set of orders, characterized by SKU, quantity, due date, penalty, and price; initial component inventory; a procurement schedule for components on each day; initial product inventory; one machine of finite capacity; the number of production cycles required to produce each product; and product specifications, namely which components comprise which products, ***find*** a production schedule that optimizes profit, or revenue less costs. This problem is dubbed *simple* (relative to the TAC SCM scheduling problem), since revenues and costs are deterministic.

5.3.1 Integer Linear Programming Solution

In this section, we present an integer linear programming (ILP) solution to the simple scheduling problem.

Constants and Variables

Let O denote the set of orders. Each order $i \in O$ is characterized by the following information: SKU s_i, price p_i, quantity q_i, due date d_i, penalty ρ_i, and reserve price r_i. Let D denote the maximum due date among all orders and E denote the maximum acceptable overdue date. Let l range over days $1, 2, \ldots, D + E \equiv N$. Now, ρ_{il} is the penalty incurred if order i is filled on day l. For notational simplicity, we let π_{il} represent the profit for filling order i on day l. The constant π_{il} is formally defined as follows:

$$\pi_{il} = \begin{cases} p_i & l \leq d_i \\ p_i - \rho_{il} & d_i < l \leq d_i + E \\ -\rho_{i(d_i+E)} & l > d_i + E \end{cases}$$

Let a_k denote the quantity of component k in initial inventory and b_j denote the quantity of SKU j in initial inventory. According to the procurement schedule, let a_{kl} denote the quantity of component k to be delivered on day l. Let C denote the capacity of the machine in terms of production cycles, and let c_j denote the number of production cycles required to manufacture SKU j. If component k is part of SKU j, then $e_{jk} = 1$; otherwise, $e_{jk} = 0$. Similarly, if order i is for SKU j, then $f_{ij} = 1$; otherwise, $f_{ij} = 0$.

In addition to these constants, our solution relies on the following variables:

- $z_{il} \in \{0, 1\}$, which indicates whether or not order i is filled on day l. (For notational simplicity, we allow an order i to be filled on days $l > d_i + E$; however, conceptually, orders filled on day $d_i + E + 1$ are in fact unfilled orders and are treated as such in the formalization.)[3]
- $y_{jl} \in \mathbb{Z}_+$, which denotes the amount of SKU j scheduled for production on day l.

Objective Function and Constraints

The simple scheduling problem can be stated as follows:

$$\max \sum_{i \in O} \sum_{l=1}^{d_i+E+1} z_{il} \pi_{il} \tag{5.1}$$

subject to:

$$\sum_{l=1}^{d_i+E+1} z_{il} = 1, \quad \forall i \tag{5.2}$$

$$\sum_{\{i \mid f_{ij}=1\}} \sum_{l=1}^{L} q_i z_{il} \leq b_j + \sum_{l=1}^{n-1} y_{jl}, \quad \forall j, n \in \{1, \ldots, N\}, L = \min(n, d_i + E) \tag{5.3}$$

$$\sum_{\{j \mid e_{jk}=1\}} \sum_{l=1}^{n} y_{jl} \leq a_k + \sum_{l=1}^{n-1} a_{kl} \quad \forall k, n \in \{1, \ldots, N\} \tag{5.4}$$

$$\sum_{j} c_j y_{jl} \leq C, \quad \forall l \tag{5.5}$$

$$z_{il} \in \{0, 1\}, \quad \forall i, l \tag{5.6}$$

$$y_{jl} \in \mathbb{Z}_+, \quad \forall j, l \tag{5.7}$$

- **Equation 5.1** is the objective function, namely to maximize profits, where the quantity $\sum_{l=1}^{d_i+E+1} z_{il}$ indicates whether or not order i is filled on day l.
- **Equation 5.2** states that an order must be filled exactly once. (Every order is either filled on some day $l \leq d_i + E$, or it is filled on day $d_i + E + 1$, meaning it is not filled.)
- **Equation 5.3** states that the total quantity of SKU j associated with all orders filled by day n does not exceed the total inventory produced by day $n - 1$ plus any initial inventory of SKU j.
- **Equation 5.4** expresses the resource constraints on components: The total quantity of component k used through day n must not exceed the total quantity of component k ordered by day $n - 1$ from all suppliers plus any initial inventory of component k.
- **Equation 5.5** enforces the capacity constraint: The total number of production cycles used to produce all SKU types on day l must not exceed the machine's daily capacity C.

[3] We introduce these variables for ease of exposition of the ILP, but in our implementation $z_{id_i+E+1} = 1 - \left(\sum_{l=1}^{d_i+E} z_{il}\right)$.

5.4 Probabilistic Scheduling

The simple scheduling problem is extended to a *probabilistic* scheduling problem with an additional input. We add a set of RFQs, characterized like orders, but with an additional parameter α_i that represents i's likelihood of becoming an order. (For orders $i \in O$, $\alpha_i = 1$.) Implicitly, this formulation of the problem assumes that all likelihoods are independent. In probabilistic scheduling, the objective is to find a production schedule that maximizes *expected* profit. The following stochastic program (SP) achieves this objective.

5.4.1 Stochastic Programming Solution

Given a set of orders, and a set of RFQs *today* only a fraction of which will be realized *tomorrow*, we seek to produce an "optimal" set of SKUs *today* s.t. *tomorrow*'s profits will be maximized. More specifically, we seek to produce some set of SKUs, trading off production of those SKUs that can be used to fill the most profitable RFQs with those that can be used to fill those RFQs that are most likely to become orders.

Let $w_i \in \{0, 1\}$ indicate whether or not order i is filled on day 1,[4] and let $v_j \in \mathbb{Z}_+$ denotes the amount of SKU j scheduled for production on day 1. Let Ω_m denote the set of RFQs that are realized in the mth scenario (σ_m). Now let $z_{ilm} \in \{0, 1\}$ indicate whether or not order $i \in O$ or RFQ $i \in \Omega_m$ is filled on day l in scenario m, and let $y_{jlm} \in \mathbb{Z}_+$ denote the amount of SKU j scheduled for production on day l in scenario m.

$$\max \sum_{i \in O} w_i \pi_{i1} + \sum_m P(\sigma_m) \left[\sum_{i \in O \cup \Omega_m} \sum_{l=2}^{d_i+E+1} z_{ilm} \pi_{il} \right] \tag{5.8}$$

subject to:

$$w_i + \sum_{l=2}^{d_i+E+1} z_{ilm} = 1, \quad \forall m, i \in O \cup \Omega_m \tag{5.9}$$

Stage 1:

$$\sum_{\{i \mid f_{ij}=1\}} q_i w_i \leq b_j, \quad \forall j \tag{5.10}$$

$$\sum_{\{j \mid e_{jk}=1\}} v_j \leq a_k, \quad \forall k \tag{5.11}$$

$$\sum_j c_j v_j \leq C \tag{5.12}$$

$$w_i \in \{0, 1\}, \quad \forall i \tag{5.13}$$

$$v_j \in \mathbb{Z}_+, \quad \forall j \tag{5.14}$$

[4]Note: $w_i = 0$ for all RFQs i.

Stage 2:

$$\sum_{\{i \mid f_{ij}=1\}} q_i \left(w_i + \sum_{l=2}^{L} z_{ilm} \right) \leq b_j + v_j + \sum_{l=2}^{n-1} y_{jlm},$$
$$\forall j, m, n \in \{2, \ldots, N\}, L = \min(n, d_i + E) \tag{5.15}$$

$$\sum_{\{j \mid e_{jk}=1\}} \left(v_j + \sum_{l=2}^{n} y_{jlm} \right) \leq a_k + \sum_{l=1}^{n-1} a_{klm} \tag{5.16}$$
$$\forall k, m, n \in \{2, \ldots, N\}$$

$$\sum_j c_j y_{jlm} \leq C, \quad \forall m, l \in \{2, \ldots, N\} \tag{5.17}$$

$$z_{ilm} \in \{0, 1\}, \quad \forall i, l, m \tag{5.18}$$

$$y_{jlm} \in \mathbb{Z}_+, \quad \forall j, l, m \tag{5.19}$$

- **Equation 5.8** is the objective function, namely to maximize profits, where the quantity $\sum_{l=1}^{d_i+E+1} z_{ilm}$ indicates whether or not order i is filled on day l in scenario Ω_m.
- **Equation 5.9** states that orders and RFQs must be filled exactly once. In particular, an order can be filled on day 1, or it can be filled at some later date in the scenarios. An RFQ can only be filled at some later date in the scenarios.
- **Equations 5.10, 5.11, and 5.12** pertain to the w_i and v_j variables: i.e., production and the allocation of inventory to orders on day 1. The total quantity of SKU j allocated to orders on day 1 cannot exceed the initial inventory of SKU j. The total quantity of component k used in production on day 1 cannot exceed the initial inventory of component k. The total number of production cycles used to produce all SKU types on day 1 must not exceed the machine's daily capacity C.

The final set of constraints pertains to production and the allocation of inventory to orders and RFQs on days $2, \ldots, N$ in the various scenarios.

- **Equation 5.15** expresses the resource constraints on inventory. In all scenarios, the total quantity of SKU j associated with all orders filled by day n (either on day 1 or on some later date in the scenarios) cannot exceed the total inventory produced by day $n-1$ plus and the initial inventory.
- **Equation 5.16** expresses the resource constraints on components. In all scenarios, the total quantity of component k used through day n cannot exceed the total quantity of component k procured by day $n-1$ and any initial inventory.
- **Equation 5.17** enforces the capacity constraint. In all scenarios, the total number of production cycles used to produce all SKU types on day l cannot exceed the machine's daily capacity C.

Lastly, let us the compute the probabilities of the various scenarios σ_m. Viewing σ_m as a bit vector, $\sigma_{mi} \in \{0, 1\}$ indicates whether or not RFQ i is realized in scenario m. Now, the probability of the mth scenario is given by:

$$P(\sigma_m) = \prod_i \alpha_i^{\sigma_{mi}} (1 - \alpha_i)^{1-\sigma_{mi}} \tag{5.20}$$

5.5 Approximation Algorithms

Algorithm	Output
Expected-Value (EV)	ILP with expected profits and quantities
Expected-Profit (EP)	ILP with expected profits
Expected-Quantity (EQ)	ILP with expected quantities
SAA-Greedy (SAAG)	SP using only current RFQs
SAA-Average (SAAA)	SP using average future RFQs
SAA-Sampling (SAAS)	SP using sampled future RFQs
Not-in-time Production (NTP)	ILP ignoring RFQs

Table 5.1: Approximation Algorithms

Table 5.1 summarizes the seven ILPs that are featured in our experiments. All ILPs were solved using CPLEX version 7.5, terminating with the first feasible solution. The first three algorithms approximate the SP solution by solving variants of the simple scheduling problem. The *expected-value* algorithm solves the simple scheduling problem using expected profits and expected quantities. Expected profits are computed by multiplying π_{il} by α_i in Equation 5.1.[5] Expected quantities are computed by multiplying q_i by α_i in Equation 5.3. The *expected-profit* (respectively, *expected-quantity*) algorithm solves the simple scheduling problem using only expected profits (respectively, quantities).

The next three algorithms approximate the SP solution using *sample average approximation* (SAA), whereby they sample a subset of the scenario space according to its distribution, and optimize only with respect to those samples. *SAA-greedy* samples scenarios only consisting of one day's worth of actual RFQs. This algorithm makes no attempt to reason about future RFQs. *SAA-average* samples scenarios consisting of N days' worth of RFQs, assuming that all future RFQs look like an average RFQ. *SAA-sampling* samples scenarios consisting of N days' worth of RFQs; but, SAA-sampling generates sample future RFQs from an RFQ distribution, rather than assume that all future RFQs look like an average RFQ.[6]

Finally, *not-in-time* production ignores stochastic information entirely. It only schedules orders—i.e., RFQs that have been realized. As its name suggests, this strategy can often lead to late penalties, since production does not begin until one day after RFQs are received.

5.6 Empirical Results

The experiments we performed modeled the scheduling problem faced by an agent competing in the TAC SCM game, and similar problems faced by dynamic supply chain management systems. These experiments tested two hypotheses: (i) algorithms that utilize more stochastic information outperform those that do not; and (ii) algorithms that look ahead into the future outperform greedy algorithms.

[5] Recall that $\alpha_i = 1$ for all orders i.

[6] All of our SAA algorithms sampled 30 scenarios, since larger sample sizes showed no significant advantage.

(a) Ranges of Distributions

Parameter	Range
SKU	[1, 16]
Price	[$1600, $2300]
Quantity	[1, 20]
Penalty	[5%, 15%] of Price
Probability	[0, 1]

(b) Description of Metrics

Metric	Description
P	mean profit per order
C	% cycles used to fill orders
P/C	mean profit per cycle
EVPI	expected value of perfect information
VSI	value of stochastic information

Table 5.2: Distribution Ranges and Metric Descriptions

Each N day trial of our experiments proceeded as follows. On each day, the algorithms received randomly generated RFQs drawn from a distribution similar to that of the TAC SCM game specification. Specifically, 200 RFQs were generated at random, with parameters uniformly distributed in the ranges shown in Table 5.1(a). Unlike in TAC SCM, (i) each RFQ was assigned some probability of becoming an order, and (ii) each RFQ was due on the day *immediately* after it was issued. Given a set of outstanding orders and new RFQs, the algorithms generated schedules and produced inventory. (Note: Based on the above distributions, 100 RFQs were expected to be converted to orders each day. Since each RFQ takes an average of 55 production cycles, the production of all orders requires more than the 2000 cycle capacity granted—the numbers 55 and 2000 are based on the TAC SCM game specification.) The next day after some more of the RFQs became orders, the algorithms allocated (i.e., delivered) product inventory resulting from production on previous days to current orders. Each order that was filled yielded some revenue, orders filled after their due dates also yielded revenue but incurred a penalty, and orders that were not filled at all incurred the maximum penalty of 5 times the RFQ's daily penalty value.

In our experiments, we made the following simplifying assumptions: no initial orders, no initial product inventory, and infinite component inventory. The third simplification, as alluded to earlier, is an artifact of the TAC SCM game design in 2003. These first two simplifications were designed to isolate the effects being tested by avoiding unnecessary complexity.

5.6.1 Metrics

Table 5.1(b) describes the metrics computed during each trial that were used to evaluate the approximation algorithms. The first metric, mean profit per order, was the primary measure of an algorithm's performance. Secondly, the percentage of cycles used to fill orders, indicates that percentage of the 2000 available cycles which were used by an algorithm to produce PCs that were actually sold. Perhaps more informatively, the next metric, profit per cycle, measures how well the algorithms filled more profitable, rather than less profitable, orders.

The *expected value of perfect information* is calculated by subtracting the mean profit an algorithm achieved from the maximum possible, which it could have achieved had it had perfect foresight: i.e., if it knew exactly which RFQs would become orders. The maximum possible mean profit was calculated using the ILP described in Section 5.3 after the fact. The *value of stochastic information* is the difference between an algorithm's mean profit and that of the Expected Value algorithm. This metric describes how much an

algorithm gained or lost by utilizing stochastic information beyond simple expected values.

Algorithm	P	C	P/C	EVPI	VSI
SAA-Greedy	**$1,207**	**95.7%**	**$63.59**	**78,550**	**48,105**
Expected Profit	$448	93.9%	$24.40	154,450	-27,800
Expected Quantity	$-1,251	81.5%	$-77.71	324,390	-197,740
Expected Value	$726	90.8%	$47.65	126,650	0

Table 5.3: Two Day Experiments: Metric Values

Algorithm	P	C	P/C	EVPI	VSI
SAA-Greedy	$1,567	98.6%	$79.5	87,350	34,310
SAA-Sample	$1,620	98.1%	$82.6	76,810	44,848
SAA-Average	**$1,635**	98.1%	**$83.4**	**73,670**	**47,990**
Expected Profit	$1,294	**98.7%**	$65.69	142,000	-20,200
Expected Quantity	$593	95.8%	$31.34	282,100	-160,300
Expected Value	$1,395	96.8%	$72.34	121,800	0
Not-In-Time	$-4,557	49.3%	$-462.53	1,312,100	-1,190,400

Table 5.4: Three Day Experiments: Metric Values

5.6.2 Two Day Experiments

In the two day experiments, algorithms received one set of RFQs and scheduled one day of production. The resulting product inventory was allocated to orders on the day 2. Any orders that were not filled incurred the maximum late penalty.

These experiments tested the ability of the algorithms to schedule production relying on only stochastic information. After day 1, there was no opportunity for production; thus, there was no opportunity to satisfy any orders that could not be filled from day 1's production.

The metric values described in Table 5.1(b) for the two day experiments are shown in Table 5.3. In addition, the 95% confidence intervals of each algorithm's mean profit are shown in Figure 5.1(a).[7] Since SAAA and SAAS are identical to SAAG when there is only one day of production; these lookahead algorithms were excluded from the two day experiments. The NTP algorithm does not have a chance to schedule any production at all in these experiments, and was also excluded.

In the two day experiments, SAAG outperformed the other algorithms under all metrics. Figure 5.1(a) shows with 95% confidence that SAAG was significantly better in terms of mean profit. In second place (in terms of mean profit) was the EV algorithm. Despite selling fewer cycles, the EV algorithm outperformed the EP algorithm in terms of mean profit. These results suggest that the EV algorithm was filling fewer orders, but choosing some of the more profitable ones (as evidenced by the P/C values). The EP algorithm uses a more risky technique when scheduling production, since it attempts to fill every RFQ in its entirety. When it chose to fill an RFQ with a large expected profit and the RFQ did not become an order, at best the products

[7] These confidence were intervals calculated using the bootstrap percentile-t method (see, for example, [32])

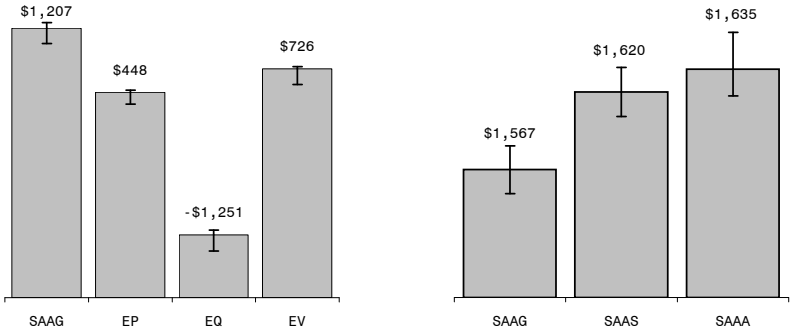
that were made could be given to other less profitable RFQs. The EV algorithm subverts this problem by only producing RFQs in proportion to their likelihood of becoming an order. EQ performed relatively poorly on all computed metrics because it was scheduling production to fill expected quantities of what were sometimes unlikely realizations.

By design, SAAG uses more stochastic information than the other algorithms; therefore, the results from these experiments confirm our hypothesis that using more stochastic information leads to better performance.

5.6.3 Three Day Experiments

In the three day experiments, algorithms received two sets of RFQs and scheduled two days of production. The optimal solution to this scheduling problem in the expected sense is described by the stochastic program in Section 5.4.1, when the set of scenarios includes all combinations of realizations over both days of RFQs.

More specifically, the three day experiments proceeded as follows: The first set of RFQs, all of which were due on day 2, was received on day 1. The algorithms then scheduled production and built up their product inventory. On day 2, a subset of day 1's RFQs was selected at random to become orders. In addition, a second set of RFQs was received, all of which were due on day 3. The algorithms again scheduled production and built up their product inventory. In addition, any orders due on day 2 that could be filled were shipped, and revenues were recorded. On day 3, a subset of day 2's RFQs was selected at random to become orders. At this point, any outstanding orders due on day 2 that could be filled were shipped, and revenues were recorded, less late penalties; any orders due on day 3 that could be fulfilled were shipped, and revenues were recorded; and, penalties were recorded for any unfilled orders.



(a) Two Day Experiments (b) Three Day Experiments

Figure 5.1: Mean Profits with 95% Confidence Intervals

The purpose of these experiments was (i) to show that using more stochastic information is at least as useful across multiple days as it was in the 2 day experiment, and (ii) to test the ability of the algorithms that made use of stochastic information to plan for the future given stochastic knowledge about the shape of future RFQs. To an extent, these experiments also tested the ability of the algorithms to recover from possible misuse of stochastic information in the two day experiments; but, such affects would be better uncovered by multiple day experiments.

As shown in Table 5.6.1, the stochastic programs outperformed all of the other schedulers in all but one calculated metric. Once again, these results confirm our hypothesis that using more stochastic information leads to better performance. All other results are consistent with results from the two day experiments.

Figure 5.1(b) shows that the stochastic algorithms that rely on forecasts about future RFQs outperformed SAAG. Unlike the greedy algorithm, the algorithms with lookahead use stochastic information about future RFQs to make scheduling decisions. These experiments confirmed our second hypothesis: using more stochastic information about the future also leads to better performance.

The improvement seen was the result of day 1's with low-priced RFQs. The greedy algorithm was forced to cope with these poor RFQs because it did not utilize any stochastic information about the future. On the other hand, the algorithms with lookahead chose to schedule production that filled predicted future RFQs with higher prices, rather than waste production cycles on RFQs with low prices. SAAS and SAAA perform comparably in these experiments (see Figure 5.1(b)) because the RFQs sampled by SAAS were drawn from a uniform distribution, and thus tended to reflect mean RFQs. Given a non-uniform distribution, we conjecture that the sampling algorithm would make better use of stochastic information about future RFQs than an algorithm that relies on only the mean.

This paper only considers experiments of 2 and 3 days in duration. In future work, we plan to assess the performance of these algorithms over many days, such as the typical 220 days of the TAC SCM game.

5.7 Conclusion

The research problems in the Trading Agent Competitions are typically approached using clever heuristics and optimization techniques. With a few notable exceptions [42, 98], these methods have tended to ignore some of the information that characterizes the uncertainty in the problems. This paper suggests that it is possible to substantially improve the performance of algorithms by incorporating stochastic information about the future. More importantly, this paper shows that the precise methodology for including stochastic information is an important indicator of an algorithm's performance. Indeed, the scheduling decisions determined by a stochastic programming approach that aims to characterize all of the uncertainly outperforms methods that make no use or partial use of uncertainty.

Research on dynamic supply chain management can proceed in a number of future directions. By itself, the probabilistic scheduling approach makes worthwhile decisions given a fixed distribution for future RFQs. However, in the bidding problem, there is an opportunity to alter the distributions of the RFQs that could become orders, namely by raising or lowering bids. In BOTTICELLI, the probabilistic scheduling algorithm serves as a critical component for evaluating various bidding strategies. The algorithms presented here rely heavily on stochastic programming to handle uncertainty; however, there are other techniques, such as consensus [12] and POMDPs [57], for coping with uncertainty, which may prove useful when the full TAC SCM problem (including procurement) is considered.

Chapter 6

ILP Bidding in TAC SCM[1]

The chapter describes the architecture of Brown University's agent, BOTTICELLI, a finalist in the 2003 Trading Agent Competition in Supply Chain Management (TAC SCM). In TAC SCM, a simulated computer manufacturing scenario, BOTTICELLI competes with other agents to win customer orders and negotiates with suppliers to procure the components necessary to complete its orders.

In this chapter, two subproblems that dictate BOTTICELLI's behavior are formalized: bidding and scheduling. Mathematical programming approaches are applied in attempt to solve these problems optimally. In addition, greedy methods that yield useful approximations are described. Test results compare the performance and computational efficiency of these alternative techniques.

6.1 Introduction

A supply chain is a network of autonomous entities, or agents, engaged in *procurement* of raw materials, *manufacturing*—converting raw materials into finished products—and *distribution* of finished products. The Trading Agent Competition in Supply Chain Management (TAC SCM) is a simulated computer manufacturing scenario in which software agents tackle complex problems in supply chain management. This chapter describes the structure of Brown University's agent BOTTICELLI, a finalist in TAC SCM 2003.

TAC SCM agents face uncertainty about the future, but they must make decisions before the uncertainty is resolved: e.g., agents must procure raw materials and manufacture finished products before customer orders arrive. BOTTICELLI handles the uncertainty in manufacturing and distribution using stochastic programming techniques (see Benisch *et al.* [11]). Here, we focus on our approach to the *bidding* problem: find an optimal set of bids to place on customer RFQs, balancing the tradeoff between maximizing profits—by placing high bids—and maximizing the likelihood of winning multiple customer orders—by placing low bids.

This chapter is organized as follows. In Section 6.2, we give an overview of TAC SCM. Next we describe

[1] Published as [10].

the architecture of our agent, BOTTICELLI. This architecture emphasizes three problems—bidding, production scheduling, and delivery scheduling. Section 6.4 describes a heuristic approach to these problems: greedy scheduling and bidding via hill-climbing. Section 6.5 details solutions that approximate optimal stochastic programming solutions. Section 6.6 presents experimental results.

6.2 TAC SCM

In TAC SCM, six software agents compete in a simulated sector of a market economy, specifically the personal computer (PC) manufacturing sector. Each agent can manufacture 16 different types of computers, characterized by different *stock keeping units* (SKUs). Building each SKU requires a different combination of components, of which there are 10 different types. These components are acquired from a common pool of suppliers at costs that vary as a function of demand. After assembly, each agent can sell its PCs to a common pool of customers by underbidding the other agents. The agents are ranked based on their profits over 220 days, each of which lasts 15 seconds.

Each day in the TAC SCM simulation customers send a set of *requests for quotes* (RFQs) to the agents. Each RFQ contains a SKU, a quantity, a due date, a penalty rate, and a reserve price—the highest price the customer is willing to pay. Each agent sends an *offer* to each customer for each RFQ, representing the price at which it is willing to satisfy that RFQ. After the customer receives all its offers, it selects the agent with the lowest-priced offer and awards that agent with an *order*. Either: the winning agent delivers the entire order by its due date, in which case it is paid in full; it delivers the entire order within five days of its due date, in which case it is paid the amount of its offer less a penalty based on the number of late days; or, it cannot deliver the entire order within five days of its due date, in which case the order is canceled, no revenues are accrued, and the maximum penalty is incurred.

Meanwhile, the agents themselves are sending RFQs to suppliers, requesting a specific quantity of a component to arrive on a particular day. The suppliers respond to these requests the next day with either partial or full offers, indicating the price per unit at which the RFQ can be satisfied. If an agent receives a partial offer, the supplier cannot deliver the requested quantity of the component on the day on which it was requested, but it can deliver a lesser quantity on that day. Full offers either have a delivery date on the day requested, or a delivery date later than the one requested, in which case they are often accompanied by partial offers. Among these offers, an agent can choose to accept at most one, in which case agent and supplier enter into a contract agreeing that the agent will be charged for the components upon their arrival.

At the end of each day, each agent converts the components it acquired from suppliers into SKUs according to a production schedule it generates for its factory. It also reports a delivery schedule assigning the SKUs in its inventory to customer orders.

6.3 Agent Architecture

Each simulated TAC day represents a decision cycle for an agent, during which time the agents must solve four problems: procurement, bidding, production scheduling, and delivery scheduling. The *procurement*

TAC SCM Decision Problem
Objective:
Maximize Expected Profits
Inputs:
Product Pricing Model
Component Cost Model
Set of Supplier Offers
Set of Customer RFQs
Set of Customer Orders
Procurement Schedule
Component Inventory
Product Inventory
Outputs:
Procurement Schedule: set of Supplier RFQs and Orders
Bidding Policy: map from Customer RFQs to Prices
Production Schedule: map from Cycles to SKUs
Delivery Schedule: map from SKUs to Customer Orders

Figure 6.1: TAC SCM Decision Problem

problem involves communicating with suppliers via RFQs, and selecting supplier offers to accept among those which are received in response to these RFQs. The *bidding* problem is to decide how to assign offer prices to each customer RFQ. The *production scheduling* problem is to decide how many of each SKU to assemble each day. The *delivery scheduling* problem is to decide which orders to ship to which customers, using product inventory. The objective in all of these problems is to maximize *expected* profits, given some probabilistic model that captures the uncertainty in the game. A high-level description of the TAC SCM decision problem is presented in Figure 6.1.

An artifact in the design of TAC SCM 2003 (namely, negligible component prices on day 1), resulted in us placing little emphasis on *procurement*. Rather, we focused on the development of solutions to the *bidding*, *scheduling*, and *delivery* problems. High-level descriptions of the three problems are given in Figures 6.4, 6.6, and 6.7. These three problems are highly interconnected. Indeed, an optimal solution to the production scheduling problem yields an optimal solution to the delivery scheduling problem, since ultimately revenues depend on which orders are successfully delivered to their respective customers. Moreover, an optimal solution to the bidding problem yields an optimal solution to both scheduling problems, since bidding decisions depend on manufacturing and distribution constraints: too few winning bids lead to missed revenue opportunities; too many winning bids lead to late penalties.

The architecture of BOTTICELLI was designed with these relationships in mind, and thus the bidding module envelops the scheduling module, which in turn envelops the delivery module as shown in Figure 6.2. Once a bidding policy is determined by the bidding module, the scheduling module finds a production schedule, and the delivery module ships products to customers.

The flow of information through the agent is as follows: Each day the modeling module receives information about other agents' actions on the previous day as well as information about the offers the bidding module submitted and the orders that resulted from those offers. The modeling module uses this information to update its models and passes an updated model to the bidding module. The bidding module uses the new model to produce an offer for each of the day's RFQs. The offer prices are determined with the aid of the

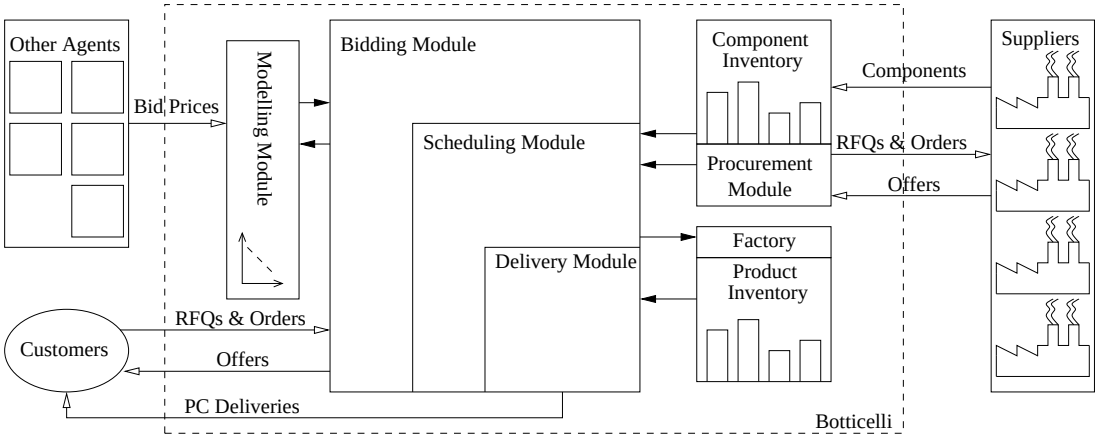


Figure 6.2: Botticelli: A Modular Design

scheduling module. When invoked, the scheduling module learns from the procurement module the quantity of each component that is expected to be in inventory on any particular day. It then determines how to allocate machine cycles to make products for existing orders and likely future orders. The scheduling module relies on the delivery module to determine how to allocate product inventory to existing orders and likely future orders. After the bidding, scheduling, and delivery modules finalize their decisions, the procurement module sends to suppliers RFQs for additional components and orders based on the current offers.

6.4 Bidding: A Hill-Climbing Approach

In the preliminary rounds, BOTTICELLI relied on a hill-climbing bidder, which successively adjusts bid prices according to the results of a scheduler. At a high-level, the bidder is initialized with some set of bid prices; given these prices, a production and delivery schedule is computed; and, based on the results of the scheduler, bid prices are tweaked. The goal of this hill-climbing algorithm is to fill our production schedule, which we assume is positively correlated with maximizing expected profits. TacTex utilizes a similar solution to the bidding problem [87].

In a preprocessing step, we schedule only orders, no offers. As long as all orders can be scheduled for delivery, we proceed with the hill-climbing bidder.

It is crucial to our approach that the scheduler make use of the probabilities of winning each offer: the scheduler must schedule offers based on *expected quantities*.

We initialize bids to prices at which, according to our pricing model, we will win every RFQ with certainty. At these initial prices, if the scheduler cannot fit every order and RFQ into the schedule, then those RFQs which are not deemed profitable enough to include in the schedule at their current prices form a natural set of RFQs for which to raise prices. Indeed, we increase the prices of these RFQs, thereby decreasing their winning probabilities. In the next iteration, the scheduler, which schedules according to expected quantities, may be able to schedule these RFQs for production. Prices are increased (i.e., probabilities are decreased)

until all RFQs can be scheduled. This process is guaranteed to converge, since the winning probability of RFQs above their reserve prices is zero, yielding a corresponding expected quantity of zero.

6.4.1 Scheduling: A Greedy Approach

Our greedy scheduler is passed both orders and offers, which it sorts as follows:

- Orders are placed before offers, since offers might not be won.
  - Orders are sorted by ascending due date, then by descending penalty.
  - Offers are sorted by descending profit per cycle (p_ι/c_j, where $j = f_\iota$), then by ascending due date, and lastly by descending penalty.

Note that offers are not sorted by probability. We experimented with this ordering, but profitability proved to be more important than probability.

Let o be the current order or offer and let j be o's SKU. The greedy scheduler addresses the orders and offers in sorted order as follows:

1. Schedule backwards from o's due date. That is, start by scheduling as much as possible of SKU j on the day o is due. If more needs to be scheduled, then schedule as much as possible on each successively earlier day until either no more is needed or the current day is reached.
2. If more of SKU j still needs to be produced, allocate as much as possible from product inventory.
3. If still more of SKU j is needed, schedule forwards from o's due date until either all of order o is scheduled or the cancellation date is reached.
4. If the cancellation date is reached, then cancel all scheduled production of SKU j for o.

Note that if o's due date is the current day, then there is no time to produce any more of SKU j. In this case, the greedy scheduler begins at step 2.

6.5 Bidding: A Mathematical Programming Approach

We now formulate mathematical programs to solve the delivery scheduling, production scheduling, and bidding problems. Our proposed solution to the bidding problem relies on a solution to the production scheduling problem. Similarly, our proposed solution to the production scheduling problem relies on a solution to the delivery scheduling problem. In our exposition, we distinguish between *simple* optimization problems, in which there is no uncertainty, and *stochastic* optimization problems. We present optimal solutions to the simple subproblems before describing our approximate solutions to the stochastic optimization problems. All solutions are described in terms of the variables, constants, and abbreviations listed in Figure 6.3.

6.5.1 Simple Scheduling

In simple scheduling, there is no uncertainty because there are no customer RFQs. The sole purpose of simple scheduling is to fill standing customer orders.

Variables

x_ι	bidding policy: bid price for RFQ ι
y_{jl}	production schedule: quantity of SKU j scheduled for production on day l
z_{il}	delivery schedule: 1 if order i is delivered on day l; 0 otherwise
$z'_{\iota l}$	delivery schedule: 1 if RFQ ι is delivered on day l; 0 otherwise

Constants in the Objective Functions

R	number of RFQs
O	number of orders
D	latest due date among all orders
E	number of days before a late order is canceled
q_i	quantity of order i
d_i	due date of order i
p_i	revenue for delivering order i on or before $d_i + E$
ρ_{il}	penalty incurred if order i is delivered on day l
q'_ι	quantity of RFQ ι
d'_ι	due date of RFQ ι
$\rho'_{\iota l}$	penalty incurred if RFQ ι is delivered on day l

Abbreviations in the Objective Functions

π_{il} revenue earned by delivering order i on day l

$$\pi_{il} = \begin{cases} q_i p_i & l \le d_i \\ q_i p_i - \rho_{il} & d_i < l \le d_i + E \end{cases}$$

$\pi'_{\iota l}(p)$ revenue earned by delivering RFQ ι on day l at price p

$$\pi'_{\iota l}(p) = \begin{cases} q'_\iota p & l \le d'_\iota \\ q'_\iota p - \rho'_{\iota l} & d'_\iota < l \le d'_\iota + E \end{cases}$$

ζ_i 1 if order i is not delivered at all; 0 otherwise

$$\zeta_i = 1 - \sum_{l=1}^{d_i+E} z_{il}$$

ζ'_ι 1 if RFQ ι is not delivered at all; 0 otherwise

$$\zeta'_\iota = 1 - \sum_{l=2}^{d'_\iota+E} z'_{\iota l}$$

Additional Constants in the Constraints

a_k	components of type k in initial inventory
α_{kl}	components of type k delivered on day $l+1$
b_j	number of PCs of SKU j in initial inventory
c_j	cycles expended to produce one PC of SKU j
e_{jk}	1 if SKU j contains component type k; 0 otherwise
f_{ij}	1 if order i is for SKU type j; 0 otherwise
$f'_{\iota j}$	1 if RFQ ι is for SKU type j; 0 otherwise
C_d	number of cycles on day d

Probabilistic Pricing Model

$P_\iota(p)$	probability of winning RFQ ι at price p

Figure 6.3: Mathematical Programming Variables, Constants, and Abbreviations

Simple Delivery Scheduling

Delivery Scheduling
Inputs:
Production Schedule
Set of Customer Orders
Product Inventory
Output:
Delivery Schedule: map from SKUs to Customer Orders

Figure 6.4: Simple Delivery Scheduling

The (simple) delivery scheduling problem is one of allocating SKUs in product inventory to customer orders, given a $(D+E)$-day production schedule (see Figure 6.4). The following integer program solves the delivery scheduling problem. (Note: y_{jl} is constant in this formulation.)

$$\max_{z} \sum_{i=1}^{O} \left[\left(\sum_{l=1}^{d_i+E} z_{il}\pi_{il} \right) - \zeta_i \rho_{i(d_i+E)} \right] \tag{6.1}$$

subject to:

$$z_{il} \in \{0,1\}, \quad \forall i,l \tag{6.2}$$

$$\sum_{l=1}^{d_i+E} z_{il} \leq 1, \quad \forall i \tag{6.3}$$

$$\sum_{l=1}^{t} \sum_{\{i \mid f_{ij}=1\}} q_i z_{il} \leq b_j + \sum_{l=1}^{t-1} y_{jl}, \quad \forall j, t = 1, \ldots, D+E$$

The objective (Equation 6.1) is to maximize revenue and minimize penalties; but, no order can be delivered more than once (Equation 6.3); and, the total quantity of SKU j associated with orders delivered by day t cannot exceed the total inventory of SKU j produced by day $t-1$ plus any initial inventory (Equation 6.4).

Simple Production Scheduling

Simple Production Scheduling
Inputs:
Set of Customer Orders
Procurement Schedule
Component Inventory
Product Inventory
Outputs:
Production Schedule: map from Cycles to SKUs
Delivery Schedule: map from SKUs to Customer Orders

Figure 6.5: Simple Production Scheduling

The simple production scheduling problem is one of allocating cycles to SKUs, given a set of customer orders, initial component and product inventory, and a $(D+E)$-day procurement schedule (see Figure 6.5).

The following integer program solves the simple production scheduling problem.

$$\max_{y,z} \sum_{i=1}^{O} \left[\left(\sum_{l=1}^{d_i+E} z_{il}\pi_{il} \right) - \zeta_i \rho_{i(d_i+E)} \right] \tag{6.5}$$

subject to Constraints 6.2, 6.3, 6.4, and the following:

$$y_{jl} \in \mathbb{Z}_{\geq 0}, \quad \forall j, l \tag{6.6}$$

$$\sum_{l=1}^{t} \sum_{\{j \mid e_{jk}=1\}} y_{jl} \leq a_k + \sum_{l=1}^{t-1} \alpha_{kl} \quad \forall k, t \tag{6.7}$$

$$\sum_{j} c_j y_{jl} \leq C_l, \quad \forall l \tag{6.8}$$

As in delivery scheduling, the objective (Equation 6.5) is to maximize revenue and minimize penalties; and, all of the delivery scheduling constraints still apply. In addition, Equation 6.7 expresses the resource constraint on components: The total quantity of component k used through day t cannot exceed the total quantity of component k delivered by day $t-1$ plus any initial inventory of component k. Equation 6.8 enforces the capacity constraint: The total number of production cycles used to produce all SKU types on day l cannot exceed the machine's capacity on day l.

6.5.2 Stochastic Scheduling and Bidding

Allowing for customer RFQs as well as standing customer orders introduces uncertainty into the scheduling problems. This uncertainty also arises in the bidding problem, where its exact nature depends on bids.

To handle this uncertainty, the scheduling problem can be formulated as a stochastic program (see Benisch *et. al.* [11]). In solving this stochastic program, we show that the sample average approximation method (SAA) [62] outperforms the expected value method [14] on this problem. Nonetheless, we relied on the expected value method in our implementation of BOTTICELLI-2003 because it readily applies to the bidding problem, whereas SAA does not.

Expected Production Scheduling

In the production scheduling problem, the objective is to allocate cycles to SKUs not only to fill existing customer orders, but in addition to fill offers—customer RFQs equipped with bid prices—which may or may not become orders. We model this uncertainty by associating probabilities with offers: offers with low bid prices are assigned high probabilities, whereas offers with high bid prices are assigned low probabilities.

The following integer program approximates the production scheduling problem. (Note: x_ι is constant in this formulation.)

$$\max_{y,z,z'} \sum_{i=1}^{O} \left[\left(\sum_{l=1}^{d_i+E} z_{il}\pi_{il} \right) - \zeta_i \rho_{i(d_i+E)} \right] + \sum_{\iota=1}^{R} P_\iota(x_\iota) \left[\left(\sum_{l=2}^{d'_\iota+E} z'_{\iota l}\pi'_{\iota l}(x_\iota) \right) - \zeta'_\iota \rho'_{\iota(d'_\iota+E)} \right] \tag{6.9}$$

Production Scheduling
Additional Inputs:
Bidding Policy
Product Pricing Model
Set of Customer Orders
Procurement Schedule
Component Inventory
Product Inventory
Outputs:
Production Schedule: map from Cycles to SKUs
Delivery Schedule: map from SKUs to Customer Orders

Figure 6.6: Production Scheduling

subject to Constraints 6.2, 6.3, 6.6, 6.7, 6.8, and the following:

$$z'_{\iota l} \in \{0,1\}, \quad \forall \iota, l \tag{6.10}$$

$$\sum_{l=2}^{d'_\iota+E} z'_{\iota l} \leq 1, \quad \forall \iota \tag{6.11}$$

$$\sum_{l=1}^{t} \sum_{\{i \mid f_{ij}=1\}} q_i z_{il} + \sum_{l=2}^{t} \sum_{\{\iota \mid f'_{\iota j}=1\}} P_\iota(x_\iota) q'_\iota z'_{\iota l} \leq b_j + \sum_{l=1}^{t-1} y_{jl}, \quad \forall j, t \tag{6.12}$$

The objective function (Equation 6.9) maximizes profits from orders and *expected* profits from RFQs. Equation 6.11 states that no RFQ can be delivered more than once. Equation 6.12, which considers RFQs as well as orders, replaces Equation 6.4. Note the use of *expected* quantity $P_\iota(x_\iota)q'_\iota$ regarding RFQs in Equation 6.12.

Bidding

Bidding
Inputs:
Product Pricing Model
Set of Customer RFQs
Set of Customer Orders
Procurement Schedule
Component Inventory
Product Inventory
Outputs:
Bidding Policy: map from Customer RFQs to Prices
Production Schedule: map from Cycles to SKUs
Delivery Schedule: map from SKUs to Customer Orders

Figure 6.7: Bidding

The objective in the bidding problem is to find an optimal bidding policy. We solve this problem by extending the solution to the production scheduling problem based on the expected value method. In production scheduling, all RFQs are equipped with bid prices, which are constants. In the bidding problem, the prices at which to offer to fill RFQs are variables. Once prices become variables rather than constants, the objective function is no longer linear. (In fact, in our formulation, it is not even quadratic.) Thus, in our implementation we discretize prices to recover a linear formulation:

M number of prices

$\mu_{m\iota}$ price of RFQ ι with index m

$z'_{\iota lm}$ 1 if RFQ ι is delivered on day l at price indexed by m;
0 otherwise

Now the following integer program approximates the bidding problem.

$$\max_{y,z,z'} \sum_{i=1}^{O} \left[\left(\sum_{l=1}^{d_i+E} z_{il}\pi_{il} \right) - \zeta_i \rho_{i(d_i+E)} \right] + \sum_{m=1}^{M} \sum_{\iota=1}^{R} P_\iota(\mu_{m\iota}) \left[\left(\sum_{l=2}^{d'_\iota+E} z'_{\iota lm}\pi'_{\iota l}(\mu_{m\iota}) \right) - \zeta'_\iota \rho'_{\iota(d'_\iota+E)} \right] \tag{6.13}$$

subject to Constraints 6.2, 6.3, 6.6, 6.7, 6.8, and the following:

$$z'_{\iota lm} \in \{0,1\}, \quad \forall \iota, l, m \tag{6.14}$$

$$\sum_{m=1}^{M} \sum_{l=2}^{d'_\iota+E} z'_{\iota lm} \leq 1, \quad \forall \iota \tag{6.15}$$

$$\sum_{l=1}^{t} \sum_{\{i \mid f_{ij}=1\}} q_i z_{il} + \sum_{m=1}^{M} \sum_{l=2}^{t} \sum_{\{\iota \mid f'_{\iota j}=1\}} P_\iota(z'_{\iota lm}) q'_\iota z'_{\iota lm} \leq b_j + \sum_{l=1}^{t-1} y_{jl}, \quad \forall j, t \tag{6.16}$$

6.6 Experiments

In this section we report on experiments designed to compare the performance of three bidding algorithms, one based on our mathematical programming solution, one hill-climbing bidder, and one blend of the two.

6.6.1 Heuristics

To bid optimally in TAC SCM, an agent would have to optimize with respect to (i) each of the other agent's individual strategies; and (ii) all possible future scenarios, weighted by their likelihoods. Agent modeling is not feasible in TAC, since the behavior of individual agents is observed only by the server. Thus, we collapse all agents' behaviors into one model (see Section 6.6.1). Furthermore, since it would be intractable to consider all possible futures, we rely on an heuristic that stands in the place of simulating the future—specifically, future orders (see Section 6.6.1).

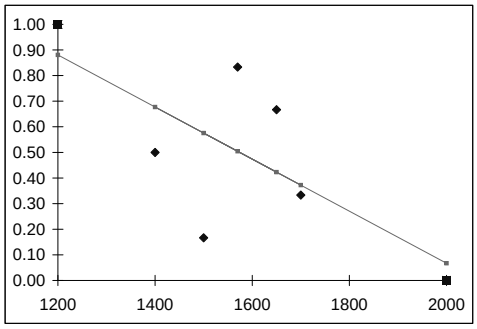


Figure 6.8: Price vs. Probability for a SKU. Diamonds are data points from offers sent during the past d days. Squares are data points from the previous day's minimum and maximum prices.

Modeling

The modeling module predicts the relationship between the bid price of an offer and the probability of winning that offer. There are several sources of information available for modeling this relationship. In our implementation, we utilize two: the first is a report provided by the server each day with the maximum and minimum closing prices for each SKU on the previous day; the second is BOTTICELLI's past offer prices and the orders that resulted. Our modeling module is concerned only with price and probability relationships for each SKU, rather than for each RFQ, since maximum and minimum prices are SKU-specific.

For each SKU, the modeler plots the minimum and maximum prices from the previous day at probabilities 1 and 0, respectively. Intuitively, low prices are likely to be winning prices, while high prices are likely to be losing prices. In addition, for each of the previous d days, BOTTICELLI's average offer prices are plotted against the ratio of the number of offers won to the number of offers issued. In total, our modeling module is provided with $d + 2$ points, which it fits using a least-squares linear regression. This linear *cdf* (price vs. probability graph) is adopted as the model that is input to the bidding module. (See Figure 6.8.)

By experimentation, we found the value of 5 to be a good choice for d. This value allowed BOTTICELLI to be responsive enough to the changes in price that often accompanied another agent receiving a shipment of supplies, but prevented any drastic overreactions. We experimented with using additional information to create more stable models, such as providing weights for points based on the number of offers they represented, and maintaining the average of the d previous days' minimum and maximum prices. These methods, however, did not respond well to price jumps that were typical of the 2003 TAC SCM competition.

The Triangle Method

In scheduling for multiple days of production, BOTTICELLI's scheduling module relies on the following heuristic: do not use all cycles on all days, but rather save production cycles on future days for future RFQs (see Figure 6.9). This heuristic is motivated by two assumptions. First, higher revenues can be earned by winning the same quantity of RFQs over multiple days, rather than winning a large quantity of RFQs on one day, since, according to our model, an agent can only win a large quantity on one day by bidding low prices. Second, the "character" RFQs of tomorrow will not differ significantly from the RFQs of today, since all

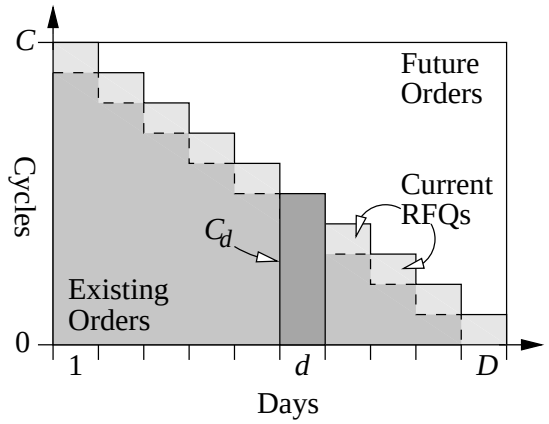


Figure 6.9: On day d, only $C_d = \frac{C((D-d)+1)}{D}$ cycles are made available to the scheduler. Cycles outside the triangle are reserved for future orders. D is the number of days of production in the schedule. C is the daily production capacity.

RFQs are drawn from a uniform distribution. In particular, future RFQs will not be significantly better or worse than today's RFQs in terms of quantity, due date, etc. If, however, a change in the *number* of RFQs is predicted,[2] BOTTICELLI saves more (less) cycles if the number of RFQs is predicted to increase (decrease), since prices tend to increase (decrease) accordingly.

6.6.2 Experimental Setup

Our experiments consisted of 20 day trials, which proceeded as follows: On each day, the algorithms received a randomly generated set of RFQs drawn from a distribution similar to that of the TAC SCM game specification. Specifically, 300 RFQs were generated at random, with parameters uniformly distributed in the ranges shown in Table 6.1. Given these RFQs, the algorithms produced a bidding policy as well as production and delivery schedules for $D = 10$ days. Based on its bid prices and the corresponding probabilities, an algorithm won orders for some of the RFQs. The algorithms were then responsible for producing and delivering the products for these RFQs before their due dates or they were penalized according to the rate specified in the RFQ. The tests continued in this fashion for 20 days; this number was long enough to allow the algorithms to distinguish themselves, but short enough to allow several hundred iterations.

In order to mitigate any start effects in our experiments, the algorithms were initialized with the same set of 150 customer orders (thus, the first day looked like all other days). We made the simplifying assumption that all algorithms had an infinite component inventory, which, as alluded to earlier, is an artifact of the TAC SCM game design in 2003. Finally, to isolate the effects of the bidding algorithms, we relied on models that could perfectly predict the likelihood of winning any RFQ at any price.

Parameter	Range
Price	[$1600, $2300]
Quantity	[1, 20]
SKU	[1, 16]
Penalty	[5%, 15%] of Price

Table 6.1: Uniform Distribution Ranges

	Profits	Deliveries	$\overline{\text{Price}}$	Penalty
HG	$7,781,100	6,847	$1,193	$505,610
HE	$8,019,600	7,286	$1,095	$285,950
EB	$9,600,900	7,860	$1,222	$113,660

Table 6.2: Experimental Results

6.6.3 Experimental Results

The algorithms included in our experiments were the hill-climbing bidder with a greedy production scheduler (HG), the hill-climbing bidder with an expected production scheduler (HE), and the expected bidder (EB), which used its own schedule for production. Both of the hill-climbing bidders utilized a greedy scheduler to evaluate candidate bidding policies, as such policies needed to be evaluated hundreds of times. (The greedy scheduler completed in .01 seconds, on average, whereas the expected production scheduler completed in 1 second.) However, we allowed one of the hill-climbing bidders to utilize an expected scheduler for production scheduling only. Our hypothesis was that the expected bidder with built in scheduling and delivery modules would out perform all of the others, as it would be capable of performing a more global optimization while solving the bidding problem.

Relevant statistics of the 500 trials are given in Table 6.2. The mean profits of each algorithm over 20 days with 95% confidence intervals are shown in Table 6.3. These results validated our hypothesis. The expected bidder outperformed both instances of the the hill-climbing bidders in every category in Table 6.2. The 95% confidence intervals shown in Table 6.3 reveal that the difference in profits is statistically significant. The addition of the expected scheduling algorithm to the hill-climbing bidder helped it to achieve fewer penalties by improving the production scheduling solutions; however, the lack of a global bidding strategy still crippled its abilities. It seems that the expected bidder produced results that were close to optimal, since its total penalty was relatively small and it managed to utilize its factory at nearly full capacity each day without wasting many finished products.

6.7 Conclusion

Following Kiekintveld [60], we identify three key issues in supply chain management that are modeled in TAC SCM: (i) *uncertainty* about the future; (ii) *strategic behavior* among the entities; and (iii) *dynamism*:

[2] BOTTICELLI predicts the level of demand using a particle filter. Details of this approach are beyond the scope of this chapter.

	Low	High
HG	$7,756k	$7,804k
HE	$7,988k	$8,050k
EB	$9,585k	$9,617k

Table 6.3: Mean Profits—95% Confidence Intervals

i.e., the temporal nature of the chain. BOTTICELLI adequately handles uncertainty (in the bidding problem), but makes simplifying assumptions to handle the strategic and dynamic components of the game. Rather than model each competing agent's strategic behavior individually, we collapse all agents' behaviors into one model, and optimize with respect to this model. In essence, we use decision-theoretic optimization techniques to approximate solutions to game-theoretic problems. Dynamic optimization models and techniques (e.g. MDPs) might be applicable in TAC SCM, but to optimize with respect to all possible future scenarios is clearly intractable. Instead, we rely on an heuristic we call the *triangle method*, by which we save production cycles on future days for future RFQs, particularly if prices are predicted to increase. In future versions of BOTTICELLI, we plan to build more powerful models of the agents' strategic environment, and to incorporate more sophisticated methods of dynamic optimization, particularly in the procurement problem.

Chapter 7

Greedy Bidding in TAC SCM[1]

We present a fast and effective bidding strategy for the Trading Agent Competition in Supply Chain Management (TAC SCM). In TAC SCM, manufacturers compete to procure computer parts from suppliers (the procurement problem), and then sell assembled computers to customers in reverse auctions (the bidding problem). This chapter is concerned only with bidding, in which an agent must decide how many computers to sell and at what prices to sell them. We propose a greedy solution, Marginal Bidding, inspired by the Equimarginal Principle, which states that revenue is maximized among possible uses of a resource when the return on the last unit of the resource is the same across all areas of use. We show experimentally that certain variations of Marginal Bidding can compute bids faster than our ILP solution, which enables Marginal Bidders to consider future demand as well as current demand, and hence achieve greater revenues when knowledge of the future is valuable.

7.1 Introduction

A supply chain is a network of autonomous entities engaged in procurement of raw materials, manufacturing—converting raw materials into finished products—and distribution of finished products. The Trading Agent Competition in Supply Chain Management (TAC SCM) is a simulated computer manufacturing scenario in which software agents operate a dynamic supply chain [5].

In this chapter, we study the TAC SCM bidding problem, where the goal is to choose prices at which to offer to sell computers to customers today, balancing the tradeoff between maximizing revenue per order—by placing high bids—and maximizing the quantity of customer orders won—by placing low bids, within the constraints of current and future component availability and production capacity. Ideally, these decisions should be made taking into account future demand: in a bull market it may be advantageous to reserve today's production capacity for future, more profitable demand; in a bear market it may be preferable to bid more aggressively early on, claiming a larger share of current demand to be fulfilled with products manufactured

[1] Based on [48].

in the future.

To model these tradeoffs, we formulate bidding in TAC SCM as an N-day recursive, stochastic, integer linear program (ILP). The mathematical program is recursive because the agent faces the same decision variables day after day, namely the prices at which to set its current bids so as to maximize the sum of its current revenue and its expected future revenue. It is stochastic in part because of the inherent uncertainty in future demand. However, we also use stochasticity to model the game-theoretic dynamics of bidding in a reverse auction, thereby reducing what is truly a game-theoretic problem to a decision-theoretic one. This is an important simplifying assumption that permeates our study.

A tractable approximation of 1-day bidding, called *expected bidding*, was considered in Benisch *et al.* [10]. We revisit this problem here, and show that it reduces to a generalization of the classic knapsack problem, the so-called *nonlinear knapsack problem* (NLK). Then, inspired by the Equimarginal Principle—which states that revenue is maximized among possible uses of a resource when the return on the last unit of the resource is the same across all areas of use—we propose a greedy solution to the expected bidding problem, which we call Marginal Bidding. We advocate for Marginal Bidding in this chapter because it scales linearly with the number of days, and can hence more easily solve an N-day extension of expected bidding than traditional ILP solutions.

To analyze the performance of various heuristics designed for TAC SCM, we built a simulator that generates decision-theoretic simplifications of the game-theoretic problems TAC SCM agents face, such as bidding. Using our simulator, we compared the performance of several variants of Marginal Bidding with an ILP solution. We show that certain variations of Marginal Bidding can compute bids faster than our ILP solution; hence, incorporating a Marginal Bidder into a TAC SCM agent would allow for more time to be spent on other decision problems (e.g., procurement). Moreover, this speedup enables Marginal Bidders to reason about future demand as well as current demand, and hence achieve greater revenues when knowledge of the future is valuable. While the gains to be realized by reasoning about future demand in TAC SCM appear modest, we demonstrate that more substantial gains can be realized under more volatile or seasonal conditions that generate more extreme market swings.

This chapter is organized as follows. We begin by describing the Equimarginal Principle of marginal utility theory, originally posited by Gossen in the mid 1800's. We note that this principle can be applied to solve the nonlinear knapsack problem. Then, we present a discretization technique coupled with a greedy algorithm, which we prove approximately solve the NLK. (Technically, we prove that our approach yields a Fully Polynomial Time Approximation Scheme—a FPTAS—for the NLK.) Next, we formalize TAC SCM bidding as an N-day recursive stochastic program, and argue that expected bidding, a 1-day deterministic approximation, can be reduced to solving an instance of the NLK. Then, we present Marginal Bidding, a heuristic for solving an N-day extension of expected bidding that incorporates the aforementioned discretization technique and greedy approach to solving the NLK. Finally, we compare experimentally the performance of two heuristics, Marginal Bidding and an ILP, in simulations of the TAC SCM bidding problem.

7.2 The Equimarginal Principle

The Prussian economist H. H. Gossen is credited with observing two fundamental laws of utility. The first is the Equimarginal Principle:

> If a man is free to choose among several pleasures but has not time to afford them all to their full extent, then in order to maximize the sum of his pleasures he must engage in them all to at least some extent before enjoying the largest one fully, so that the amount of each pleasure is the same at the moment when it is stopped; and this however different the absolute magnitude of the various pleasures may be.

The Equimarginal Principle applies to problems in which a limited resource (in the above quote, time, but later, means) is to be distributed among a set of independent possible uses. Such problems are ubiquitous. Two problems commonly cited in economics textbooks include: a consumer allocating her (fixed) income among different commodities to maximize her utility; and a firm deciding how to proportion its (finite) labor and capital to maximize its profits.

The second of Gossen's laws is the Law of Diminishing Marginal Returns:

> The amount of any pleasure is steadily decreasing as we continue until at last saturation is reached.

A key assumption underlying both of Gossen's laws is that one cannot enjoy all pleasures indefinitely because a pleasure is not free—rather, it comes at some expense. Indeed, when Gossen writes the "amount of pleasure" he means the additional value that derives from enjoying a bit more of the pleasure at a bit more expense. In modern terms, this quantity—the ratio of a pleasure's marginal value to its marginal cost—can be construed as *marginal return*.

Assuming diminishing marginal returns, it is easy to see that in an optimal solution to such a resource allocation problem, marginal returns are equal.[2] Indeed, if the marginal returns were unequal, a better allocation could be achieved by redistributing a unit of the resource from the use with a lower marginal return to the use with a higher marginal return. Gossen's claim is less obvious: that equal marginal returns imply an optimal solution. For a proof, see Mas-Colell *et al.* [73] (Theorem M.K.3 on page 961), for example.

7.2.1 The Nonlinear Knapsack Problem

The problem domains in which the Equimarginal Principle applies have the flavor of the *knapsack problem*. In this problem, we are given a set of n items, each with a value v_i and a weight w_i, together with a knapsack of finite capacity $C \geq 0$. Our objective is to pack a variety of items in the knapsack such that the sum of the values of the items packed is maximized, but their total weight does not exceed the capacity of the knapsack. Formally,

$$\max_{x_1,\ldots,x_n} \sum_{i=1}^{n} v_i x_i \tag{7.1}$$

[2] For ease of exposition, we assume that in an optimal solution, a strictly positive amount of the resource is allocated to each use: i.e., there exists an interior solution.

$$\text{s.t.} \quad \sum_{i=1}^{n} w_i x_i \leq C \tag{7.2}$$

In the continuous version of the problem, the x_is are in the range $[0, 1]$; in the 0/1 version (which is NP-hard), they are in the set $\{0, 1\}$. In either case, the x_is are bounded; otherwise, the problems would be unbounded.

In the aforementioned sample economics problems, the decision faced is one of choosing not only the best uses for the resource (i.e., which items to pack), but the quantity $x_i \geq 0$ of the resource to allocate to each use, where, in general, the value of a use can depend on its quantity. This final consideration creates a knapsack problem with a potentially nonlinear objective function: i.e., a *nonlinear knapsack problem* (NLK) problem (see, for example, Hochbaum [54]). Specifically,

$$\max_{x_1,\ldots,x_n} \sum_{i=1}^{n} f_i(x_i) \tag{7.3}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} g_i(x_i) \leq B \quad \text{and} \quad \forall i \;\; x_i \geq 0 \tag{7.4}$$

In NLKs, the f_is are value functions; the g_is are cost functions; and the knapsack's capacity C is typically re-interpreted as a budget B.

In a typical instance of the NLK, the x_is are unbounded above, the f_is are real-valued, concave, and nondecreasing, and the g_is are real-valued, convex, and nondecreasing. Concavity (convexity) of the value (cost) function implies the derivative of the value (cost) function, i.e., marginal value (marginal cost), is nonincreasing (nondecreasing). When we divide nonincreasing marginal values by nondecreasing marginal costs, the result is "diminishing marginal returns." Hence, by the Equimarginal Principle, total value is maximized in a NLK when marginal returns are equated across all areas of use: i.e.,

$$\mu_1(x_1) = \frac{f_1'(x_1)}{g_1'(x_1)} = \ldots = \frac{f_i'(x_i)}{g_i'(x_i)} = \ldots = \frac{f_n'(x_n)}{g_n'(x_n)} = \mu_n(x_n) \tag{7.5}$$

The NLK can be solved exactly in polynomial time when f is quadratic and g is linear (see, for example, Tarasov, *et al.* [76]). The approach we take in this chapter can be applied more generally; in particular, it can be used for arbitrary nondecreasing concave value and convex cost functions.

7.2.2 A Discretization Technique

In this section, we propose a strategy for approximating the solution to the NLK. This strategy involves discretizing the problem, and reformulating it as a very special 0/1 (linear) knapsack problem that can be solved greedily. In the next section, we prove that, with finer and finer discretization, (the value of) an optimal solution to our discrete problem becomes a better and better approximation of (the value of) an optimal solution to the original NLK.

Consider a nonlinear knapsack problem with the f_is satisfying the typical assumptions, and $g_i(x_i) = c_i x_i$ for $c_i, x_i \in \mathbb{R}$, for all $i = 1, \ldots, n$. We discretize this problem by assuming the limited resource can be allocated to each use in $K \in \mathbb{N}$ equal parts, so that the size of each is $k = \frac{B}{K}$. By spending k on use i, the incremental quantity $s_i = \frac{k}{c_i}$ of i is consumed. We refer to s_i as the unit size of use i, k as the unit cost, and K as the discretization factor.

Suppose we have consumed the quantity $x_i - s_i$ of use i. Consuming an additional unit of size s_i yields the following, which we call *unit marginal return*:

$$\nu_i(x_i) = \frac{\int_{x_i - s_i}^{x_i} f_i'(t)dt}{\int_{x_i - s_i}^{x_i} g_i'(t)dt} = \frac{f_i(x_i) - f_i(x_i - s_i)}{g_i(x_i) - g_i(x_i - s_i)} = \frac{f_i(x_i) - f_i(x_i - s_i)}{c_i s_i} = \frac{f_i(x_i) - f_i(x_i - s_i)}{k} \tag{7.6}$$

Observe that our assumptions on f ensure that unit marginal returns are nonincreasing, just like marginal returns themselves.

Now, for use i and $j = 1, \ldots, K$, let $v_{ij} = \nu_i(js_i)$ be the value of the jth unit of use i, and let $w_{ij} = k$ be the cost of this unit. We rewrite the objective function (7.1) and the constraint (7.2) to pose a 0/1 (linear) knapsack problem:

$$\max_{x_{ij}} \sum_{ij} v_{ij} x_{ij} \tag{7.7}$$

$$\text{s.t.} \quad \sum_{ij} w_{ij} x_{ij} \leq B \tag{7.8}$$

Here, $x_{ij} \in \{0, 1\}$, for all $i = 1, \ldots, n$ and $j = 1, \ldots, K$.

Constraint 7.8 ensures that the budget B is not exceeded. Since weights are constant and equal to k, this constraint can be restated as follows: $\sum_{ij} x_{ij} \leq K$. Hence, our problem is in fact a very special 0/1 (linear) knapsack problem that can be solved greedily by consuming units of the various uses in sorted order by value, from highest to lowest, until the budget is exhausted, breaking ties by including the jth unit of use i before the $j + 1$st.

Further, a near-optimal solution to the (original) continuous NLK can be constructed from an optimal solution to our discrete problem, precisely because our greedy solution to the latter never includes the jth unit without first including the $j - 1$st. In Section 7.2.3 below, we derive a bound on the quality of this greedy solution as an approximate solution to the continuous NLK, but first we demonstrate the use of our discretization technique by example.

Example Suppose Alice is shopping at a bulk food store and has \$8 to spend on oats and granola. Oats cost \$2 per pound (i.e., $g_o(x_o) = 2x_o$). Granola costs \$6 per pound (i.e., $g_g(x_g) = 6x_g$). Alice's value functions for oats and granola are $f_o(x_o) = 20x_o - 2x_o^2$ and $f_g(x_g) = 24x_g - 3x_g^2$, respectively. The optimal quantities that Alice should buy can be calculated analytically. She should spend $\$\frac{44}{7}$ on oats and $\$\frac{12}{7}$ on granola. This solution has total value ~ 49.71.

Suppose this bulk food store does not accept denominations less than \$2. In other words, Alice must pay with \$2 bills. Alice now faces a discretized knapsack problem of the form just described, with $K = 4$ (the discretization factor) and $k = \frac{\$8}{4} = \$2$ (the unit cost). A unit of oats is of size $s_o = \frac{\$2}{\$2 \text{ per pound}} = 1$ pound, and a unit of granola is of size $s_{\text{g}} = \frac{\$2}{\$6 \text{ per pound}} = \frac{1}{3}$ of a pound.

Alice's marginal returns for all units are listed in Table 7.1. Because her unit marginal returns are decreasing, Alice can find an optimal solution to this discretized problem by allocating her money in a greedy manner to uses in this decreasing order. In this situation, Alice should allocate her four \$2 bills as follows: spend her first \$6 on oats, spend her last \$2 on granola.

	Oats				Granola			
UC	lbs	Value	UMV	UMR	lbs	Value	UMV	UMR
k	x_o	$f_o(x_o)$	$f_o(x_o) - f_o(x_o - s_o)$	$\nu_o(x_o)$	x_g	$f_g(x_g)$	$f_g(x_g) - f_g(x_g - s_g)$	$\nu_g(x_g)$
2	1	18	18	**9**	0.333	7.67	7.67	**3.83**
2	2	32	14	**7**	0.667	14.67	7	3.5
2	3	42	10	**5**	1	21	6.33	3.16
2	4	48	6	3	1.333	26.67	5.67	2.83

Table 7.1: Oats and Granola at a bulk food store. UC stands for unit cost, UMV for unit marginal value, and UMR for unit marginal return.

Note that this optimal solution to the discretized problem is nearly an optimal solution to the corresponding continuous problem: its value is $42 + 7.67 = 49.67$. In this situation, as in most real-life problems, the resource has to be allocated in discrete amounts (e.g., one dollar or one cent). If the store accepts half dollars, then Alice should spend \$6.50 on oats and \$1.50 on granola, which yields total value $\sim$ 49.69; if the store accepts quarters, then Alice should spend \$6.25 on oats and \$1.75 on granola, which yields total value $\sim$ 49.71. The value of the latter solution is within one cent of optimal. We formalize this intuition presently.

7.2.3 Main Theorem

Given an instance of a NLK, let $OPT_{con}(B)$ denote the value of an optimal solution to this problem, given a budget of B; and let $OPT_{dis}(B, K)$ denote the value of an optimal solution to the corresponding discretized problem with discretization factor K. We prove that $OPT_{dis}(B, K)$ approximates the value of $OPT_{con}(B)$. Specifically, $OPT_{dis}(B, K)$ is within a factor of $1 - \epsilon$ of $OPT_{con}(B)$.

Theorem 1 *Assuming the f_is are concave and nondecreasing, the g_is are convex and nondecreasing, and the f_i's and g_i's are continuous,*

$$OPT_{dis}(B) \geq OPT_{con}(B, K)\left(1 - \frac{2n}{K}\right)$$

A proof of this theorem appears in the appendix. The intuition for the proof is as follows. We introduce an intermediate solution that optimally solves a continuous NLK with a slightly different budget B'. The crucial property of this intermediate solution is that it has a value $OPT_{con}(B')$ that is close to both $OPT_{dis}(B, K)$ and $OPT_{con}(B)$. A bound on the distance between $OPT_{dis}(B, K)$ and $OPT_{con}(B)$ is then obtained by adding the distance between $OPT_{dis}(B, K)$ and $OPT_{con}(B')$ to the distance between $OPT_{con}(B')$ and $OPT_{con}(B)$.

A maximization problem admits a Fully Polynomial Time Approximation Scheme if for any $\epsilon > 0$ there exists an algorithm whose run time is polynomial in the input size and $\frac{1}{\epsilon}$ that finds a solution whose value is within a factor of $1 - \epsilon$ of the optimal. Our theorem implies that the NLK admits a FPTAS with $\epsilon = \frac{2n}{K}$ and running time $O(\frac{1}{\epsilon} n \log n)$. The algorithm is shown in Figure 7.1. The first loop runs in time $O(n)$ and the second in time $O(K \log n)$; hence the entire algorithm runs in time $O(n + K \log n) = O(n + \frac{2n}{\epsilon} \log n) = O(\frac{1}{\epsilon} n \log n)$.

Inputs:
discretization factor K
value functions f_i
cost functions g_i
Outputs:
a vector q of quantities consumed, one per use

1. for each use i
   (a) initialize $q_i = 0$
   (b) insert i with priority $\nu_i(s_i) = \frac{f_i(s_i)}{g_i(s_i)}$ into a priority queue Q
2. for $t = 1$ to K
   (a) pop off of Q a use j with the highest priority
   (b) increment q_j by s_j
   (c) insert j into Q with priority $\nu_j(q_j + s_j) = \frac{f_j(q_j+s_j)-f_j(q_j)}{g_j(q_j+s_j)-g_j(q_j)}$
3. return q

Figure 7.1: A FPTAS for NLK. The algorithm runs in time $O(\frac{1}{\epsilon} n \log n)$.

In the next section, we define a tractable approximation of the TAC SCM bidding problem called expected bidding. We note that this problem reduces to a NLK problem with g_i linear and the f_is satisfying the usual assumptions. Hence, our discretization technique, followed by an application of the greedy algorithm, can be used to compute an approximate solution to this problem.

7.3 Bidding in TAC SCM

In TAC SCM, six software agents compete in a simulated sector of a market economy, specifically the personal computer (PC) manufacturing sector. Each agent can manufacture 16 different products, characterized by different *stock keeping units* (SKUs). Building each SKU requires a different combination of components, of which there are 10 different types. These components are acquired from a common pool of suppliers at costs that vary as a function of demand. At the end of each day, each agent converts a subset of its components into SKUs according to a production schedule that it generates for its factory, within a maximum capacity of 2000 cycles. It also reports a delivery schedule assigning the SKUs in its inventory to outstanding customer orders.

The next day, the agents compete in first-price reverse auctions to sell their finished products to customers: i.e., an agent secures an order by *under*bidding the other agents. More specifically, each day the customers send *requests for quotes* (RFQs) to the agents. Each RFQ contains a SKU, a quantity, a due date, a penalty rate, and a reserve price—the highest price the customer is willing to pay. Each agent sends an *offer* in response to each RFQ, representing the price at which it is willing to satisfy that RFQ. After each customer

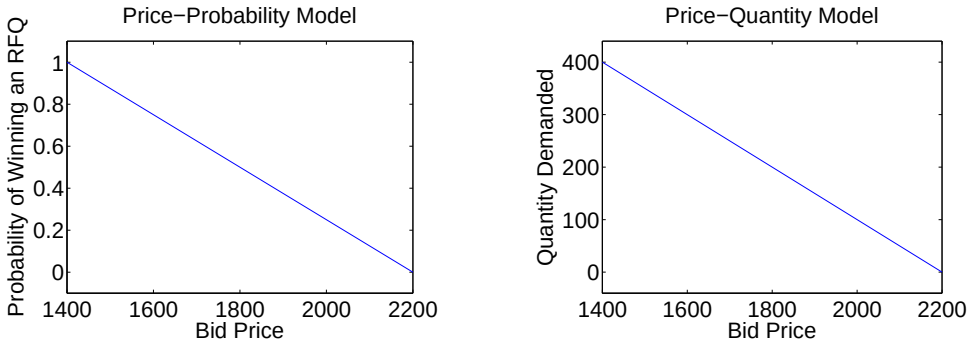


Figure 7.2: (a) Sample price-probability model. (b) Sample price-quantity model.

receives all its offers, it selects the agent with the lowest-priced offer and awards that agent with an *order*. After 220 simulated days of procurement, production, delivery, and bidding each of which lasts a total of 15 seconds, the agents are ranked based on their profits.

Assuming a suitable model of market dynamics—in particular, the current and future prices at which components can be bought and finished products sold—a TAC SCM agent faces three core decision problems [10]: *procurement* of components from suppliers, *bidding* on customer requests for quotes (RFQs), and *scheduling* of factory production and deliveries. In this chapter, we focus on the bidding problem, which subsumes the scheduling problem. A study of how our methods extend to procurement remains for future work. Before detailing our approach to bidding in TAC SCM, we discuss the model of market dynamics on which our formulation of this decision problem is based.

7.3.1 Price-Probability Models

In a marketplace with indistinguishable products, a seller hoping to adjust its market share can do so only by changing its price. Such a seller is likely to gather relevant historical data for use in predicting the market shares that correspond to various price settings. Following Benisch *et al.* [10], we assume that this prediction task has already been completed, and the agent is already endowed with a *price-probability* model that reports the probability of winning an order for each possible bid on current and future RFQs.

Rather than specifying a price-probability model for each individual RFQ, we partition the set of RFQs according to their defining characteristics so that we can obtain a richer set of price-probability models (we are assuming that models built using more data can make more accurate predictions). In TAC SCM, a natural partitioning of the set of RFQs is by SKU type and due date. We refer to each element of such a partition as a *market segment*.

Figure 7.2(a) depicts the price-probability model defined by this equation:

$$p(x) = \frac{2200 - x}{800} \qquad 1400 \leq x \leq 2200 \tag{7.9}$$

This model asserts that a bid of 2200 has no chance of winning (it is the reserve price above which there is no demand), whereas a bid of 1400 is guaranteed to win (it is the price below which there is no supply).

In between, at a price of 1800, say, a bid wins with probability 0.50. Price-probability models need not be linear, but can incorporate whatever techniques necessary to model the likelihood of a bid price being the lowest offered in a market segment.

7.3.2 The Expected Bidding Problem

The N-day stochastic bidding problem is formulated as a recursive stochastic program in Appendix 7.8.1. A tractable approximation of 1-day stochastic bidding, called *expected bidding*, was considered in Benisch *et al.* [10]. In the expected bidding problem, it is assumed that a bid that has probability p of winning an order for quantity q wins a partial order for quantity pq with probability 1. In this deterministic setup, a set of bids on $|R|$ RFQs results in exactly one set of (partial) orders, instead of $2^{|R|}$, as in Equation 7.16.

Collapsing the stochastic content of a price-probability model into deterministic statistics in the form of partial orders is achieved by scaling the model by the demand in the corresponding market segment. We call the ensuing models *price-quantity* models. Recall the price-probability model depicted in Figure 7.2(a). Assume this market segment consists of 80 RFQs of 5 SKUs each, 400 SKUs in total. Since a price of 1800 wins with probability 0.50, at this same price, an agent can expect to win 200 SKUs worth of demand (see Figure 7.2 (b)).

The objective in expected bidding is to find a set of bids x, one per market segment i, that maximizes expected revenue, subject to the constraint that expected production does not exceed available capacity, given, (i) for each market segment, a price-quantity model $h_i(x_i)$ that maps bid prices into quantities—i.e., expected market share; (ii) the total available production capacity C; and (iii) the number of cycles $c_i \in \mathbb{N}$ required to produce one unit of i.

Expected bidding can be stated formally as a mathematical program:

$$\max_{x_1,\ldots,x_n} \sum_{i=1}^{n} h_i(x_i)\, x_i \tag{7.10}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} c_i\, h_i(x_i) \leq C \tag{7.11}$$

where $x_i \in \mathbb{R}$ is the bid price in market segment i. Observe that this problem is an instance of the NLK with $f_i = h_i(x_i)\, x_i$ and $g_i = c_i\, h_i(x_i)$.

Assuming h is invertible, so that the price-quantity model is a 1 to 1 mapping between bid prices and expected market shares, selecting a bid is equivalent to selecting a quantity. In this case, by renaming variables (in particular, letting $x'_i = h_i(x_i)$), we can solve the expected bidding problem as follows:

1. Invert h.

2. Solve this mathematical program:

$$\max_{x'_1,\ldots,x'_n} \sum_{i=1}^{n} x'_i\, h_i^{-1}\,(x'_i) \tag{7.12}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} c_i x'_i \leq C \tag{7.13}$$

where $x'_i \in \mathbb{R}$ is the desired share of market segment i.

3. Bid $h^{-1}(x')$.

Hence, we have reduced the expected bidding problem to solving an instance of the NLK in which uses are market segments, the knapsack's capacity (or the budget) is the factory's capacity, the value functions $f_i(x'_i) = x'_i\, h_i^{-1}(x'_i)$, and the cost functions $g_i(x'_i) = c_i x'_i$. Assuming the f_is are concave and nondecreasing, the results we derived in Section 7.2 are directly applicable. In particular, our Theorem 1 bounds the quality of a solution to Equations 7.12 and 7.13; since the value of such a solution is equal to value of a solution to Equations 7.10 and 7.11, Theorem 1 similarly bounds the quality of a solution to expected bidding. An example of this reduction follows.

Example Consider an instance of the expected bidding problem in which $c_i = 5$ cycles and

$$h_i(x_i) = \frac{2200 - x_i}{2} \qquad 1400 \leq x_i \leq 2200 \tag{7.14}$$

for some market segment i. We invert this price-quantity model, which yields the following "quantity-price" model:

$$h_i^{-1}(x'_i) = 2200 - 2x'_i \qquad 0 \leq x'_i \leq 400 \tag{7.15}$$

(Note that $f_i(x'_i) = x'_i\, h_i^{-1}(x'_i) = x'_i\,(2200 - 2x'_i)$ is concave and nondecreasing on the interval $0 \leq x'_i \leq 400$; hence, our results from Section 7.2 apply.)

Next, we apply the discretization technique to market segment i. If the factory capacity $C = 4000$ cycles and the discretization factor $K = 10$, then the unit cost $k = 400$ cycles and the unit size $s_i = \frac{400 \text{ cycles}}{5 \text{ cycles per SKU}} = 80$ SKUs. By querying the quantity-price model in increments of 80 SKUs, we can generate a list of prices at various incremental quantities. Each revenue is then the product of a price and a corresponding quantity. Unit marginal revenues are the incremental differences in revenue corresponding to the incremental quantities. Finally, unit marginal returns are unit marginal revenues divided by unit costs.

Unit Cost k	Quantity x'_i	Price $h_i^{-1}(x'_i)$	Revenue $f_i(x'_i)$	Unit Marginal Revenue $f_i(x'_i) - f_i(x'_i - s_i)$	Unit Marginal Return $\nu_i(x'_i)$
400	80	2040	163200	163200	408
400	160	1880	300800	137600	344
400	240	1720	412800	112000	280
400	320	1560	499200	86400	216
400	400	1400	560000	60800	152

Table 7.2: Unit Marginal Returns on Market Segment i

The complete list of unit marginal returns in this example is shown Table 7.2. These unit marginal returns could have been computed directly using Equation 7.6. For example, the marginal return on the second unit in market segment i is:

$$v_{i2} = \frac{f_i(2s_i) - f_i(s_i)}{g_i(2s_i) - g_i(s_i)} = \frac{h_i^{-1}(2s_i)\, 2s_i - h_i^{-1}(s_i)\, s_i}{k} = \frac{h_i^{-1}(160)\, 160 - h_i^{-1}(80)\, 80}{(5)(80)} = 344$$

Based on the unit cost c_i and the quantity-price model $h_i^{-1}(x_i)$, we can create such a list of unit marginal returns in each market segment.[3] After doing so for all market segments (i.e., after discretizing the problem), we compute a greedy solution to the ensuing discrete problem. We input the output of this solution, namely a vector of quantities x', to the quantity-price model to obtain a vector of bids, which is our solution to the expected bidding problem.

7.3.3 Marginal Bidding in TAC SCM

There is one important aspect of the TAC SCM bidding problem that we have not thoroughly emphasized, namely that the bidding problem spans multiple days. In this section, we describe how we extend our solution to the 1-day expected bidding problem to the multi-day setting. We call the resulting heuristic *Marginal Bidding*. One of the strengths of a greedy approach to bidding in TAC SCM is that it is natural, and hence easily extensible.

The extension of a greedy solution from the 1-day to a multi-day problem requires an additional parameter. The number of days in the multi-day problem may be too large for even a greedy bidder to reason about within the available time frame. We define the bidder's *window size* W to be the number of days of demand and production considered when making decisions. For example, a window size of 17 means that the bidder can schedule production on 17 days, namely today and on 16 future days. In doing so it considers the current set of RFQs as well as an anticipated set of RFQs for 16 future days. These RFQs are partitioned into market segments by SKU and due date.

When the window size W is large, a large value of K can increase the Marginal Bidder's run time to an unacceptable level. On the other hand, a small value of K can result in a unit size s_i so large that it hinders the algorithm's ability to make short-term decisions at a fine enough granularity. Since we are interested in invoking the Marginal Bidder with large window sizes, we implicitly vary K across market segments (although the theorem presented in Section 7.2.3 is only applicable when K is constant across market segments). More specifically, the Marginal Bidder also takes as input a unit size s_i for each market segment i, with each s_i proportional to the size of i's range of due dates.

A detailed description of the Marginal Bidder appears in Figure 7.3. At a high level, first it greedily fulfills outstanding orders in nonincreasing order of revenue per cycle; second it greedily schedules production of units of the various market segments in nonincreasing order of unit marginal returns; third it bids the price associated with the quantity of demand met in each market segment. Note that bids on all RFQs in a single market segment are equal.

For simplicity, the algorithm we present does not consider component constraints, but it can easily be extended to do so. The Marginal Bidder would have to take as input current component inventory and anticipated daily component arrivals, and could only schedule units for production when sufficiently many components were predicted to be on hand. After scheduling, the corresponding components would be removed from inventory by decrementing the daily component inventory backwards from the production date.

[3] Note that in our implementation we do not explicitly create lists of all unit marginal returns. Since unit marginal returns are nonincreasing, we need only identify the next highest unit marginal return in each market segment. See Figure 7.3 for details.

Inputs:

a window size W
the factory production capacity C
M market segments, each one i characterized by:
a product, a quantity, a range of due dates, a unit size s_i,
an invertible price-quantity model $h_i(x')$, the number of cycles c_i
required to manufacture 1 of i's product, and a "successfully-
scheduled-quantity" q_i initialized to 0
a set of outstanding orders
product inventory

1. sort outstanding orders in nonincreasing order by revenue per cycle
2. for each outstanding order (traversing the list of orders in sorted order)
   (a) use product inventory to fulfill as much of the order as possible
   (b) schedule the rest of the order for production as soon as possible within the scheduling window W
   (c) if the order still cannot be satisfied entirely, undo the inventory and production schedule changes made in the last two steps
3. set j to be the market segment with the highest unit marginal return: i.e.,

$$\begin{aligned} j &= \operatorname*{argmax}_i \left(\nu_i(q_i + s_i)\right) \\ &= \operatorname*{argmax}_i \left(\frac{f_i(q_i + s_i) - f_i(q_i)}{g_i(q_i + s_i) - g_i(q_i)}\right) \\ &= \operatorname*{argmax}_i \left(\frac{(q_i + s_i)h_i^{-1}(q_i + s_i) - q_i h_i^{-1}(q_i)}{c_i s_i}\right) \end{aligned}$$

4. while $\nu_j > 0$
   (a) take up to s_j units of the product associated with j from product inventory
   (b) schedule the remaining units for production as late as possible but before the median due date associated with j and within the scheduling window W
   (c) if s_j units cannot be supplied, then set $\nu_j(q_j + s_j) = -1$ and undo the inventory and production schedule changes made in the last two steps
   (d) otherwise, if s_j units can be supplied, increment q_j by s_j
   (e) set j to be the market segment with the highest unit marginal return
5. for each market segment i
   (a) bid the price at which the agent expects to win the quantity it successfully scheduled: i.e., bid $h_i^{-1}(q_i)$

Outputs: A bid for each market segment, and hence for all the RFQs that comprise that market segment. Note that bids on all RFQs in a single market segment are equal.

Figure 7.3: Marginal Bidding Algorithm

Scheduling To schedule outstanding orders and incremental quantities of market segments for production, there are two natural approaches. First, we can schedule *as soon as possible*, meaning that production is scheduled forwards from the current day. Because the Marginal Bidder schedules greedily, using this method, the most profitable products are produced on the current day, and less profitable products are scheduled for production on subsequent days.

An alternative approach is to schedule for production *as late as possible*, which means that production for an order or an incremental quantity of a market segment is scheduled backwards from its due date. While this approach allows for production decisions to be postponed until more of the uncertainty in the market is resolved, it also allows for empty or near-empty production schedules on the current day, which can be risky. In particular, if demand or prices unexpectedly increase, the Marginal Bidder may wish it had more finished goods on hand.

The Marginal Bidder uses both of these approaches, scheduling outstanding orders as soon as possible because of the penalties incurred for defaulting, and scheduling incremental quantities of market segments as late as possible in order to allow for greater flexibility in bidding decisions.

While the Marginal Bidding algorithm is easy to understand and implement, it behooves us to demonstrate that its performance is acceptable, particularly with market segments and hence units of varying sizes, which renders our theory inapplicable. This is the subject of the remaining sections of this chapter.

7.4 Experimental Results

In this section, we report on experiments designed to compare the performance of four bidding algorithms with varying abilities to reason about the future, an ILP bidding heuristic (see Benisch *et al.* [10]) and three variations on the Marginal Bidding heuristic developed in this chapter. We expect the Marginal Bidders to compute bids faster than the ILP, and we expect this speed to enable them to consider larger windows into the future, which should lead to higher revenues than the ILP under some market conditions (and never lead to lower revenues). We test these conjectures on instances of TAC SCM bidding in a simulator we built that tests individual agents in isolation by generating decision-theoretic simplifications of the game-theoretic problems TAC SCM agents face.

7.4.1 Test Suite

We tested an integer linear programming solution with a 1 day window (ILP), meaning it did not reason about any future demand beyond the current RFQs and outstanding orders arriving each day. We compared this ILP with three variations of the marginal bidder: a marginal bidder with a 17-day[4] window (MB-17), a marginal bidder with a full-game window (MB-Full), and a marginal bidder with a hybridization of the two that considers the full game window, but does so at a coarser granularity as it reasons further into the future in order to keep its run time in check (MB-Coarse).

[4]We chose 17 as the default window size because it is the last day on which a current RFQ with the latest possible due date can be filled in TAC SCM.

The 17 Day (MB-17) and full-game (MB-Full) bidders partition demand (i.e., the set of current and future RFQs) into market segments by SKU type and due date, and the size of a unit in each market segment is 1 product. The hybrid full-game bidder (MB-Coarse) also divides demand up by SKU type and due date. For the first 17 days, it considers each due date separately, but beyond the initial 17 days it divides demand into increasingly larger chunks, whose due-date ranges grow by powers of 1.8.[5] For the coarse bidder, each market segment's unit size is 1 product multiplied by the number of days in that segment.

Since each TAC SCM day is 15 seconds, and a bidding policy is one of many decisions an agent must make each day, it may not be wise for an agent to allot too much of its daily run time to bidding alone. We thus study a likely TAC SCM situation in which the bidder is only given 5 seconds to formulate its daily bidding policy. The full-game Marginal Bidder often requires more than 5 seconds per day to compute its policy, so it is not a feasible TAC SCM bidder, but we include it in this discussion for benchmarking purposes.

In order to reach a reasonable solution within the allotted 5 seconds, the ILP dynamically calculates an appropriate degree of discretization using a formula that was empirically determined to minimize the ILP's distance from optimality within a 5 second window. The equation for the number of price points is 2300/(# of RFQs + # of Orders). An ILP with a run time of up to 15 seconds and additional price points was also tested, but did not yield significant gains.[6]

7.4.2 Experimental Design

Recall that in TAC SCM each agent submits its bids to a reverse auction, so that an RFQ is awarded to the agent that bids the lowest price below the reserve price. Using our simulator, we tested our bidding algorithms in isolation, not against other bidding agents, as would be the case in a true reverse-auction setting. The simulator simply awarded contracts by transforming each offer into an order with a certain probability, namely that which is associated with the bid price under the price-probability model for the relevant market segment. Hence, we simulated the stochastic bidding problem, although our heuristic solutions are approximate solutions to the expected bidding problem.

In our experiments, agents were endowed with perfect price prediction: i.e., the various price-probability models (one per market segment per simulation day) were shared between the agent and the simulator. Regarding demand, the number of customer RFQs of each SKU type scheduled to arrive each day was broadcast before the simulations began. Then, on each simulation day, the agents received a set of current RFQs whose quantities and due dates were sampled from the distributions outlined in the TAC SCM game specifications, and they assumed that the quantity and due date associated with each of the future RFQs were the means of the same distributions.[7] Reserve prices were also known to the agents; they were built in to the price-probability models.

[5] For example, SKUs due on days 18-19 are grouped together ($1.8 \approx 2$), as are SKUs due on days 20-22 ($1.8^2 \approx 3$), and days 23-28 ($1.8^3 \approx 6$), and so on.

[6] An ILP with a 2-day window was also tested, as was one with a 17-day window and constrained capacity (2000 cycles on day 1 and 2000 cycles on days 2 through 17). Again, these variants did not yield significant gains.

[7] The reason for drawing a distinction between the quality of the predictions of the number of RFQs of each SKU type and their attributes is: the former is somewhat predictable in TAC SCM—it is dependent on history (see, for example, Kiekintveld *et al.* [60])—while the latter is not.

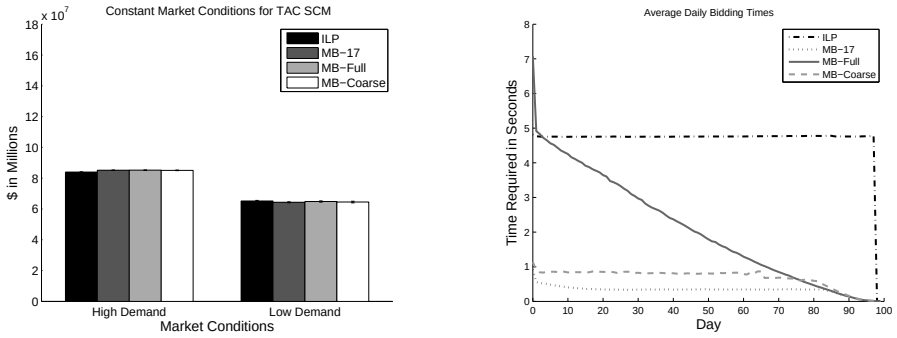


Figure 7.4: (a) Revenue from deliveries under constant market conditions. (b) Average daily bidder times in high demand conditions. Low demand bidder times were similar.

We tested our bidders by running 25 simulations of 100 day games under three families of market conditions: (i) constant: i.e., conditions on one day are reflective of the conditions on the next; (ii) gradually changing conditions; and (iii) sudden shifts, including demand or price shocks. Under the non-constant conditions we examine situations of rising demand and price. Falling demand and price conditions are not presented, but produce similar results.

For simplicity, in these simulations we assume infinitely many components. Introduction of component constraints does not appear to significantly alter the relative performance of our bidders.

7.4.3 Constant Conditions

In our first set of market conditions, we compare the bidders under constant demand and price. Presented here are steady conditions of high demand, defined as 20 RFQs per SKU type per day, which is the maximum possible according to the TAC SCM game specification, and low demand, defined as 5 RFQs per SKU type per day, the lowest possible. Prices in this experimental setup range linearly from 50% to 125% of the SKU base price.

Under such conditions, we should expect to see no particular advantage to planning for the future, since an optimal solution to the entire game can be contructed by concatenating a sequence of optimal solutions, one per day, computed for each day in isolation. Indeed, in terms of revenue, all the bidders are competitive with one another under these conditions (see Figure 7.4(a)). Note however that MB-17 and MB-Coarse arrive at their solutions an order of magnitude faster than the ILP or the MB-Full bidding algorithms (see Figure 7.4(b)).

7.4.4 Shifting Conditions

More interestingly, market conditions can change over the course of a TAC SCM game, either steadily as in a market adjustment or suddenly as in a demand or price shock. In our next experimental setup, demand is initialized to 5 RFQs per SKU per day, and prices range linearly from 50% to 75% of the SKU base price. We then considered shifts to 20 RFQs per SKU per day and prices ranging from 100% to 125% of the base prices

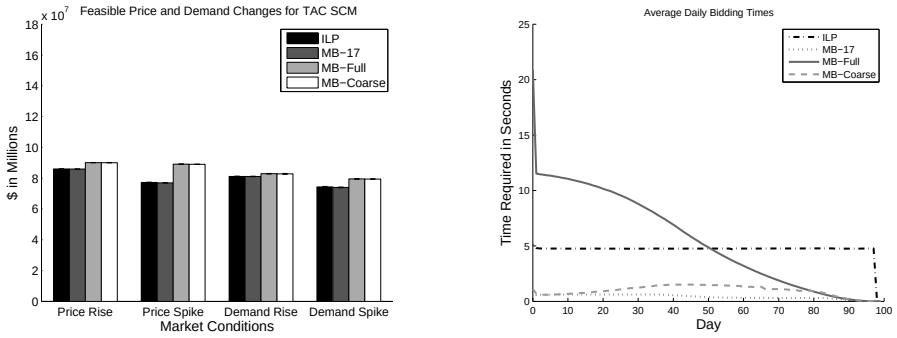


Figure 7.5: (a) Revenue from deliveries under feasible SCM market conditions. (b) Average daily bidder times in Price Rise conditions. Other market conditions had similar run times.

by day 50. These shifts are representative of the magnitude of changes an agent might observe while playing a typical TAC SCM game. These changing market conditions were tested both as steady linear accumulations from day 1 to day 100 and as abrupt surges on day 50. In our price-shifting simulations demand is held constant; in our demand-shifting simulations price is held constant.

As expected, those bidders with more extensive knowledge of the future (MB-Full, MB-Coarse) are able to exploit the mid-game surges by dedicating production from today to future demand when conditions are more favorable. Bidders with a shorter window (ILP, MB-17) are unable to plan far enough ahead to take advantage of the upcoming shifts, and hence accumulate less revenue over the course of the game. In addition to the additional revenue gained by exploiting its knowledge of the future, the MB-Coarse bidder continues to run in substantially less time than the ILP. See Figure 7.5.

The advantages of a larger window are more pronounced under those market conditions in which the shift in demand or price comes as a sudden spike rather than as a steady rise. When demand or prices rise gradually, even an agent with a small window is aware that tomorrow's market conditions are slightly more profitable than today's, and can reserve some inventory for future sales. However, when demand or price spikes suddenly, an agent is not aware of more desirable future market conditions until the spike falls within its window.

Because one of our simplifying assumptions for these simulations is that agents have perfect models of future demand and price, it is encouraging that MB-Coarse performs just as well as MB-Full. Their similar performance suggests that the benefits of looking into the future may still be realized by agents with more realistic but less accurate models.

7.4.5 Extreme Conditions

Within the context of TAC SCM, the previous experimental setup characterizes shifts from one extreme set of realistic conditions to another, and the gains resulting from knowledge of the future are modest. However, it is easy to envision markets that are more naturally volatile or are subject to large seasonal trends in demand. The greater the extent to which market conditions vary across time, the greater the opportunity for bidders

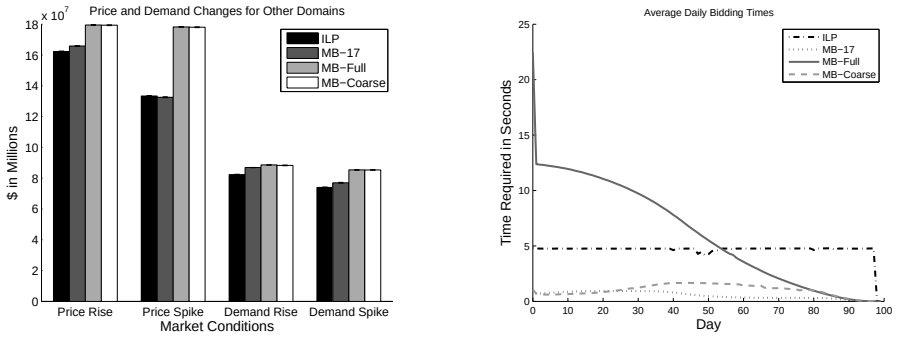


Figure 7.6: (a) Revenue from deliveries in extreme market conditions. (b) Average daily bidder times in Price Rise conditions. Other market conditions had similar run times.

able to consider a larger window into the future to earn greater profits. In order to demonstrate this effect, we present a second set of simulations assuming shifting market conditions, but the shifts are of greater magnitudes. In particular, demand surges from 5 to 40 RFQs per SKU per day, and price rises from $[50\%, 75\%]$ to $[200\%, 250\%]$ of the base prices, again as both an interpolated steady rise and as an overnight jump.

With no significant changes in run time (compare Figures 7.5(b) and 7.6(b)), the marginal bidders are able to exploit the extreme changes in market conditions, and in particular the bidders with larger windows (MB-Full and MB-Coarse) are able to earn even greater profits (see Figure 7.6(a)). Also of interest is the relative impact of demand changes versus price changes. We observe a more pronounced impact when considering knowledge of the future under price-changing conditions for two reasons.

First, because of capacity constraints, an agent can only produce a limited quantity of each product on each day. Hence, an increase in demand does not necessarily translate into an increase in the number of finished products. So even if a demand shift results in higher prices, revenues need not increase substantially, particulary in comparison to the revenue increase associated with a price increase (see Figure 7.7(a)). If the magnitude of the price shifts in our experiments were reduced, or if production capacity were increased, stockpiling products until a demand shift could be as worthwhile as stockpiling products until a price shift.

The second factor that mitigates the advantage of knowledge of the future in conditions of shifting demand is the relatively flat slopes of our quantity-price curves. With flatter slopes, the difference in revenue between prices on the initial curve and prices on the curve after a demand shift is small (Figure 7.7(b)). Thus it matters less if the agent stockpiles products for the future, and in turn it matters less if the agent has any knowledge of the future. If the quantity-price curves had steeper slopes, knowledge of the future in conditions of shifting demand would likely prove more valuable than our current experiments suggest.

7.5 Related Work

The NLK problem, also known as the Nonlinear Resource Allocation problem, is well-studied. The interested reader is referred to Patrikson [89] for a recent survey, which includes a number of algorithms that solve various formulations of the NLK. One feature of the approach described here is that we construct a solution

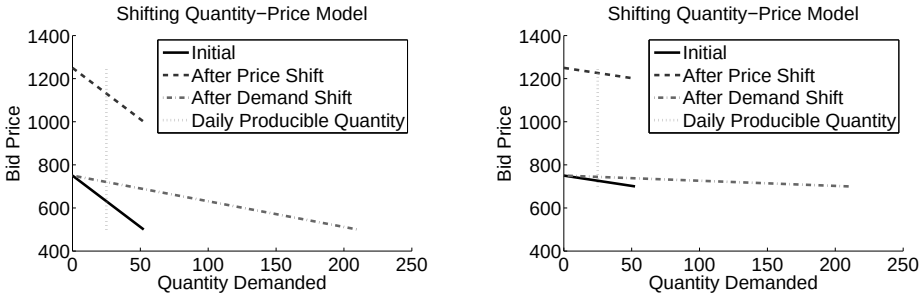


Figure 7.7: (a) Sample quantity-price models before any shift, after a price shift, and after a demand shift. To illustrate the constraining effects of production capacity, also shown is a sample daily producible quantity. In our experiments, price shifting conditions result in higher revenues than demand shifting conditions, and thus knowledge of a future price shift is more valuable than knowledge of a future demand shift. (b) For quantity-price models with flat slopes, predicting future demand is not very important.

incrementally; this makes it easy to check if the solution remains feasible under more general conditions: e.g., when not only capacity but also scheduling or component constraints are present. Also, unlike other techniques (e.g., Bretthaur and Shetty [17]), our algorithm does not rely on value or cost functions being differentiable or having closed-form representations (although our theoretical results do not hold under these more general assumptions).

Most closely related to our work is the work of Hochbaum [54]. In her study of the NLK, she employs a discretization technique that generalizes the one presented here. Correspondingly, the discretized problem we construct is a special case of her *simple allocation problem*; in our problem, all variables are binary rather than non-negative and integer-valued. Like us, she solves her discrete problem greedily, and, invoking work in a related paper [55], she connects this greedy solution back to an optimal solution to the original (continuous) NLK. Her results apply in the special case in which the g_is are linear. Our main theorem applies more generally; in particular, the g_is may be convex.

Benisch *et al.* [9] reduce a probabilistic pricing problem (akin to 1-day expected bidding) to the NLK, and present an ϵ-optimal solution to this problem assuming diminishing marginal returns. Although they demonstrate that their algorithm can be efficient in practice, they provide no theoretical guarantees on its run time. Also, their algorithm is not incremental, so it is not immediately obvious how to extend it to apply to problems with additional constraints.

The TacTex team developed a greedy bidding agent for TAC SCM along the lines of the Marginal Bidder presented here, with a few subtle distinctions [88]. TacTex is initialized to bid reserve prices on each RFQ, and then it iteratively reduces its bids according to a selection mechanism until production capacity is reached or profits are no longer increasing. The selection mechanism relies on a heuristic that determines whether the most limiting resource is production capacity (in which case it selects by profit per cycle) or component availability (in which case it selects by change-in-Profit / change-in-Probability). No theoretical guarantees validating their approach are discussed.

Finally, researchers at the Cork Constraint Computation Center implemented an ILP approach to bidding in a constraint-based TAC SCM agent, Foreseer [19]. Not unlike the expected bidder posited in Benisch *et*

al. [10], Foreseer uses profit as the objective function, bid prices as the decision variables, and constraints based on factory capacity, component availability, and reserve prices.

7.6 Summary and Future Work

In this chapter, we described a technique for solving the NLK by converting it into a (discrete) simple allocation problem that can be solved greedily. Our theoretical results establish that the greedy solution to the resulting simple allocation problem is a FPTAS for the NLK. Although more complicated algorithms with better run times are known, our simple incremental solution affords us extra flexibility. In particular, the greedy algorithm extends easily into the Marginal Bidding heuristic, which solves an extended version of the NLK with natural scheduling and component constraints.

Our ultimate goal is to develop a scalable bidding algorithm that can be extended into a procurer capable of reasoning about long-term future demand. Because the ILP considers each RFQ as a separate decision variable, its complexity grows rapidly as a function of the number of RFQs. By reasoning about SKUs in collective market segments, the Marginal Bidders avoid this complexity and appear to be more readily extensible to the procurement problem. However, it remains to be seen whether our Marginal Bidding approach can be extended to handle interdependent uses, where devoting resources to one use can affect the marginal return of another. Interdependencies arise naturally in procurement because components are shared among SKU types.

Despite the game-theoretic nature of bidding in TAC SCM, our focus here was simply on a decision-theoretic (stochastic) optimization problem, not on game-theoretic equilibrium calculations. The enormity of the decision space in TAC SCM renders game-theoretic strategic analysis intractable with current technology. It remains to be seen whether an effective game-theoretic approach can be developed to exploit strategic opportunities in the TAC SCM game. In the near future, we plan to test the robustness of our algorithms to imperfect modeling of future prices and demand. Doing so would lead to progress in addressing the challenging game-theoretic issues that arise in environments like TAC SCM that are inhabited by multiple artificially intelligent agents.

7.7 Acknowledgments

The authors would like to thank Warren Schudy for his input on the statement of our main theorem. We are also grateful to an anonymous reviewer for his or her comments. This research was supported by NSF Career Grant #IIS-0133689.

7.8 Appendix

7.8.1 The Stochastic Bidding Problem

The bidding problem posed here is intended to model the bidding problem that agents face in TAC SCM. For simplicity, we assume all due dates are set past the end of the game, making penalties irrelevant. Also, as we are concerned only with bidding and not with procurement in this chapter, all components are assumed to be infinitely available at no cost.

Agents are assumed to have perfect price prediction, that is, they know the probability of winning an order as a function of any bid they submit. We encode this information in price-probability models. They are also assumed to have access to an accurate stochastic model of the number and variety of RFQs that will arrive on each future day of the game.

A decision-theoretic version of the TAC SCM bidding problem, under the aforementioned assumptions, can be formulated naturally as a recursive stochastic program. We have not seen this program appear elsewhere in the literature (except in Odean *et al.* [84]), so we present it here, using the notation explained in Figure 7.8.

Variables

$x_r \geq 0$	bidding policy: bid price for RFQ r
$y_j \geq 0$	production schedule: quantity of SKU j
$z_i \in \{0, 1\}$	delivery schedule:
	1 if order i is delivered; 0 otherwise

Indexes

t	day index
j	SKU index

Functions

$p(r, x_r)$	probability of winning RFQ r with bid x_r

Constants

a_j	number of units of SKU j delivered
b_j	number of units of SKU j in inventory
c_j	cycles expended to produce one unit of SKU j
d_{ij}	1 if order i is for SKU j; 0 otherwise
$\pi_i(t)$	revenue (minus penalty) for delivering order i on day t
	zero if t is past order's due date
q_i	quantity of order i
N	total number of days
C	daily production capacity in cycles
O	set of outstanding orders
Q	set of (today's) orders
R	set of (today's) RFQs
R'	set of tomorrow's RFQs
h	history of RFQs received until now

Figure 7.8: Notation for Recursive Stochastic Program

The recursive function takes five inputs: today's product inventory, today's outstanding orders, today's RFQs, the history of RFQs received on previous days, and today's date. The objective is to choose bids on

today's RFQs and to decide upon today's production and delivery schedules in such a way as to maximize today's revenue plus expected future revenue.

Bids on day t are placed on RFQs received that day. The set of RFQs R' received on day $t+1$ is a random variable that is independent of any decisions but depends on the history of past RFQs received.

The bids placed on day t determine the likelihoods of receiving various sets of orders on day $t+1$. Each set of new orders is called a *scenario*. Each scenario Q is weighted by probability $\Pr(Q)$ as determined by the given price-probability model. Specifically, $\Pr(Q)$ equals the product of the probabilities of winning all RFQs that are part of Q and the probabilities of not winning RFQs that are not part of Q (Equation 7.17).

Delivery and production scheduling decisions today affect what will remain in product inventory tomorrow. Indeed, tomorrow's product inventory equals today's product inventory b minus any product inventory depleted by today's deliveries a plus any additional inventory produced today y.

Each day capacity and allocation constraints are enforced. Equation 7.18 ensures that there are enough products in inventory for today's delivery schedule. Equation 7.19 ensures that today's production schedule does not consume more cycles than the daily capacity.

The base case (Equation 7.20) of the recursion pertains to the last day. Orders can be scheduled for delivery but there is no production or bidding.

if $0 \leq t < N$,

$$F(b, O, R, h, t) = \max_{x,y,z} \sum_{i \in O} z_i \pi_i(t) +$$

$$\sum_{Q \in 2^{|R|}} \Pr(Q) E_{R'|h} \left[F(b - a + y, O \cup Q, R', h \cup R, t+1) \right] \tag{7.16}$$

subject to:

$$\Pr(Q) = \prod_{r \in Q} p(r, x_r) \prod_{r \notin Q} (1 - p(r, x_r)) \tag{7.17}$$

$$a_j = \sum_{i | i \in O, d_{ij} = 1} z_i q_i \quad \forall j; \quad a \leq b \tag{7.18}$$

$$\sum_j y_j c_j \leq C \tag{7.19}$$

if $t = N$,

$$F(b, O, R, h, t) = \max_z \sum_{i \in O} z_i \pi_i(t) \tag{7.20}$$

7.8.2 Proof of the Main Theorem

The term "unit" is used throughout the chapter to refer to the amount of use s_i that can be obtained by spending additional k dollars. With a linear g, the unit s_i remained constant regardless of how many units have been consumed before. With a convex g, unit prices increase with consumption and therefore k dollars buy less. So the unit s_i depends on how much has been bought before; i.e., it is a function $s_i(x_i)$ returning

additional amount of use i obtained from increasing the spending from $g_i(x_i) - k$ to $g_i(x_i)$:

$$s_i(x_i) = g_i^{-1}(g(x_i)) - g_i^{-1}(g(x_i) - k) = x_i - g_i^{-1}(g(x_i) - k)$$

where $g_i^{-1}(m)$ is the amount of use i obtained by spending m [8].

In light of a changing unit s_i, unit marginal returns may be better be called k-marginal returns:

- $\nu_i(x_i)$ - marginal return from spending additional k after spending $g(x_i) - k$ on use i.

In this appendix, we prove our main theorem. The proof relies on the following observation relating marginal and k-marginal returns:

Observation 1. For all uses i on which more than k is spend in the optimal solution (i.e., $x_i > g_i^{-1}(k)$)),

$$\nu_i(x_i) \leq \mu_i(x_i - s_i(x_i)) \tag{7.21}$$

$$\nu_i(x_i) \geq \mu_i(x_i) \tag{7.22}$$

Proof For each use i,

$$\begin{array}{rclcrcl}
\nu_i(x_i) & = & \frac{\int_{x_i - s_i(x_i)}^{x_i} f_i'(t)dt}{\int_{x_i - s_i(x_i)}^{x_i} g_i'(t)dt} & & \nu_i(x_i) & = & \frac{\int_{x_i - s_i(x_i)}^{x_i} f_i'(t)dt}{\int_{x_i - s_i(x_i)}^{x_i} g_i'(t)dt} \\
& \leq & \frac{\int_{x_i - s_i(x_i)}^{x_i} f_i'(x_i - s_i(x_i))dt}{\int_{x_i - s_i(x_i)}^{x_i} g_i'(x_i - s_i(x_i))dt} & & & \geq & \frac{\int_{x_i - s_i(x_i)}^{x_i} f_i'(x_i)dt}{\int_{x_i - s_i(x_i)}^{x_i} g_i'(x_i)dt} \\
& = & \frac{f_i'(x_i - s_i(x_i)) \int_{x_i - s_i(x_i)}^{x_i} 1dt}{g_i'(x_i - s_i(x_i)) \int_{x_i - s_i(x_i)}^{x_i} 1dt} & & & = & \frac{f_i'(x_i) \int_{x_i - s_i(x_i)}^{x_i} 1dt}{g_i'(x_i) \int_{x_i - s_i(x_i)}^{x_i} 1dt} \\
& = & \frac{f_i'(x_i - s_i(x_i))}{g_i'(x_i - s_i(x_i))} & & & = & \frac{f_i'(x_i)}{g_i'(x_i)} \\
& = & \mu_i(x_i - s_i(x_i)) & & & = & \mu_i(x_i)
\end{array}$$

These inequalities follow from the fact that f_i' is nonincreasing and g_i' is nondecreasing.

Recall our main theorem:

Main Theorem. *Assuming the f_is are concave and nondecreasing, the g_is are convex and nondecreasing, and the f_i's and g_i's are continuous,*

$$OPT_{dis}(B) \geq OPT_{con}(B, K)\left(1 - \frac{2n}{K}\right)$$

Proof Let x denote an optimal solution to the discrete problem. Suppose the budget is not exhausted in solution x. This can only happen when marginal returns of all units that are not in the solution are zero

[8] If less than k is spent on use i, then $s_i(x_i)$ is the amount of use i obtain from spending k: $s_i(x_i) = s_i(g_i^{-1}(k)) = g_i^{-1}(k) \quad \forall x_i \mid x_i \leq g_i^{-1}(k)$

(marginal returns cannot be negative because f_is are assumed to be nondecreasing): $\nu_i(x_i+s_i(x_i)) = 0 \quad \forall i$. However by the definition of unit marginal returns, $\nu_i(x_i + s_i(x_i)) = 0$ only if $f_i'(x_i) = 0$, which implies that $\mu_i(x_i) = 0$. Since marginal returns are zero in the solution to the discrete problem, taking additional amounts of any use does not increase the value. Therefore the value of solution x is the same as the value of the optimal solution to the continuous problem, and the theorem holds. In the remainder of the proof we will focus on the case when the budget is exhausted in the optimal solution to the discrete problem.

Let $j = \text{argmin}_{i \in U} \nu_i(x_i)$ denote the use with the smallest marginal return on its last unit. The greediness of the optimal algorithm implies that the marginal return of any unit that is included in the solution is higher than the marginal return of any unit that is not in the solution. In particular,

$$\nu_j(x_j) \geq \nu_i(x_i + s_i(x_i)) \quad \forall i \tag{7.23}$$

Define the set of uses $U = \{i | x_i > 0\}$ that are part of the solution to the discrete problem. We first focus on these uses. Equation 7.21 and the definition of j yield $\forall i \in U$: $\mu_i(x_i - s_i(x_i)) \geq \nu_i(x_i) \geq \nu_j(x_j)$; combining Equations 7.23 and 7.22 yields: $\nu_j(x_j) \geq \nu_i(x_i + s_i(x_i)) \geq \mu_i(x_i + s_i(x_i))$. Now, by the continuity of f_i', for each $i \in U$ there exists $y_i \in [x_i - s_i(x_i), x_i + s_i(x_i)]$ such that $\mu_i(y_i) = \nu_j(x_j)$, so marginal returns can be equated at the value $\nu_j(x_j)$.

The uses $i \notin U$ that are not part of the optimal solution ($x_i = 0$) to the discrete problem, may still be in the solution to the continuous problem. This is the case when $\mu_i(0) > \nu_j(x_j)$. Combining Equations 7.23 and 7.22 yields

$$\nu_j(x_j) \geq \nu_i(x_i + s_i(x_i)) = \nu_i(s_i(x_i)) \geq \mu_i(s_i(x_i)) \tag{7.24}$$

By the continuity of f_i', for all $i \notin U$ with $\mu_i(0) > \nu_j(x_j)$ there exists $y_i \in [0, s_i(x_i)]$ such that $\mu_i(y_i) = \nu_j(x_j)$, so these marginal returns can also be equated at the value $\nu_j(x_j)$.

Let B' denote the budget that is spent when marginal returns are equated at $\nu_j(x_j)$, and let y denote a corresponding solution. By the Equimarginal Principle, y is an optimal solution to the continuous problem with budget B'.

Case 1 If $B' < B$, then the budget B in the continuous problem is not exhausted at the point y of equal marginal returns. Hence, y's total value need not exceed $OPT_{con}(B)$. To upper bound the value of the continuous solution, we add to y's total value an overestimate of the amount of additional value that could be accrued by spending the remaining budget, $B - B'$.

To exhaust this budget, at most one additional unit of each good must be acquired because: (i) the budget is exhausted in the solution x to the discrete problem, and (ii) in the worst case $y_i = x_i - s_i(x_i)$. Since each unit costs k, the unspent budget is at most nk. Additional spending on any use yields a maximum marginal return of $\mu_i(y_i) = \nu_j(x_j)$, since marginal returns are nonincreasing. Therefore, spending an extra amount of nk yields at most $nk\nu_j(x_j)$ of extra value.

$$OPT_{con}(B) \leq \sum_{i=1}^{n} f_i(y_i) + nk\nu_j(x_j) \leq \sum_{i=1}^{n} f_i(x_i + s_i(x_i)) + nk\nu_j(x_j)$$

We apply Lemma 2 to get

$$OPT_{con}(B) \leq OPT_{dis}(B,K)\left(1+\frac{2n}{K}\right)$$

Case 2 If $B' \geq B$, then y's total value, namely $\sum_{i=1}^{n} f_i(y_i)$, is an immediate upper bound on $OPT_{con}(B)$. Hence,

$$\begin{aligned} OPT_{con}(B) &\leq \sum_{i=1}^{n} f_i(y_i) && (7.25)\\ &\leq \sum_{i=1}^{n} f_i(x_i+s_i(x_i)) + nk\nu_j(x_j) && (7.26)\\ &\leq OPT_{dis}(B,K)\left(1+\frac{2n}{K}\right) && (7.27)\end{aligned}$$

The last inequality follows from Lemma 2.

To complete the proof, rewrite

$$OPT_{con}(B) \leq OPT_{dis}(B,K)\left(1+\frac{2n}{K}\right)$$

as

$$\begin{aligned} OPT_{dis}(B,K) &\geq OPT_{con}(B)\frac{1}{1+\frac{2n}{K}} && (7.28)\\ &= OPT_{con}(B)\left(1-\frac{\frac{2n}{K}}{1+\frac{2n}{K}}\right) && (7.29)\\ &\geq OPT_{con}(B)\left(1-\frac{2n}{K}\right) && (7.30)\end{aligned}$$

QED

Lemma 2 *Let x denote an optimal solution to the discrete problem with budget B. Assuming the f_is are concave and nondecreasing, the g_is are convex and nondecreasing, and the f_i's and g_i's are continuous,*

$$\sum_{i=1}^{n} f_i(x_i+s_i(x_i)) + nk\nu_j(x_j) \leq OPT_{dis}(B,K)\left(1+\frac{2n}{K}\right)$$

where j is a use whose marginal return on its last unit included in x is minimal.

Proof First, observe that

$$\sum_{i=1}^{n}\left(f_i(x_i+s_i(x_i)) - f_i(x_i)\right) = \sum_{i=1}^{n} k\nu_i(x_i+s_i(x_i)) \leq \sum_{i=1}^{n} k\nu_j(x_j) = nk\nu_j(x_j)$$

Hence,

$$\begin{aligned}
&\sum_{i=1}^{n} f_i(x_i + s_i(x_i)) + nk\nu_j(x_j) \\
&\qquad = \sum_{i=1}^{n} (f_i(x_i) + [f_i(x_i + s_i(x_i)) - f_i(x_i)]) + nk\nu_j(x_j) \\
&\qquad = OPT_{dis}(B,K) + \sum_{i=1}^{n} (f_i(x_i + s_i(x_i)) - f_i(x_i)) + nk\nu_j(x_j) \\
&\qquad \leq OPT_{dis}(B,K) + 2nk\nu_j(x_j) \\
&\qquad = OPT_{dis}(B,K)\left(1 + \frac{2nk\nu_j(x_j)}{OPT_{dis}(B,K)}\right)
\end{aligned}$$

As f_is are increasing we can always exhaust the budget without decreasing the value. Therefore, assume without loss of generality that the budget is exhausted in an optimal solution to the discrete problem: i.e., all K units are consumed. Since $\nu_j(x_j)$ is the lowest unit marginal return among the K units in such a solution, each unit contributes at least $k\nu_j(x_j)$ of extra *value*, since (unit) marginal return equals (unit) marginal value divided by (unit) marginal cost. Hence, $OPT_{dis}(B,K) \geq Kk\nu_j(x_j)$, from which it follows that:

$$\begin{aligned}
\sum_{i=1}^{n} f_i(x_i + s_i(x_i)) + nk\nu_j(x_j) &\leq OPT_{dis}(B,K)\left(1 + \frac{2nk\nu_j(x_j)}{OPT_{dis}(B,K)}\right) && (7.31) \\
&\leq OPT_{dis}(B,K)\left(1 + \frac{2nk\nu_j(x_j)}{Kk\nu_j(x_j)}\right) && (7.32) \\
&= OPT_{dis}(B,K)\left(1 + \frac{2n}{K}\right) && (7.33)
\end{aligned}$$

In the above derivation, we ignored the possibility that $\nu_j(x_j)$ may be zero. However, the weak inequliaty in Equation 7.31 holds with an equality when $\nu_j(x_j) = 0$ satisfying the lemma. QED

Chapter 8

Bidding in Simultaneous Auctions[1]

We study a suite of heuristics that were designed for bidding in the simultaneous auctions that characterize the Trading Agent Competition (TAC) Travel Game. At a high-level, the design of many successful TAC agents can be summarized as: (i) *price prediction*: build a model of the auctions' clearing prices, and (ii) *optimization*: solve for an (approximately) optimal set of bids, given this model. We focus on the optimization piece of this design.

Analytically, we address the (decision-theoretic) deterministic bidding problem. We derive the class of bidding heuristics that solves this problem optimally. In particular, we prove that the marginal-value-based bidding heuristic implemented in RoxyBot–2000—one of the top-scoring agents in TAC 2000—is an instance of this class. Moreover, we identify the special set of circumstances in which bidding marginal values themselves is also optimal.

Experimentally, we embed these heuristics in TAC Travel agents and evaluate their success in the game-theoretic bidding problem that characterizes TAC Travel auctions. We find that RoxyBot–2000's bidding heuristic dominates the others in our test set. We conclude that RoxyBot–2000's bidding heuristic is effective in that it performs well in a decision-theoretic setting assuming perfect price prediction and a game-theoretic setting where price predictions are imperfect.

8.1 Introduction

Simultaneous auctions, which arise naturally on websites such as `ebay.com` and `amazon.com`, are forums on which to trade many goods simultaneously. Such auctions present a challenge to bidders, particularly when complementary and substitutable goods are on sale. Complementary goods are goods whose values are superadditive: *i.e.*, for goods x and y, $v(x) + v(y) \leq v(xy)$. For example, a flash, a tripod, and a case complement a camera, since an agent does not desire any of the former if she does not acquire the latter. Substitutable goods are goods whose values are subadditive: i.e., for goods x and y, $v(x) + v(y) \geq v(xy)$. For example, a Canon and an Olympus are substitutes, since an agent desires one or the other, but not both.

[1] Joint work with Amy Greenwald and Seong Jae Lee.

In contrast to *combinatorial auctions*, in which bids may be placed for combinations of goods (e.g., "camera and flash for $295"), in simultaneous auctions, separate bids are placed for each individual good. In combinatorial auctions, the NP-hard problem of choosing a set of winning bids that maximizes revenue—the so-called winner determination problem—falls in the hands of the auctioneer. In simultaneous auctions, however, the complexity burden falls upon the bidders.

In this chapter, we study heuristics that were designed for bidding in the simultaneous auctions that characterize the Trading Agent Competition (TAC) Travel Game [109]. A TAC Travel agent is a simulated travel agent whose task is to organize itineraries for a group of clients to travel to and from TACTown and Boston. The agent's objective is to procure travel goods that satisfy its clients' preferences as inexpensively as possible. Travel goods are sold in simultaneous auctions, as follows:

- flights are sold by the "TAC seller" in dynamic posted-pricing environments; no resale is permitted
- hotel reservations are sold by the "TAC seller" in multi-unit ascending call markets; specifically, 16 hotel reservations are sold in each hotel auction at the $16th$ highest price; no resale is permitted
- agents trade tickets to entertainment events among themselves in continuous double auctions

Flights and hotel reservations are complementary goods: flights are not useful to a client without the complementary hotel reservations; nor are hotel reservations useful to a client without the complementary flights. Tickets to entertainment events, e.g., the Boston Red Sox and the Boston Symphony Orchestra, are substitutable. Similarly, travel packages themselves are substitutes: e.g., arriving on Monday and departing on Tuesday vs. arriving on Monday and departing on Friday.

At a high-level, the design of many successful TAC agents (for example, Walverine [25], RoxyBot [46], and ATTac [98]) can be summarized as:

1. *predict*: build a model of the auctions' clearing prices
2. *optimize*: solve for an (approximately) optimal set of bids, given this model

This chapter is devoted to the study of the optimization piece of this design, which we model as the problem of bidding in "pseudo-auctions." We define a pseudo-auction as an idealized auction setting in which (i) there is only one bidder, and (ii) prices are specified by an exogenous model, that is, a model in which the bidder's price predictions are independent of its bidding strategy. Given such a model, the bidder faces the *bidding problem:* what is its utility-maximizing set of bids?

In the present chapter, we assume the agent builds a *deterministic* model of the auctions' prices: that is, there is no noise in the agent's price predictions; rather its predictions are point estimates. This assumption gives rise to the *deterministic* bidding problem. In our analysis, we focus on the deterministic *second-price* bidding problem, in which the payment rule is: "pay the predicted price." This problem is an abstraction of the problem of bidding in TAC Travel auctions, in which—under appropriate assumptions—agents can be viewed as price-takers.

Analytically, we study a set of heuristics in the context of a (single-unit) decision-theoretic bidding problem. Specifically, we derive the class of bidding heuristics that solves the deterministic second-price bidding

problem optimally. We also prove that the marginal-value-based bidding heuristic implemented in RoxyBot–2000 [46], and RoxyBot–2000*, a slight variant of RoxyBot–2000, are both instances of this class. The classic marginal value bidding heuristic itself, however, fails to solve this problem optimally in general, as noted in [45]. Nonetheless, we identify the special set of circumstances in which bidding marginal values is optimal.

Experimentally, we embed these heuristics in TAC Travel agents and evaluate their success in the (multi-unit) game-theoretic bidding problem that characterizes TAC Travel flight and hotel auctions.[2] We find that RoxyBot–2000's bidding heuristic dominates the others in our test set. Based on both our analytical and experimental results, we conclude that RoxyBot–2000's bidding heuristic is effective in that it performs well in a decision-theoretic setting when prices are given—equivalently, under the assumption of perfect price prediction—and it performs well in a game-theoretic setting where price predictions are imperfect.

It is not our contention that solutions to the decision-theoretic bidding problem are generally applicable as solutions to the game-theoretic bidding problem—the contrast between the conclusions of our analytical and experimental studies rule out this possibility.[3] We do believe, however, that the study of a related decision-theoretic bidding problem can inform the design of artificially intelligent agents that face a game-theoretic bidding problem.

8.2 Bidding in Simultaneous Pseudo-Auctions

In this chapter, we study a suite of heuristics for bidding in simultaneous auctions for complementary and substitutable goods. We develop these heuristics in the context of an optimization problem faced by an agent that is bidding in simultaneous "pseudo-auctions."

There are two defining features of pseudo-auctions: (i) there is only one bidder, and (ii) the auction's prices are specified (i.e., predicted) by an exogenous model, that is, a model in which the bidder's price predictions are independent of its bidding strategy.

The winner determination rule in a pseudo-auction is: "win by bidding at least the predicted price." In a first-price pseudo-auction the payment rule is "pay the winning bid price," whereas in a second-price pseudo-auction the payment rule is "pay the predicted price."

Implicit in our definitions of the first-price and second-price bidding problems is the assumption that these problems characterize the optimization problem faced by an agent bidding in first- and second-price "sealed-bid" pseudo-auctions.

Given an instance of a bidding problem, a *bidding heuristic* searches for a suitable $b \in \mathbb{R}^X$, that is, a function from a set of goods X to bids $b(x) \in \mathbb{R}$ (equivalently,[4] a bid vector).

The extension of a real-valued function $q : X \to \mathbb{R}$ on goods to a real-valued function $\tilde{q} : 2^X \to \mathbb{R}$ on bundles (i.e., sets of goods) is called *linear* if and only if $\tilde{q}(Y) = \sum_{x \in Y} q(x)$ for all $Y \subseteq X$.

[2] To reduce variance, we disregard entertainment tickets in our experiments.

[3] RoxyBot–2000* is optimal in our analytic framework, but is suboptimal in our experimental framework.

[4] If Z is finite, $\mathbb{R}^Z = \{f : Z \to \mathbb{R}\}$ is isomorphic to $\mathbb{R}^{|Z|}$.

Pseudo-Auction Winner Determination Rule Given a set of goods X and a pricing function $p : X \to \mathbb{R}$, $\mathrm{W}(X, p, b) \subseteq X$ is the set of goods the agent wins by bidding $b \in \mathbb{R}^X$: i.e.,

$$x \in \mathrm{W}(X, p, b) \text{ if and only if } b(x) \geq p(x) \tag{8.1}$$

Definition Given a set of goods X, a valuation function $v : 2^X \to \mathbb{R}$, a distribution f over pricing functions $p : X \to \mathbb{R}$, and a payment rule $\tilde{q}$, the (simultaneous) *bidding problem* is defined as follows:

$$\mathrm{SIM}(X, v, f) = \max_{b \in \mathbb{R}^X} \mathbb{E}_{p \sim f} \left[v(\mathrm{W}(X, p, b)) - \tilde{q}(\mathrm{W}(X, p, b)) \right]$$

The payment rule is $\tilde{q} = \tilde{b}$ in the *first-price bidding problem* and $\tilde{q} = \tilde{p}$ in the *second-price bidding problem*

The deterministic bidding problem is the special case of the bidding problem in which prices are certain.

Definition Given a set of goods X, a valuation function $v : 2^X \to \mathbb{R}$, and a pricing function $p : X \to \mathbb{R}$, the *deterministic bidding problem* is defined as follows:

$$\mathrm{DET}(X, v, p) = \max_{b \in \mathbb{R}^X} \left[v(\mathrm{W}(X, p, b)) - \tilde{q}(\mathrm{W}(X, p, b)) \right]$$

We refer to the deterministic first- and second-price bidding problems as *1stDET* and *2ndDET* respectively.

Since prices are fully specified in the deterministic bidding problem, the key decision an agent faces is which goods to buy. But the problem of deciding which goods to buy is "the acquisition problem" [44]. Indeed, the deterministic bidding problem reduces to the acquisition problem.

Definition Given a set of goods X, a valuation function $v : 2^X \to \mathbb{R}$, and a pricing function $q : 2^X \to \mathbb{R}$, the *acquisition problem* is defined as follows:

$$\mathrm{ACQ}(X, v, q) = \max_{Y \subseteq X} \left(v(Y) - q(Y) \right) \tag{8.2}$$

Theorem 2 *Given a set of goods X, a valuation function $v : 2^X \to \mathbb{R}$, and a pricing function $p : X \to \mathbb{R}$, the following bidding heuristic, which returns $b^* \in \mathbb{R}^X$, solves 2ndDET(X, v, p) optimally:*

1. *select an optimal acquisition $A^* \in \arg$ ACQ$(X, v, \tilde{p})$*
2. *bid*

$$b^*(x) \in \begin{cases} [p(x), \infty) & \text{if } x \in A^* \\ (-\infty, p(x)) & \text{otherwise} \end{cases} \tag{8.3}$$

In particular, first solving for an optimal acquisition A^* and then bidding $p(x)$ on all goods $x \in A^*$ and bidding $-\infty$ otherwise is an optimal heuristic in the deterministic second-price bidding problem. This heuristic is also optimal in the deterministic first-price bidding problem.

8.3 An Analysis of Marginal Values

In the preceding section, we derived an optimal class of bidding heuristics for the deterministic bidding problem. If an agent is clairvoyant—i.e., if it can predict the auctions' clearing prices perfectly—then it can bid using any heuristic in this class, whenever the acquisition problem is computationally feasible. The TAC Travel acquisition problem, for one, is NP-hard [44]. Even more egregious, typical agents are not clairvoyant. Hence, it may be reasonable for agents to employ alternative bidding heuristics.

In this section, we broaden our study of bidding heuristics by investigating two classic strategies: bidding independent and marginal values. We illustrate the performance of these heuristics (and some of the complexity of bidding in simultaneous auctions on complementary and substitutable goods) on a series of numerical examples. Also in this section, an analysis of marginal values leads to a "characterization theorem," which completely characterizes the relationship between marginal values and prices, assuming linear prices.

8.3.1 Bidding Independent and Marginal Values

Perhaps the most straightforward bidding heuristic is Heuristic IV, for independent values: For each good in each auction, bid its independent value. Unfortunately, with heuristic IV, an agent can fail to win goods it wishes it had won, when goods are complements, and can succeed at winning goods it wishes it had not won, when goods are substitutes.

Definition Given a set of goods X and a valuation function $v : 2^X \rightarrow \mathbb{R}$, the *independent value* of a good $x \in X$ is given by: $\iota(x) = v(\{x\})$.

Example Suppose an agent values a camera and flash together at 500, but values either good alone at 1. Also, suppose these two goods are sold separately in two simultaneous auctions, and the clearing prices are 200 for the camera and 100 for the flash. If the agent were to bid only its independent values (1), it would lose both goods, obtaining utility of 0 rather than $500 - 200 - 100 = 200$. This outcome is undesirable: the agent fails to win goods it wishes it had won.

Example Now suppose an agent values a Canon AE–1 at 300 and a Canon A–1 at 200, but values both cameras together at only 400. Also, suppose these two goods are sold separately in two simultaneous auctions, and the clearing prices are 275 for the AE–1 and 175 for the A–1. If the agent were to bid its independent values, it would win both goods, obtaining utility of $400 - 450 = -50$. This outcome is also undesirable: the agent wins goods it wishes it had not won.

A natural alternative to Heuristic IV is Heuristic MV, for marginal value: For each good, bid its *marginal* value. Unfortunately, even with heuristic MV, an agent can can succeed at winning goods it wishes it had not won—in particular, when goods are substitutes—although an MV agent never fails to win goods it wishes it had won (see Theorem 3).

Definition Given a set of goods X, a valuation function $v : 2^X \rightarrow \mathbb{R}$, and a pricing function $q : 2^X \rightarrow \mathbb{R}$. The *marginal value* $\mu(x) \equiv \mu(x, X, v, q)$ of good $x \in X$ is defined as follows:

$$\mu(x) = \max_{Y \subseteq X \setminus \{x\}} [v(Y \cup \{x\}) - q(Y)] - \max_{Y \subseteq X \setminus \{x\}} [v(Y) - q(Y)]$$

Intuitively, the marginal value of x is simply the difference between the value of an optimal acquisition when x costs 0, and the value of an optimal acquisition when x costs ∞.

Example Consider once again the setup of Example 8.3.1. Given both the camera and flash together, the agent's value is 500; but either one of these components without the other is valued at only 1. If the clearing prices of the camera and flash are 200 and 100, respectively, then bidding according to MV, $(500 - 100) - (0 - 0) = 400$ on the camera and $(500 - 200) - (0 - 0) = 300$ on the flash, the agent wins both goods, as desired.

Example Consider once again the setup of Example 8.3.1, where an agent values a Canon AE–1 at 300 and a Canon A–1 at 200, and both cameras together at 400. If the clearing prices of the camera and flash are 275 and 175, respectively, then bidding according to MV, $(300 - 0) - (200 - 175) = 275$ on the camera and $(200 - 0) - (300 - 275) = 175$ on the flash, the agent wins both goods. As in Example 8.3.1, this is not the desired outcome: the agent wins goods it wishes it had not won.

Example 8.3.1 shows that, for complementary goods, the MV heuristic can be an effective means of solving the deterministic bidding problem, in spite of the well-documented *exposure* problem. An agent suffers from its exposure if it bids more on a good than its independent value of that good [67, 90]. As noted here, an agent can also suffer from another kind of exposure in which it bids more on a set of goods than its combinatorial value of that set of goods. Indeed, in Example 8.3.1, which concerns substitutable goods, the MV heuristic suffers from its exposure.

Although the marginal value bidding heuristic does not solve the deterministic second-price bidding problem optimally in general, in what follows, we derive the special set of circumstances in which bidding marginal values is optimal. Our derivation follows from Theorem 2 and an immediate corollary of the following "characterization theorem."

8.3.2 Characterization Theorem

Throughout this section, we assume we are given a set of goods X, a valuation function $v : 2^X \to \mathbb{R}$, and a pricing function $p : X \to \mathbb{R}$. Our main theorem, which generalizes [43], completely characterizes the relationship between marginal values and prices, assuming linear prices. In words, this theorem states the following: if x is contained in an optimal acquisition, then either x is contained in all optimal acquisitions, in which case its marginal value is strictly greater than its price, or x is not contained in all optimal acquisitions, in which case its marginal value is exactly equal to its price (as in Example 8.3.1); otherwise, if x is not contained in any optimal acquisition, then its marginal value is strictly less than its price.

Theorem 3 *Assume prices are linear. If $A_1^*, \ldots, A_n^* \subseteq X$ are all the optimal solutions to the acquisition problem ACQ$(X, v, \tilde{p})$, then for all goods $x \in X$,*

1. *$\mu(x) > p(x)$ if and only if $x \in \bigcap_{i=1}^n A_i^*$*
2. *$\mu(x) = p(x)$ if and only if $x \in \bigcup_{i=1}^n A_i^*$ but $x \notin \bigcap_{i=1}^n A_i^*$*

3. $\mu(x) < p(x)$ *if and only if* $x \notin \bigcup_{i=1}^{n} A_i^*$

Our proof of Theorem 3 relies on the following observation:

Observation 1 *Assume prices are linear. The following equalities are equivalent: for all* $x \in X$*,*

$$\begin{aligned} & \mu(x) = p(x) \\ \textit{iff} \quad & \max_{Y \subseteq X \setminus x} [v(Y \cup \{x\}) - \tilde{p}(Y)] - \max_{Y \subseteq X \setminus x} [v(Y) - \tilde{p}(Y)] = p(x) \\ \textit{iff} \quad & \max_{Y \subseteq X \setminus x} [v(Y \cup \{x\}) - \tilde{p}(Y \cup \{x\})] = \max_{Y \subseteq X \setminus x} [v(Y) - \tilde{p}(Y)] \end{aligned}$$

The same holds when the equal signs are replaced by (weak or strict) inequality signs.

Proof of Theorem 3 [Claim 1]: The proof follows immediately from Obs. 1 and the following tautology: an arbitrary good $x \in X$ is included in all optimal acquisitions if and only if the value of an optimal acquisition with x necessarily included is strictly greater than the value of an optimal aquisition with x necessarily excluded: i.e.,

$$\begin{aligned} & \max_{Y \subseteq X \setminus x} [v(Y \cup \{x\}) - \tilde{p}(Y \cup \{x\})] > \max_{Y \subseteq X \setminus x} [v(Y) - \tilde{p}(Y)] \\ \text{iff} \quad & x \in \bigcap_{i=1}^{n} A_i^* \end{aligned}$$

■

Proof of Theorem 3 [Claim 2]: For all $Y \subseteq X$, define the utility $u(Y) = v(Y) - \tilde{p}(Y)$.

[$\Longrightarrow$] By assumption, $\mu(x) = p(x)$. Hence, by Obs. 1, $A_1 \cup \{x\}$ and A_2 have the same utility, where

$$A_1 \in \arg \max_{Y \subseteq X \setminus x} [v(Y \cup \{x\}) - \tilde{p}(Y \cup \{x\})] \tag{8.4}$$

$$A_2 \in \arg \max_{Y \subseteq X \setminus x} [v(Y) - \tilde{p}(Y)] \tag{8.5}$$

Note that $x \in A_1 \cup \{x\}$ but $x \notin A_2$. An optimal acquisition either contains x or it does not. Hence, one of $A_1 \cup \{x\}$ or A_2 must be optimal, but since they have the same utility, both of them are optimal. We have constructed an optimal acquisition with x and an optimal acquisition without x. Therefore, $x \in \bigcup_{i=1}^{n} A_i^*$ but $x \notin \bigcap_{i=1}^{n} A_i^*$.

[$\Longleftarrow$] By assumption, $x \in \bigcup_{i=1}^{n} A_i^*$ but $x \notin \bigcap_{i=1}^{n} A_i^*$. Hence, there exists $A_i^* \neq A_j^*$ such that $x \in A_i^*$ but $x \notin A_j^*$. Now

$$\begin{aligned} & x \in A_i^* \Rightarrow u(A_i^*) = \max_{Y \subseteq X \setminus x} [v(Y \cup \{x\}) - \tilde{p}(Y \cup \{x\})] \\ & x \notin A_j^* \Rightarrow u(A_j^*) = \max_{Y \subseteq X \setminus x} [v(Y) - \tilde{p}(Y)] \\ & u(A_i^*) = u(A_j^*) \quad \text{iff} \\ & \max_{Y \subseteq X \setminus x} [v(Y \cup \{x\}) - \tilde{p}(Y \cup \{x\})] = \max_{Y \subseteq X \setminus x} [v(Y) - \tilde{p}(Y)] \end{aligned}$$

This last equation is equivalent to $\mu(x) = p(x)$ by Obs. 1. ■

When the optimal acquisition is unique, the marginal value of a good is strictly greater than its price if and only if the good is in the optimal acquisition; otherwise, the marginal value of the good is strictly less than its price. This corollary of Theorem 3 is immediate.

Corollary 1 *Assume prices are linear. If $A^* \subseteq X$ is the* unique *solution to the acquisition problem ACQ$(X, v, \tilde{p})$, then*

1. $\mu(x) > p(x)$ *if and only if* $x \in A^*$*, and*
2. $\mu(x) < p(x)$ *if and only if* $x \notin A^*$*.*

Finally, we characterize the relationship between $\mu(x, A^*)$ and $\mu(x, X)$, that is, the marginal utility of a good x relative to an optimal acquisition A^* vs. the marginal utility of a good x relative to the set of all goods X. Intuitively, the marginal value of a good that is in an optimal acquisition cannot decrease if the set of available goods is restricted to include precisely the goods in that acquistion. Analogously, the marginal value of a good that is not in an optimal acquisition cannot increase if the set of available goods is restricted to include precisely the goods in that acquistion.

Proposition 2 *If $A^* \subseteq X$ is an optimal solution to the acquisition problem ACQ(X, v, q), then*

- $\mu(x, A^*) \geq \mu(x, X)$*, for all* $x \in A^*$*, and*
- $\mu(x, A^*) \leq \mu(x, X)$*, for all* $x \notin A^*$ *(i.e.,* $x \in X \setminus A^*$*),*

where $q : 2^X \rightarrow \mathbb{R}$ is an arbitrary pricing function.

8.4 A Test Suite of Bidding Heuristics

We now articulate the inner workings of four select bidding heuristics. StraightMV is an implementation of the marginal value bidding heuristic. SecondBot generalizes the class of bidding heuristics that solves the deterministic second-price bidding problem optimally. FirstBot, a bidding heuristic that solves the deterministic *first*-price bidding problem optimally, RoxyBot–2000, and a slight variant, RoxyBot–2000*, are all instances of SecondBot.

We argue that FirstBot, RoxyBot–2000, and RoxyBot–2000* all solve the deterministic second-price bidding problem optimally, assuming linear prices. We also establish that StraightMV is optimal whenever the solution to the acquisition problem is unique, again, assuming prices are linear.

Also in this section, we work through an example of the deterministic second-price bidding problem, and compare the performance of these four heuristics on this problem *without assuming clairvoyance*—that is, the agents optimize with respect to imperfect price predictions.

StraightMV is an implementation of the classic marginal value bidding heuristic. It bids the marginal value of each good in each auction, given as input predictions of the auctions' clearing prices. StraightMV calculates $|X|$ marginal values; hence, it solves $2|X|$ acquisition problems.

Theorem 4 *Bidding marginal values is optimal in the deterministic second-price bidding problem whenever the solution to the acquisition problem is unique, assuming prices are linear.*

Proof The proof follows immediately from Theorem 2 and Corollary 1.

Example A (TAC) travel agent is deciding what to bid on hotels for a client for whom it has already purchased flights. The client's willingness to pay for travel packages including the good and bad hotel, respectively, are 1055 and 1000. Flights alone are worthless to the client.

Suppose the agent predicts the clearing price of the good hotel to be 80 and the clearing price of the bad hotel to be 30, while in reality, the clearing price of the good and bad hotels are uniformly distributed in the ranges $[70, 90]$ and $[20, 40]$, respectively.

Given its predictions, the marginal value of the good hotel is $(1055 - 0) - (1000 - 30) = 85$, while the marginal value of the bad hotel is $(1000 - 0) - (1055 - 80) = 25$. StraightMV bids precisely these marginal values: 85 on the good hotel and 25 on the bad hotel.

By bidding marginal values, a StraightMV agent is likely to win either too many substitutes or too few complements. The probability of winning the good hotel is $\frac{85-70}{90-70} = 0.75$, while the probability of winning the bad hotel is $\frac{25-20}{40-20} = 0.25$. Consequently, the probability of winning too many substitutes (both hotels) is $(.75)(.25) = .1875$, as is the probability of winning too few complements (neither hotel).

Whereas StraightMV can win too many substitutes assuming either perfect or imperfection price prediction, none of the following three heuristics ever wins too many substitutes. Moreover, whereas StraightMV can win too few complements assuming imperfect price prediction, in turn, each of the next three heuristics wins more and more complements.

SecondBot SecondBot first solves for an optimal acquisition $A^* \in \arg \text{ACQ}(X, v, \tilde{p})$, and then bids on the goods in A^* according to some function g. We study three instances of SecondBot, corresponding to three choices of the bid function $g(x) \equiv g(x, X, v, p, A^*)$.

- FirstBot: $g = p$
- RoxyBot–2000: $g = h$ where $h(x) = \mu(x, X, v, p)$, for $x \in X$
- RoxyBot–2000*: $g = h^*$ where $h^*(x) = \mu(x, A^*, v, p)$, for $x \in X$

Note the distinction between RoxyBot–2000 and RoxyBot–2000*: the former calculates marginal values with respect to the set of goods X, whereas the latter calculates marginal values with respect to the optimal acquisition A^*.

These three instances of SecondBot place progressively higher and higher bids:

- FirstBot bids $p(x)$ on all goods $x \in A^*$
- RoxyBot–2000 bids $\mu(x, X, v, p) \geq p(x)$ on all goods $x \in A^*$ (by Theorem 3)
- RoxyBot–2000* bids $\mu(x, A^*, v, p) \geq \mu(x, X, v, p)$ on all goods $x \in A^*$ (by Proposition 2)

FirstBot solves only 1 acquisition problem. In the worst case (when $A^* = X$), these versions of RoxyBot calculate $|X|$ marginal values, solving $2|X|+1$ acquisition problems in total. In practice (e.g., in TAC Travel games), however, they calculate far fewer marginal values than StraightMV.

Theorem 5 *FirstBot, RoxyBot–2000, and RoxyBot–2000* are optimal bidding heuristics in the deterministic second-price bidding problem, assuming prices are linear.*

Proof The proof follows immediately from Theorem 2, Theorem 3, and Proposition 2.

Example Since SecondBot bids only on the goods in a single acquisition, it cannot win too many substitutes, but still it may win too few complements. Continuing Example 8.4, the values of the travel packages with the good and bad hotels are $1055 - 80 = 975$ and $1000 - 30 = 970$, respectively. Hence, the travel package with the good hotel is the unique optimal acquisition.

FirstBot bids the predicted price (80) on the good hotel and nothing on the bad hotel, which yields a 50% chance of winning too few complements (i.e., losing both hotels). RoxyBot–2000 bids its marginal value (85) on the good hotel and nothing on the bad hotel, which yields a 25% chance of winning too few complements.

RoxyBot–2000* assumes the bad hotel is not available. Under this assumption, the marginal value of the good hotel is $1055 - 0 = 1055$. This is the only bid RoxyBot–2000* submits. Since $1055 \geq 90$, the upper bound on the clearing price of the good hotel, RoxyBot–2000* wins neither too many substitutes nor too few complements in this example.

8.5 Bidding in TAC Travel Auctions

Having conducted an analytic study of the deterministic second-price bidding problem, and, in doing so, having developed a test suite of bidding heuristics, we now describe an experimental study in which we embedded these heuristics in TAC Travel agents and played TAC Travel games. There are three key differences between our analytical and experimental setups:

- the former is decision-theoretic, while the latter is game-theoretic
- the former is a single-unit second-price design, while the latter is a k-unit kth price design (NB: if $k = 1$, the latter is a first-price design rather than a second-price design)
- in the former, prices are given, which amounts to an assumption of perfect price prediction by an agent; in the latter, an agent's price predictions are imperfect

Still, we conducted these experiments to shed some light on the efficacy of our test suite of bidding heuristics, which are optimal or near-optimal in a decision-theoretic problem, in a related game-theoretic bidding problem. It is not our contention, however, that solutions to a decision-theoretic bidding problem are generally applicable as solutions to a game-theoretic bidding problem.

8.5.1 Experimental Setup

In our experiments, we embedded in TAC Travel agents our test suite of bidding heuristics, generalized to bid marginal values in multi-unit auctions. These agents played numerous TAC Travel games, bidding only in flight and hotel auctions. We disregarded entertainment ticket auctions to reduce the variance in the agents' scores. This feature of our experimental design was implemented by modifying the TAC–2004 Classic Java Server [33].

Recall that there are two key architectural components of TAC Travel agents, price prediction and optimization. In our experimental design, we vary the optimization component of the agents, but we fix the price prediction component. The tâtonnement process is a method of computing competitive equilibrium prices in a market [107]. All the agents in our experiments predict the hotel auctions' clearing prices via the tâtonnement process, just as it was implemented in Walverine–2003 [25].

8.5.2 Experimental Results

In this section, we report the results of two experiments with our test suite of bidding heuristics in TAC Travel games. For both experiments, we report the average score earned by each agent, along with the corresponding average utilities and costs. In addition, we report the results of paired t-tests, in which we consider pairings of the agents' scores, utilities, and costs. In our statistical tests, the null hypothesis is: "there is no difference between the means."

4 Agent Experiment In our first set of experimental games (224 of them), we pitted two copies of each of our four bidding agents (StraightMV, RoxyBot–2000, RoxyBot–2000*, and FirstBot) against one another. The agents' average scores, utilities, and costs in these games are shown in Table 8.1. Scorewise, RoxyBot–2000 ever so slightly outperforms RoxyBot–2000*, who outperforms StraightMV, who substantially outperforms FirstBot. The difference in the mean scores earned by RoxyBot–2000 and RoxyBot–2000* is statistically insignificant, but all other differences are statistically significant. It is interesting to note that StraightMV obtains a higher utility on average than either variant of RoxyBot ($\sim$40), but it does so at a substantially higher cost (105–110). StraightMV bids on all goods, not only the goods in a single optimal acquisition. "You've got to be in it to win it." These differences are statistically significant. Finally, FirstBot is unsuccessful because it places too many losing bids, evidenced by its unusually low cost.

Rank	Agent	Avg Score	Avg Utility	Avg Cost
1	RoxyBot–2000	2738.275	8271.348	5533.074
2	RoxyBot–2000*	2731.678	8270.230	5538.552
3	StraightMV	2667.645	8310.908	5643.263
4	FirstBot	1998.806	7476.808	5478.002

Table 8.1: 4 Agent Experiment: Average Scores, Utilities, and Costs (448 Observations).

2 Agent Experiment In our second set of experimental games (205 of them), we pitted four copies of RoxyBot–2000 and RoxyBot–2000* against one another. The results of these games are depicted in Table 8.2. This

time, when RoxyBot–2000 outperforms RoxyBot–2000*, the differences between the mean scores, utilities, and costs are statistically significant (all three p-values are less than 0.001).

This outcome can be explained as follows. Embedded in both RoxyBot–2000 and RoxyBot–2000* are optimal bidding heuristics for the deterministic second-price bidding problem. The TAC Travel hotel bidding problem is not a second-price auction, however; it is not even a $k+1$st-price k-unit auction. On the contrary, it is a kth-price k-unit auction. As such, it mimics a first-price auction, in which bidders are not price-takers. Rather, a winning bidder pays what it bids, and thus is incentivized to shade its bids downwards. But RoxyBot–2000* shades its bids *upwards*! In doing so, it wins more goods than RoxyBot–2000, so that it obtains a higher utility than RoxyBot–2000 (103.184), but this increase in utility is achieved at a substantially higher cost (341.821).

This 2 agent experiment serves to highlight one of the key differences between decision-theoretic bidding in pseudo-auctions and game-theoretic bidding in auctions. RoxyBot–2000* is an optimal bidding heuristic in the former, but it is suboptimal in the latter. We conclude that RoxyBot–2000 is the dominant TAC Travel agent in our test suite.

Rank	Agent	Avg Score	Avg Utility	Avg Cost
1	RoxyBot–2000	2298.145	8137.993	5839.848
2	RoxyBot–2000*	2059.508	8241.177	6181.669

Table 8.2: 2 Agent Experiment: Average Scores, Utilities, and Costs (820 Observations).

8.6 Conclusion

Based on both our analytical and experimental results, we conclude that RoxyBot–2000's bidding heuristic is effective in that it performs well in a decision-theoretic setting when prices are given—equivalently, under the assumption of perfect price prediction—and it performs well in a game-theoretic setting where price predictions are imperfect. However, this bidding heuristic, which operates on (deterministic) price point estimates, does not explicitly plan for uncertainty in the auction dynamics. A heuristic that would be superior in this respect would optimize with respect to noisy (i.e., stochastic) models of estimated clearing prices. Indeed, embedded in RoxyBot–2006, the top-scoring agent in TAC–2006, is such a bidding heuristic.

Chapter 9

Bidding in TAC Travel[1]

In this chapter, we describe our entrant in the travel division of the 2006 Trading Agent Competition (TAC). At a high level, the design of many successful autonomous trading agents can be summarized as follows: (i) price prediction: build a model of market prices; and (ii) optimization: solve for an approximately optimal set of bids, given this model. To predict, we simulate *simultaneous ascending auctions*. To optimize, we apply the *sample average approximation* method. abbreviated SAA; hence the title of this paper. Our agent dominated the preliminary and seeding rounds of TAC Travel in 2006, and emerged as champion in the finals in a photo finish.

9.1 Introduction

The annual Trading Agent Competition (TAC) challenges its entrants to design and build autonomous bidding agents capable of effective trading in an online travel[2] shopping game. The first TAC, held in Boston in 2000, attracted 16 entrants from six countries in North America, Europe, and Asia. Excitement generated from this event led to refinement of the game rules, and continuation of regular tournaments with increasing levels of competition over the next six years. Year-by-year, entrants improved their designs, developing new ideas and building on previously successful techniques. Since TAC's inception, the lead author has entered successive modifications of her autonomous trading agent, RoxyBot. This chapter reports on RoxyBot-06, the latest incarnation and the top scorer in the TAC-06 tournament.

The key feature captured by the TAC travel game is that goods are highly interdependent (e.g., flights and hotels must be coordinated), yet the markets for these goods operate independently. A second important feature of TAC is that agents trade via three different kinds of market mechanisms, each of which presents distinct challenges. Flights are traded in a posted-price environment, where a designated party sets a price that the other parties must "take or leave." Hotels are traded in simultaneous ascending auctions, like the FCC

[1] Joint work with Amy Greenwald and Seong Jae Lee.

[2] At present, there are two divisions of TAC: Travel and Supply Chain Management. This chapter is concerned only with the former; for a description of the latter, see Arunachalam and Sadeh [5]. In this chapter, when we say TAC, we mean TAC Travel.

spectrum auctions. Entertainment tickets are traded in continuous double auctions, like the New York Stock Exchange. In grappling with all three mechanisms while constructing their agent strategies, participants are confronted by a number of interesting problems.

The success of an autonomous trading agent, particularly TAC agents, often hinges upon two key modules: (i) *price prediction*, in which the agent builds a model of market prices; and (ii) *optimization*, in which the agent solves for an approximately optimal set of bids, given this model. For example, at the core of RoxyBot's 2000 architecture [46] was a *deterministic* optimization problem, namely how to bid given price predictions in the form of point estimates. In spite of its effectiveness in the TAC-00 tournament, a weakness of the 2000 design was that RoxyBot could not explicitly reason about variance within prices. In the years since 2000, we recast the key challenges faced by TAC agents as several different *stochastic* bidding problems (see, for example, Greenwald and Boyan [45]), whose solutions exploit price predictions in the form of distributions. In spite of our perseverance, RoxyBot fared unimpressively in tournament conditions year after year... until 2006. Half a decade in the laboratory spent searching for bidding heuristics that can exploit stochastic information at reasonable computational expense finally bore fruit, as RoxyBot emerged victorious in TAC-06. In a nutshell, the secret of RoxyBot-06's success is: (hotel) price prediction by simulating simultaneous ascending auctions, and optimization based on the sample average approximation method. Details of this approach are the subject of the present article.

Overview This chapter is organized as follows. Starting in Section 9.2, we summarize the TAC market game. Next, in Section 9.3, we present a high-level view of RoxyBot's 2006 architecture. This design is grounded in two key assumptions: fixed other-agent behaviors and market information encapsulated by prices. In Section 9.4, we describe RoxyBot's optimization technique, the sample average approximation method. We argue that it is optimal in pseudo-auctions, an abstract model of auctions characterized by the aforementioned assumptions. Implementation details are relegated to Appendix 9.10. In Section 9.5, we describe RoxyBot's price prediction techniques for flights, hotels, and entertainment, in turn. Our hotel price prediction method is perhaps of greatest interest. Following Wellman *et al.* [25], we take as our hotel price predictions approximate competitive equilibrium prices. Only, instead of computing those prices by running the tâtonnement process, we simulate simultaneous ascending auctions. We show that the latter computation is faster, and does not sacrifice accuracy. In Section 9.6, we detail the results of the TAC-06 tournament, reporting statistics that shed light on the bidding strategies of the participating agents. Finally, in Section 9.7, we evaluate the collective behavior of the autonomous agents in the TAC finals since 2002. We find that the accuracy of competitive equilibrium calculations has varied from year to year and is highly dependent on the particular agent pool. Still, generally speaking, the collective is moving toward competitive equilibrium behavior.

9.2 TAC Market Game: A Brief Summary

In this section, we briefly summarize the TAC game. For more details, see `http://www.sics.se/tac/`.

Eight agents play the TAC game. Each is a simulated travel agent whose task is to organize itineraries for

its clients to travel to and from "TACTown" during a five day (four night) period. In the time allotted (nine minutes), each agent's objective is to procure travel goods as inexpensively as possible, trading off against the fact that those goods are ultimately compiled into feasible trips that satisfy its client preferences to the greatest extent possible. The agents know the preferences of their own eight clients only, not the other 56.

Travel goods are sold in simultaneous auctions as follows:

- Flight tickets are sold by "TACAir" in dynamic posted-pricing environments. There are flights both to and from TACTown on each applicable day. No resale of flight tickets by agents is permitted.

  Flight price quotes are broadcast by the TAC server every ten seconds.

- Hotel reservations are sold by the "TAC seller" in multi-unit ascending call markets. Specifically, 16 hotel reservations are sold in each hotel auction to the 16 highest bidders at the $16th$ highest price. There are two hotels: a good one and a bad one. No resale of hotel reservations by agents is permitted. Nor is bid withdrawal allowed.

  More specifically, the eight hotel auctions clear on the minute with exactly one auction closing at each of minutes one through eight. (The precise auction to close is chosen at random, with all open auctions equally likely to be selected.) For the auction that closes, the TAC server broadcasts the final closing price, and informs each agent of its winnings. For the others, the TAC server reports the current ask price, and informs each agent of its "hypothetical quantity won" (HQW).

- Agents are allocated an initial endowment of entertainment tickets, which they trade among themselves in continuous double auctions (CDAs). There are three entertainment events scheduled each day.

  Although the event auctions clear continuously, price quotes are broadcast only every 30 seconds.

One of the primary challenges posed by TAC is to design and build autonomous agents that bid effectively on interdependent (i.e., complementary or substitutable) goods that are sold in separate markets. Flight tickets and hotel reservations are complementary because flights are not useful to a client without the corresponding hotel reservations, nor vice versa. Tickets to entertainment events (e.g., the Boston Red Sox and the Boston Symphony Orchestra) are substitutable because a client cannot attend multiple events simultaneously.

9.3 RoxyBot-06's Architecture: A High-Level View

In our approach to the problem of bidding on interdependent goods in the separate TAC markets, we adopt some simplifying assumptions. Rather than tackle the game-theoretic problem of characterizing strategic equilibria, we focus on a single agent's (decision-theoretic) problem of optimizing its own bidding behavior, assuming the other agents' strategies are fixed. In addition, we assume that the environment can be modeled in terms of the agent's predictions about market clearing prices. These prices serve to summarize the relevant information hidden in other agents' bidding strategies. These two assumptions—fixed other-agent behaviors and market information encapsulated by prices—support the modular design of RoxyBot-06 and many other successful TAC agents, which consists of two key stages: (i) price prediction; and (ii) optimization.

REPEAT

{*start bid interval*}

0. Download current prices and winnings from server
1. *predict*: build stochastic models
   a. *flights*: Bayesian updating/learning
   b. *hotels*: simultaneous ascending auctions
   c. *entertainment*: sample historical data
2. *optimize*: sample average approximation
3. Upload current bids to server
   (three separate threads)

{*end bid interval*}

UNTIL game over

Table 9.1: A high-level view of RoxyBot-06's architecture.

The optimization problem faced by TAC agents is a dynamic one that incorporates aspects of sequentiality as well as simultaneity in auctions. The markets operate simultaneously, but in addition, prices are discovered incrementally over time. In principle, a clairvoyant agent—one with knowledge of future clearing prices—could justifiably employ an open-loop strategy: it could solve the TAC optimization problem once at the start of the game and place all its bids accordingly, never reconsidering those decisions. A more practical alternative (and the usual approach taken in TAC[3]), is to incorporate into an agent's architecture a closed loop, or *bidding cycle*, enabling the agent to condition its behavior on the evolution of prices. As price information is revealed, the agent improves its price predictions, and reoptimizes its bidding decisions, repeatedly.

One distinguishing feature of RoxyBot-06 is that it builds stochastic models of market clearing prices, rather than predicting clearing prices as point estimates. Given its stochastic price predictions, stochastic optimization lies at the heart of RoxyBot-06. Assuming time is discretized into stages, or bid intervals, during each iteration of its bidding cycle, RoxyBot-06 faces an n-stage stochastic optimization problem, where n is the number of stages remaining in the game. The key input to this optimization problem is a sequence of $n-1$ stochastic models of future prices (current prices are known), each one a joint probability distribution over all goods conditioned on past prices. The solution to this optimization problem, and the output of each iteration of the bidding cycle, is a vector of bids, one per good (or auction).

Table 9.1 presents a high-level view of RoxyBot-06's architecture, emphasizing its bidding cycle. At the start of each bid interval, current prices and winnings are downloaded from the TAC server. Next, the key prediction and optimization routines are run. In the prediction module, stochastic models of flight, hotel, and entertainment prices are built. In the optimization module, bids are constructed as an approximate solution to an n-stage stochastic optimization problem. Prior to the end of each bid interval, the agents' bids are uploaded to the TAC server using three separate threads: (i) the flight thread bids on a flight only if its price is near its predicted minimum; (ii) the hotel thread bids on open hotels only if it is moments before the end of a minute; and (iii) the entertainment thread places bids immediately.

[3] An exception is livingagents [36], the winner of TAC 2001.

We discuss the details of RoxyBot-06's optimization module first, and its price prediction module second.

9.4 Optimization

We characterize RoxyBot-06's optimization routine as (i) stochastic, (ii) global, and (iii) dynamic. It takes as input stochastic price predictions; it simultaneously considers flight, hotel, and entertainment bids in unison; and it simultaneously reasons about bids to be placed in both current and future stages of the game.

9.4.1 TAC Market Mechanisms

Before discussing the specifics of RoxyBot-06's approach to optimizing its bids, we discuss some of the issues that can impact bidding decisions in each of the three TAC markets based on their respective mechanisms.

Flights Recall that flights are sold by TACAir in dynamic posted-pricing environments. The hallmark of a posted-price mechanism is that a designated party sets a price that the other parties can "take or leave." However, posted-price markets may differ regarding limits on quantity, the manner or frequency by which prices change, or other features. In the case of TAC flight markets, TACAir posts sell prices for an effectively unlimited quantity of flights to and from TACTown on each day, and TAC agents can purchase any number of flights by submitting bids at or above those prices. TACAir updates prices every ten seconds according to a known stochastic process (see Section 9.5.1). This process has both revealed and hidden state, but is not affected in any way by the TAC agents' actions (i.e., flight purchases).

The fundamental issue regarding TAC flight decisions is a common one: balancing concern about future price increases with the benefit of delaying commitment to travel on particular days. If flight prices were non-increasing, agents would simply delay their flight purchases until the end of the game, when all uncertainty about hotel markets (i.e., what reservations each agent procures) has been resolved. By committing to a flight any earlier, an agent risks finding that its choice was suboptimal, based on subsequent shifts in hotel prices or availability. An extreme (but not unusual) instance of this risk is that it may end up wasting the flight entirely, if it cannot obtain hotel rooms to compile a feasible trip on the corresponding days. The dynamic aspect of RoxyBot-06's optimization module allows the agent to seemlessly reason about these tradeoffs.

Hotels In a *simultaneous ascending auction* (SimAA) [27], goods are sold to agents through an array of ascending auctions, one for each good. The auctions proceed concurrently, and bidding is organized in rounds. At any given time, the price quote is defined to be the highest bid received thus far, or zero if there are no bids as of yet. The ask price is the price quote plus a fixed increment. To be admissible, a new bid must beat the quote by offering at least the ask price. If an auction receives multiple admissible bids in a given round, it admits the highest (breaking ties arbitrarily). An auction is quiescent when a round passes with no new admissible bids. When all are simultaneously quiescent, the auctions close and their respective goods are allotted as per the last admitted bids.

The TAC hotel auctions run simultaneously, but differ from the abstract SimAA mechanism in two basic

ways. First, each TAC hotel auction is multi-unit: the top k bidders are allocated the k units, and the clearing price is the lowest winning bid. Second, rather than wait until all auctions are quiescent, one randomly selected auction closes each minute. In this respect TAC hotel auctions represent a hybrid between simultaneous and sequential auctions. Still, TAC hotel auctions and SimAAs share many of the same characteristics. Most importantly, because no good in a SimAA is committed until all are, an agent's bidding strategy in one auction cannot be contingent on the outcome in another. Similarly, an agent bidding for a set of hotels on contiguous days runs the risk that it will win some but not all hotels it desires.

Entertainment Most TAC designers treat entertainment trading as a task only loosely coupled to flight and hotel bidding. The markets are clearly interdependent, as the value of an entertainment ticket depends on what other tickets the agent holds, as well as the possible travel packages it can assemble for its clients. The relationship is relatively weak, however, since entertainment merely provides bonus utility; it does not affect trip feasibility like flights and hotels. Often a ticket not used for one client can be given to another, or sold to another agent. Entertainment markets are open throughout the game, and are not subject to time-dependent price movements or rigid clearing schedules like the other markets. Nonetheless, RoxyBot-06 makes its entertainment-bidding decisions in conjunction with its flight and hotel-bidding decisions, all within its global optimization module. The effectiveness of this approach is noted in Section 9.6.

9.4.2 Abstract Auction Model

Recall that our treatment of bidding is decision-theoretic, rather than game-theoretic. In particular, we focus on a single agent's problem of optimizing its own bidding behavior, assuming the other agents' strategies are fixed. In keeping with our basic agent architecture, we further assume that the environment can be modeled in terms of the agent's predictions about market clearing prices. We introduce the term *pseudo-auction* to refer to a market mechanism defined by these two assumptions—fixed other-agent behaviors and market information encapsulated by prices. The optimization problem that RoxyBot solves is one of bidding in pseudo-auctions, not (true) auctions. In this section, we formally develop this abstract auction model and relate it to TAC auctions; in the next, we define and propose heuristics to solve various pseudo-auction bidding problems.

Basic Formalism

In this section, we formalize the basic concepts needed to precisely formulate bidding under uncertainty as an optimization problem, including: packages—sets of goods, possibly multiple units of each; a function that describes how much the agent values each package; pricelines—data structures in which to store the prices of each unit of each good; and bids—pairs of vectors corresponding to buy and sell offers.

Packages Let G denote an ordered set of n distinct goods and let $N \in \mathbb{N}^n$ represent the multiset of these goods in the marketplace, with N_g denoting the number of units of each good $g \in G$. A *package* M is a collection of goods, that is, a "submultiset" of N. We write $M \subseteq N$ whenever $M_g \leq N_g$ for all $g \in G$.

It is instructive to interpret this notation in the TAC domain. The flights, hotel rooms, and entertainment events up for auction in TAC comprise an ordered set of 28 distinct goods. In principle, the multiset of goods

in the TAC marketplace is:

$$N^{\text{TAC}} = \langle \underbrace{\infty, \ldots, \infty}_{\text{8 flights}}, \underbrace{16, \ldots, 16}_{\text{8 hotels}}, \underbrace{8, \ldots, 8}_{\text{12 events}} \rangle \in \mathbb{N}^{28}$$

In practice, however, since each agent works to satisfy the preferences of only eight clients, it suffices to consider the multiset of goods:

$$N^{\text{TAC8}} = \langle \underbrace{8 \ldots, 8}_{\text{8 flights}}, \underbrace{8, \ldots, 8}_{\text{8 hotels}}, \underbrace{8, \ldots, 8}_{\text{12 events}} \rangle \subseteq N^{\text{TAC}}$$

A trip corresponds to a package, specifically some $M \subseteq N^{\text{TAC8}}$ that satisfies the TAC feasibility constraints.

Given $A, B \subseteq N$, we rely on the following basic operations: for all $g \in G$,

$$(A \oplus B)_g \equiv A_g + B_g$$
$$(A \ominus B)_g \equiv A_g - B_g$$

For example, if $G = \{\alpha, \beta, \gamma\}$ and $N = \langle 1, 2, 3 \rangle$, then $A = \langle 0, 1, 2 \rangle \subseteq N$ and $B = \langle 1, 1, 1 \rangle \subseteq N$. Moreover, $(A \oplus B)_\alpha = 1$, $(A \oplus B)_\beta = 2$, and $(A \oplus B)_\gamma = 3$; and $(A \ominus B)_\alpha = -1$, $(A \ominus B)_\beta = 0$, and $(A \ominus B)_\gamma = 1$.

Value Let $\mathcal{N}$ denote the set of all submultisets of N: i.e., packages comprised of the goods in N. We denote $v : \mathcal{N} \to \mathbb{R}$ a function that describes the value the bidding agent attributes to each viable package.

In TAC, each agent's objective is to compile packages for $m = 8$ individual clients. As such, the agent's value function takes special form. Each client c is characterized by its own value function $v_c : \mathcal{N} \to \mathbb{R}$, and the agent's value for a collection of packages is the sum of its clients' respective values for those packages: given a vector of packages $\vec{X} = (X_1, \ldots, X_m)$,

$$v(\vec{X}) = \sum_{c=1}^{m} v_c(X_c). \tag{9.1}$$

Pricelines A *buyer priceline* for good g is a vector $\vec{p}_g \in \mathbb{R}_+^{N_g}$, where the kth component, p_{gk}, stores the *marginal cost* to the agent of acquiring the kth unit of good g. For example, if an agent currently holds four units of a good $\tilde{g}$, and if four additional units of $\tilde{g}$ are available at costs of \$25, \$40, \$65, and \$100, then the corresponding buyer priceline (a vector of length 8) is given by $\vec{p}_{\tilde{g}} = \langle 0, 0, 0, 0, 25, 40, 65, 100 \rangle$. The leading zeros indicate that the four goods the agent holds may be "acquired" at no cost. We assume buyer pricelines are nondecreasing. Given a set of buyer pricelines $P = \{\vec{p}_g \mid g \in G\}$, we define costs additively, that is, the *cost* of the goods in multiset $Y \subseteq N$ is given by:

$$\begin{aligned} \forall g, \quad \text{Cost}_g(Y, P) &= \sum_{k=1}^{Y_g} p_{gk}, \\ \text{Cost}(Y, P) &= \sum_{g \in G} \text{Cost}_g(Y, P). \end{aligned} \tag{9.2}$$

A *seller priceline* for good g is a vector $\vec{\pi}_g \in \mathbb{R}_+^{N_g}$. Much like a buyer priceline, the kth component of a seller priceline for g stores the *marginal revenue* that an agent could earn from the kth unit it sells. For

example, if the market demands four units of good $\tilde{g}$, which can be sold at prices of \$20, \$15, \$10, and \$5, then the corresponding seller priceline is given by $\vec{\pi}_{\tilde{g}} = \langle 20, 15, 10, 5, 0, 0, 0, 0 \rangle$. Analogously to buyer pricelines, the tail of zero revenues indicates that the market demands only four of those units. We assume seller pricelines are nonincreasing. Given a set of seller pricelines $\Pi = \{\vec{\pi}_g \mid g \in G\}$, we define revenue additively, that is, the *revenue* associated with multiset $Z \subseteq N$ is given by:

$$\forall g, \quad \text{Revenue}_g(Z, \Pi) = \sum_{k=1}^{Z_g} \pi_{gk}, \tag{9.3}$$

$$\text{Revenue}(Z, \Pi) = \sum_{g \in G} \text{Revenue}_g(Z, \Pi). \tag{9.4}$$

If a priceline is constant, we say that prices are *linear*. We refer to the constant value as a *unit price*. With linear prices, the cost of acquiring k units of good g is k times the unit price of good g.

Bids An agent submits a bid β expressing offers to buy or sell various units of the goods in the marketplace. We divide β into two components $\langle \vec{b}, \vec{a} \rangle$, where for each good g the bid consists of a *buy offer*, $\vec{b}_g = \langle b_{g1}, \ldots, b_{gN_g} \rangle$, and a *sell offer*, $\vec{a}_g = \langle a_{g1}, \ldots, a_{gN_g} \rangle$. The bid price $b_{gk} \in \mathbb{R}_+$ (resp. $a_{gk} \in \mathbb{R}_+$) represents an offer to buy (sell) the kth unit of good g at that price.

By definition, the agent cannot buy (sell) the kth unit unless it also buys (sells) units $1, \ldots, k-1$. To accommodate this fact, we impose the following constraint: Buy offers must be nonincreasing in k, and sell offers nondecreasing. In addition, an agent may not offer to sell a good for less than the price at which it is willing to buy that good: i.e., $b_{g1} < a_{g1}$. Otherwise, it would simultaneously buy and sell good g. We refer to these restrictions as *bid monotonicity* constraints.

Pseudo-Auction Rules

Equipped with this formalism, we can specify the rules that govern pseudo-auctions. As in a true auction, the outcome of a pseudo-auction dictates the quantity of each good to exchange, and at what prices, conditional on the agent's bid. The quantity issue is resolved by the *winner determination rule* whereas the price issue is resolved by the *payment rule*.

Pseudo-Auction Winner Determination Rule Given buyer and seller pricelines P and Π, and bid $\beta = \langle \vec{b}, \vec{a} \rangle$, the agent buys the multiset of goods $\text{Buy}(\beta, P)$ and sells the multiset of goods $\text{Sell}(\beta, \Pi)$, where

$$\text{Buy}_g(\beta, P) = \max_k k \text{ such that } b_{gk} \geq p_{gk}$$

$$\text{Sell}_g(\beta, \Pi) = \max_k k \text{ such that } a_{gk} \leq \pi_{gk}$$

Note that the monotonicity restrictions on bids ensure that the agent's offer is better than or equal to the price for every unit it exchanges, and that the agent does not simultaneously buy and sell any good.

There are at least two alternative payment rules an agent may face. In a *first-price pseudo-auction*, the agent pays its bid price (for buy offers, or is paid its bid price for sell offers) for each good it wins. In a *second-price pseudo-auction*, the agent pays (or is paid) the prevailing prices, as specified by the realized

buyer and seller pricelines. This terminology derives by analogy from the standard first- and second-price sealed bid auctions [64, 106]. In these mechanisms, the high bidder for a single item pays its bid (the first price), or the highest losing bid (the second price), respectively. The salient property is that in first-price pseudo-auctions, the price is set by the bid of the winner, whereas in second-price pseudo-auctions an agent's bid price determines whether or not it wins but not the price it pays.

In this paper, we focus on the second-price model. That is, our basic problem definitions presume second-price auctions; however, our bidding heuristics are not tailored to this case. As in true auctions, adopting the second-price model in pseudo-auctions simplifies the problem for the bidder. It also provides a reasonable approximation to the situation faced by TAC agents, as we now argue:

- In TAC entertainment auctions, agents submit bids (i.e., buy and sell offers) of the form specified above. If we interpret an agent's buyer and seller pricelines as the current order book (not including the agent's own bid), then the agent's immediate winnings are as determined by the winner determination rule, and payments are according to the second-price rule (i.e., the order-book prices prevail).

- In TAC hotel auctions, only buy bids are allowed. Assuming once again an order book that reflects all outstanding bids other than the agent's own, an accurate buyer priceline would indicate that the agent can win k units of a good if it pays—for *all* k units—a price just above the $(17-k)$th existing (other-agent) offer. The actual price it pays will be that of the 16th-highest unit offer (including its own offer). Since the agent's own bid may affect the price,[4] this situation lies between the first- and second-price characterizations of pseudo-auctions described above.

- In TAC flight auctions, agents may buy any number of units at the posted price. The situation at any given time is modeled exactly by the second-price pseudo-auction abstraction.

9.4.3 Bidding Problems and Heuristics

We are now ready to discuss the optimization module repeatedly employed by RoxyBot-06 within its bidding cycle to construct its bids. The key bidding decisions are: what goods to bid on, at what price, and when?

Technically, RoxyBot-06 faces an n-stage stochastic optimization problem. It solves this problem by collapsing those n stages into only two relevant stages, "current" and "future," necessitating only one stochastic pricing model (current prices are known). This approach is reasonable in TAC, and other similar combinations of one-shot and continuously-clearing simultaneous auction environments, as we now explain.

Since hotel auctions close in a random, unspecified order, RoxyBot-06, like most TAC agents, operates under the assumption that all hotel auctions close at the end of the current stage. Hence, the only pressing decisions regarding hotels are: what goods to bid on now and at what price? There is no need to consider the timing of bid placement. Accordingly, there is only one model of hotel prices.

In contrast, since flight and entertainment auctions clear continuously, a trading agent should reason about the relevant tradeoffs in timing its placement of bids on these goods. Still, under the assumption that hotel

[4] It can do so in two ways. First, the agent may submit the 16th-highest unit offer, in which case it sets the price. Second, when it bids for multiple units, the number it wins determines the price-setting unit, thus affecting the price for all winning units. Note that this second effect would be present even if the auction cleared at the 17th-highest price.

auctions close at the end of the current stage, it suffices to consider only one stochastic pricing model. Why? Because in future stages hotel prices, and hence winnings, are known, so the only remaining decisions are what flight and entertainment tickets to buy. But a reasonable agent will time its bids in these markets to capitalize on the "best" prices. (The best prices are the minima for buying and the maxima for selling.) Hence, it suffices for an agent's model of the future to predict only the best good prices, conditioned on current prices, of course. No further information is necessary.

Having established that it suffices for RoxyBot-06 to pose and solve a two-stage, rather than an n-stage, stochastic optimization problem, we now proceed to abstractly define a series of such problems designed to capture the essence of bidding under uncertainty in TAC-like hybrid markets that incorporate aspects of simultaneous and sequential, one-shot and continuously-clearing, auctions. Also in this section, we discuss heuristic solutions to this bidding problem: specifically, the expected value method (EVM), an approach that collapses stochastic information, and sample average approximation (SAA), an approach that exploits stochastic information and characterizes RoxyBot-06. The implementation details of SAA as it applies in the TAC domain are relegated to Appendix 9.10.

Problem Statements

The problem of bidding under uncertainty can be formulated as a two-stage stochastic program with integer recourse (see Birge and Louveaux [14] for an introduction to stochastic programming). In the first stage, when current prices are known, but future prices are uncertain, bids are selected. In the second stage, all uncertainty is resolved, and additional goods are traded, perhaps at undesirable prices. The objective in a stochastic program is to assign values to the first-stage variables (the bids) that maximize the sum of the first-stage objectives and the expected value of the ensuing objective in the second-stage. It is in the second stage that the bidder has *recourse*, and since it makes integer-valued decisions in that stage (the bidder decides what goods to buy and sell at known prices), the bidding problem is one with integer recourse.

In this section, we formulate a series of bidding problems as two-stage stochastic programs with integer recourse, each one tailored to a different type of auction mechanism, illustrating a different type of bidding decision. The mechanisms we study, inspired by TAC, are one-shot and continuously-clearing variants of second-price pseudo-auctions. In the former, bids can only be placed in the first stage; in the latter, decisions are made in both the first and second stages. Ultimately, we combine all decision problems into one grand bidding under uncertainty problem.

In our formal problem statements, we rely on the following notation:

- Variables:
  - Q^1 is a multiset of goods to buy now
  - Q^2 is a multiset of goods to buy later
  - R^1 is a multiset of goods to sell now
  - R^2 is a multiset of goods to sell later
- Constants:

- P^1 is a set of current buyer pricelines
- P^2 is a set of future buyer pricelines
- Π^1 is a set of current seller pricelines
- Π^2 is a set of future seller pricelines

Note that P^1 and Π^1 are always known, whereas P^2 and Π^2 are uncertain in the first stage but their uncertainty is resolved in the second stage.

Flight Bidding Problem An agent's task in bidding in flight auctions is to decide how many flights to buy now at current prices and later at the lowest future prices, given (known) current prices and a stochastic model of future prices. Although in TAC all units of each flight sell for the same price at any one time, we state the flight bidding problem more generally: we allow for different prices for different units of the same flight.

Continuously-Clearing, Buying Given a set of current buyer pricelines P^1 and a probability distribution f over future buyer pricelines P^2,

$$\mathsf{FLT}(f) = \max_{Q^1 \in \mathbb{Z}^n} \mathbb{E}_{P^2 \sim f} \left[\max_{Q^2 \in \mathbb{Z}^n} v(Q^1 \oplus Q^2) - \big(\text{Cost}(Q^1, P^1) + \text{Cost}(Q^1 \oplus Q^2, P^2) - \text{Cost}(Q^1, P^2)\big) \right] \tag{9.5}$$

Note that there are two cost terms referring to future pricelines (Cost$(\cdot, P^2)$). The first of these terms adds the total cost of the goods bought in the first and second stages. The second term subtracts the cost of the goods bought in just the first stage. This construction ensures that, if an agent buys k units of a good now, any later purchases of that good incur the charges of units $(k+1, k+2, ...)$ in the good's future priceline.

Entertainment Bidding Problem Abstractly, the entertainment *buying* problem is the same as the flight bidding problem. An agent must decide how many entertainment tickets to buy now at current prices and later at the lowest future prices. The entertainment *selling* problem is the opposite of this buying problem. An agent must decide how many tickets to sell now at current prices and later at the highest future prices.

Continuously-Clearing, Buying and Selling Given a set of current buyer and seller pricelines $(P, \Pi)^1$ and a probability distribution f over future buyer and seller pricelines $(P, \Pi)^2$,

$$\begin{aligned} \mathsf{ENT}(f) = \max_{Q^1, R^1 \in \mathbb{Z}^n} \mathbb{E}_{(P,\Pi)^2 \sim f} \Bigg[\max_{Q^2, R^2 \in \mathbb{Z}^n} & v((Q^1 \oplus Q^2) \ominus (R^1 \oplus R^2) \\ & - \big(\text{Cost}(Q^1, P^1) + \text{Cost}(Q^1 \oplus Q^2, P^2) - \text{Cost}(Q^1, P^2)\big) \\ & + \big(\text{Revenue}(R^1, \Pi^1) + \text{Revenue}(R^1 \oplus R^2, \Pi^2) - \text{Revenue}(R^1, \Pi^2)\big) \Bigg] \end{aligned} \tag{9.6}$$

subject to $Q^1 \supseteq R^1$ and $Q^1 \oplus Q^2 \supseteq R^1 \oplus R^2$, for all $(P, \Pi)^2$.

The constraints ensure that an agent does not sell more units of any good than it buys.

Hotel Bidding Problem Hotel auctions close at fixed times, but in an unknown order. Hence, during each iteration of an agent's bidding cycle, one-shot auctions approximate these auctions well. Unlike in the continuous setup, where decisions are made in both the first and second stages, in the one-shot setup, bids can only be placed in the first stage; in the second stage, winnings are determined and evaluated.

One-Shot, Buying Given a probability distribution f over future buyer pricelines P^2,

$$\mathsf{HOT}(f) = \max_{\beta^1 = \langle \vec{b}, 0 \rangle} \mathbb{E}_{P^2 \sim f} \left[v(\text{Buy}(\beta^1, P^2)) - \text{Cost}(\text{Buy}(\beta^1, P^2), P^2) \right] \tag{9.7}$$

Hotel Bidding Problem, with Selling Although it is not possible for agents to sell TAC hotel auctions, one could imagine an analogous auction setup in which it were possible to sell goods as well as buy them.

One-Shot, Buying and Selling Given a probability distribution f over future buyer and seller pricelines $(P, \Pi)^2$,

$$\max_{\beta^1 = \langle \vec{b}, \vec{a} \rangle} \mathbb{E}_{(P,\Pi)^2 \sim f} \left[v(\text{Buy}(\beta^1, P^2) \ominus \text{Sell}(\beta^1, \Pi^2)) - \text{Cost}(\text{Buy}(\beta^1, P^2), P^2) + \text{Revenue}(\text{Sell}(\beta^1, \Pi^2), \Pi^2) \right] \tag{9.8}$$

subject to $\text{Buy}(\beta^1, P^2) \geq \text{Sell}(\beta^1, \Pi^2)$, for all $(P, \Pi)^2$.

Bidding Problem Finally, we present (a slight generalization of) the TAC bidding problem by combining the four previous stochastic optimization problems into one. This abstract problem models bidding to buy and sell goods both via continuously-clearing and one-shot second-price pseudo-auctions, as follows:

Bidding Under Uncertainty Given a set of current buyer and seller pricelines $(P, \Pi)^1$ and a probability distribution f over future buyer and seller pricelines $(P, \Pi)^2$,

$$\begin{aligned} &\mathsf{BID}(f) = \\ &\max_{Q^1, R^1 \in \mathbb{Z}^n} \max_{\beta^1 = \langle \vec{b}, \vec{a} \rangle} \mathbb{E}_{(P,\Pi)^2 \sim f} \Bigg[\max_{Q^2, R^2 \in \mathbb{Z}^n} v((Q^1 \oplus Q^2) \ominus (R^1 \oplus R^2) \oplus \text{Buy}(\beta^1, P^2) \ominus \text{Sell}(\beta^1, P^2)) \\ &\qquad - \big(\text{Cost}(Q^1, P^1) + \text{Cost}(Q^1 \oplus Q^2, P^2) - \text{Cost}(Q^1, P^2) + \text{Cost}(\text{Buy}(\beta^1, P^2), P^2) \big) \\ &\quad + \big(\text{Revenue}(R^1, \Pi^1) + \text{Revenue}(R^1 \oplus R^2, \Pi^2) - \text{Revenue}(R^1, \Pi^2) + \text{Revenue}(\text{Sell}(\beta^1, \Pi^2), \Pi^2) \big) \Bigg] \end{aligned} \tag{9.9}$$

subject to $Q^1 \supseteq R^1$ and $Q^1 \oplus Q^2 \supseteq R^1 \oplus R^2$ and $\text{Buy}(\beta^1, P^2) \geq \text{Sell}(\beta^1, \Pi^2)$, for all $(P, \Pi)^2$.

Once again, this bidding problem is (i) stochastic: it takes as input a stochastic model of future prices; (ii) global: it seemlessly integrates flight, hotel, and entertainment bidding decisions; and (iii) dynamic: it facilitates simultaneous reasoning about current and future stages of the game.

Next, we describe various heuristic approaches to solving bidding under uncertainty.

Heuristic Solutions

The *expected value method* is a standard way of approximating the solution to a stochastic optimization problem. First, the given distribution is collapsed into a point estimate (e.g., the mean); then, a solution to the corresponding deterministic optimization problem is output as an approximate solution to the original stochastic optimization problem. This approach is inferior to the *sample average approximation* method, another means of approximating the solution to a stochastic optimization problem, which proceeds in two steps, as follows: (i) generate a set of sample scenarios, and (ii) solve an approximation of the problem that incorporates only the sample scenarios.

Using the theory of large deviations, Ahmed and Shapiro [4] establish the following: as $S \to \infty$, the probability that an optimal solution to the sample average approximation of a stochastic program with integer recourse is an optimal solution to the original stochastic optimization problem approaches 1 exponentially fast. Given hard time and space constraints, however, it is not always possible to sample sufficiently many scenarios to infer any reasonable guarantees about the quality of a solution to a sample average approximation. Hence, we propose a modified SAA heuristic, in which SAA is fed some tailor-made "important" scenarios, and we apply this idea to the bidding problem.

The bids that SAA places are prices that appear in one of its scenarios. There is no reason for SAA to bid higher on any good than its highest sampled price, because bidding the highest price is enough to win the good in all scenarios. (Similarly, there is also no reason for SAA to bid lower on any good than its lowest sampled price; instead, it suffices to bid zero.) Hence, SAA cannot win a good if the prices of that good in all of its scenarios are lower than the clearing price. How likely is this possibility?

Each draw from the distribution has an equal chance of being the highest-priced, assuming there are no ties. The probability that all of the sampled scenario prices are lower than the clearing price is $1/(S+1)$, where S is the number of scenarios. In particular, the probability that an SAA agent with 49 scenarios bidding in TAC Travel has a chance to win all 8 hotels (i.e., the probability that the price in at least one of its scenarios is higher than the clearing price) is only $\left(1 - \frac{1}{49+1}\right)^8 = 0.98^8 \approx 0.85$.

To remedy this situation, we designed and implemented a variant of SAA. The SAA* heuristic is a close cousin of SAA, the only difference arising in their respective scenario sets. Whereas SAA samples S scenarios, SAA* samples only $S - |N|$ scenarios, where $|N| = \sum_g N_g$. SAA* creates an additional $|N|$ scenarios as follows: for each unit k of each good $g \in G$, it sets the price of the kth unit of good g to the upper limit of its range of possible prices and, after conditioning on this price setting, it sets the prices of the other goods to their mean values. The SAA* bidding heuristic characterizes RoxyBot-06.

What is unique about our agent's approach is that since 2002 [45], RoxyBot has determined its bids by optimizing with respect to a *set* of scenarios. To our knowledge no other agents optimize with respect to multiple scenarios simultaneously. ATTAC-01 [98], the top-scoring agent in 2003, considered multiple scenarios, but computed bids independently for each and then averaged those bids.

9.4.4 Summary

In this section, we developed a series of bidding problems, and heuristics solutions to those problems, that capture the essence of bidding in the one-shot and continuously-clearing auctions that characterize TAC. To some extent at least, our approach to bidding has been validated by the success of RoxyBot-06 in TAC-06. Our presentation was deliberately abstract, though, so as to suggest that our problems and their solutions are applicable well beyond the realm of TAC: e.g., to bidding for interdependent goods in separate eBay auctions. It remains to validate our approach in other application domains.

9.5 Price Prediction

Next, we describe how RoxyBot-06 builds its stochastic models of flight, hotel, and event prices. Each model is a discrete probability distribution, represented by a set of "scenarios." Each scenario is a vector of "future" prices—prices at which goods can be bought and sold after the current stage. For flights, the price prediction model is not stochastic: the future buy price is simple RoxyBot-06's prediction of the expected minimum price during the current stage. For hotels, the future buy prices are predicted by Monte Carlo simulations of simultaneous ascending auctions to approximate competitive equilibrium prices. There are no current buy prices for hotels. For entertainment, RoxyBot-06 predicts future buy and sell prices based on historical data. Details of these price prediction methods are the focus of this section.

9.5.1 Flights

Efforts to deliberate about flight purchasing start with understanding the TAC model of flight price evolution.

TAC Flight Prices' Stochastic Process

Flight prices follow a biased random walk. They are initialized uniformly in the range $[250, 400]$, and constrained to remain in the range $[150, 800]$. At the start of each TAC game instance, a bound z on the final perturbation value is selected for each flight. These bounds are not revealed to the agents. What is revealed to the agents is a sequence of random flight prices. Every ten seconds, TACAir perturbs the price of each flight by a random value that depends on the hidden parameter z and the current time t as follows: given constants $c, d \in \mathbb{R}$ and $T > 0$, each (intermediate) bound on the perturbation value is a linear function of t:

$$x(t, z) = c + \frac{t}{T}(z - c) \tag{9.10}$$

The perturbation value at time t is drawn uniformly from one of the following ranges (see Algorithm 1):

- $U[-c, x(t, z)]$, if $x(t, z) > 0$
- $U[-c, +c]$, if $x(t, z) = 0$
- $U[x(t, z), +c]$, if $x(t, z) < 0$

Observe that the expected perturbation value in each case is simply the average of the corresponding upper and lower bounds. In particular,

- if $x(t, z) > c$, then the expected perturbation is positive;
- if $x(t, z) \in (0, c)$, then the expected perturbation is negative;
- if $x(t, z) \in (-c, 0)$, then the expected perturbation is positive;
- otherwise, if $x(t, z) \in \{-c, 0, c\}$, then the expected perturbation is zero.

Moreover, using Equation 9.10, we can compute the expected perturbation value conditioned on z:

- if $z \in [0, c]$, then $x(t, z) \in [0, c]$, so prices are expected not to increase;
- if $z \in [c, c + d]$, then $x(t, z) \in [c, c + d]$, so prices are expected not to decrease;
- if $z \in [-c, 0]$, then $x(t, z) \in [-c, c]$, so prices are expected not to increase while $t \leq \frac{cT}{c-z}$ and they are expected not to decrease while $t \geq \frac{cT}{c-z}$.

Based on the above discussion, we note the following: for TAC's parameter settings, namely $c = 10$, $d = 30$, and $T = 540$, with z uniformly distributed in the range $[-c, d]$, given no further information about z, flight prices are expected to increase (i.e., the expected perturbation is positive). Conditioned on z, however, flight prices may increase *or* decrease (i.e., the expected perturbation can be positive or negative). To facilitate their flight deliberations, one of the tasks faced by TAC agents is to model the probability distribution $P_t[z]$ associated with z at time t for use in predicting current and future flight prices.

An Application of Bayesian Updating

A model of a probability distribution can be built using Bayesian updating. Although the value of the hidden parameter z is never revealed to the agents, the agents do observe flight prices that depend on this value throughout the game. Before making any observations, assuming z is uniformly distributed (as it is in TAC), it is equally likely to be anywhere in $[a, b]$: i.e., $P[z] = \frac{1}{b-a}$. Given a sequence of observations $y_1, \ldots, y_t$, the probability distribution $P[z \mid y_1, \ldots, y_t]$ can be updated using Bayes' rule. Specifically,

$$P[z \mid y_1, \ldots, y_t] = \frac{P[y_1, \ldots, y_t \mid z]P[z]}{\int_{z'} P[y_1, \ldots, y_t \mid z']P[z']\, dz'} \tag{9.11}$$

where

$$P[y_1, \ldots, y_t \mid z] = \prod_{i=1}^{t} P[y_i \mid y_1, \ldots, y_{i-1}, z] \tag{9.12}$$

$$= \prod_{i=1}^{t} P[y_i \mid z] \tag{9.13}$$

Equation 9.13 follows from the fact that future observations are independent of past observations; observations depend only on the hidden parameter z.

Algorithm 1 getRange(t, z)

compute $x(t, z)$ {Equation 9.10}
if $x(t, z) > 0$ **then**
  $a = -c$; $b = \lceil x(t, z) \rceil$
else if $x(t, z) < 0$ **then**
  $a = \lfloor x(t, z) \rfloor$; $b = +c$
else
  $a = -c$; $b = +c$
end if
return $[a, b]$ {range}

Algorithm 2 Flight_Prediction(t, y_{t+1}, P_t)

for all $z \in R$ **do**
  $[a, b] =$ getRange(t, z)
  if $y_{t+1} \in [a, b]$ **then**
    $Q_{t+1}[z] = \left(\frac{1}{b-a}\right) P_t[z]$
  else
    $Q_{t+1}[z] = 0$
  end if
end for{update probabilities}
for all $z \in R$ **do**
  $P_{t+1}[z] = \frac{Q_{t+1}[z]}{\int_{z'} Q_{t+1}[z']\, dz'}$
end for{normalize probabilities}
return P_{t+1} {probabilities}

An implementation of RoxyBot-06's Bayesian updating procedure is described in Algorithm 2. Letting $P_0[z] = P[z]$ and $Q_{t+1}[z] = P[y_{t+1} \mid z]P_t[z]$,

$$P_{t+1}[z] = \frac{Q_{t+1}[z]}{\int_{z'} Q_{t+1}[z']\, dz'} \tag{9.14}$$

Note that $P_{t+1}[z] = P[z \mid y_1, \ldots, y_{t+1}]$.

RoxyBot-06's Flight Prices Prediction Method

Given a probability distribution $P_t[z]$, to predict a flight price, RoxyBot could simulate a random walk from time $t + 1, \ldots, t'$ and select the minimum price (see Algorithm 3). In practice, however, only RoxyBot-06's hotel and event price predictions are stochastic; its flight price predictions are point estimates (i.e., constant across scenarios). For each flight and for each possible value of the hidden parameter z, RoxyBot-06 simulates an "expected" random walk (see Algorithm 4), selects the minimum price, and then outputs as its prediction the expectation of these minima, averaging according to $P_t[z]$. Alternative scenario generation procedures are also possible. In Algorithm 3, an agent could calculate expected perturbations instead of sampling; or, in Algorithm 4, an agent could sample instead of calculating expected perturbations. Our choice of flight price prediction method was guided by time constraints.

Algorithm 3 Sample_Minimum_Price(t, t', p_t, P_{t+1})

```
min[z] = +∞
sample z ~ P_{t+1}[z]
for τ = t+1, ..., t' do
  [a, b] = getRange(τ, z)
  sample Δ ~ U[a, b] {perturbation}
  p_τ = p_{τ-1} + Δ {perturb price}
  p_τ = max(150, min(800, p_τ))
  if p_τ < min[z] then
    min[z] = p_τ
  end if
end for{random walk}
return min
```

Algorithm 4 Expected_Minimum_Price(t, t', p_t, P_{t+1})

```
for all z ∈ R do
  min[z] = +∞
  for τ = t+1, ..., t' do
    [a, b] = getRange(τ, z)
    Δ = (b-a)/2 {expected perturbation}
    p_τ = p_{τ-1} + Δ {perturb price}
    p_τ = max(150, min(800, p_τ))
    if p_τ < min[z] then
      min[z] = p_τ
    end if
  end for
end for
return ∫_z P_{t+1}[z] min[z] dz
```

9.5.2 Hotels

RoxyBot-06's approach to hotel price prediction is inspired by Walverine's [25], in which the tâtonnement method [107] is used to approximate competitive equilibrium (CE) prices. In a competitive market where each individual's effect on prices is negligible, equilibrium prices are prices at which supply equals demand, assuming all producers are profit-maximizing and all consumers are utility-maximizing.

Formally, let $\vec{p}$ denote a vector of prices. If $\vec{y}(\vec{p})$ denotes the cumulative supply of all producers, and if $\vec{x}(\vec{p})$ denotes the cumulative demand of all consumers, then $\vec{z}(\vec{p}) = \vec{x}(\vec{p}) - \vec{y}(\vec{p})$ denotes the excess demand in the market. The tâtonnement process adjusts the price vector at iteration $n+1$, given the price vector at iteration n and a sequence $\{\alpha_n\}$ of adjustment rates: $\vec{p}_{n+1} = \vec{p}_n + \alpha_n \vec{z}(\vec{p}_n)$.

In the TAC game context, tâtonnement is not guaranteed to converge. Walverine attempts to force convergence by letting $\alpha_n \to 0$. We fix $\alpha_n = \frac{1}{24}$ and instead force convergence by modifying the adjustment process to simulate *simultaneous ascending auctions* (SimAA) [27]. In SimAAs prices increase as long as there is excess demand but they can never decrease: $\vec{p}_{n+1} = \vec{p}_n + \alpha_n \max\{\vec{z}(\vec{p}_n), 0\}$.

In TAC, cumulative supply is fixed. Hence, the key to computing excess demand is to compute cumulative demand. Each TAC agent knows the preferences of its own clients, but must estimate the demand of the

others. Walverine computes a single hotel price prediction (a point estimate) by considering its own clients' demands together with those of 56 "expected" clients. Briefly, the utility of an expected client is an average across travel dates and hotel types augmented with fixed entertainment bonuses that favor longer trips (see Wellman *et al.* [25] for details). In contrast, RoxyBot-06 builds a stochastic model of hotel prices consisting of S scenarios by considering its own clients' demands together with S random samples of 56 clients. A (random or expected) client's demand is simply the quantity of each good in its optimal package, given current prices. The cumulative demand is the sum total of all client's individual demands.

In Figure 9.1, we present two scatter plots that depict the quality of various hotel price predictions at the beginning of the TAC 2002 final games. All price predictions are evaluated using two metrics: Euclidean distance and the "expected value of perfect prediction" (EVPP). Euclidean distance is a measure of the difference between two vectors, in this case the actual and the predicted prices. The value of perfect prediction (VPP) for a client is the difference between its surplus (value of its preferred package less price) based on actual and predicted prices. EVPP is the VPP averaged over the distribution of client preferences.[5]

On the left, we plot the predictions generated using the CE methods: tâtonnement and SimAA, both with $\alpha = \frac{1}{24}$; expected, random, and exact. The "exact" predictions are computed with knowledge of the actual clients' in the games, not just the client distribution; hence, they serve as a lower bound on the performance of these techniques on this data set. Under both metrics, and for both expected and random, SimAA outperforms tâtonnement. The right plot juxtaposes RoxyBot-06's predictions (SimAA random) and the TAC 2002 agents' predictions.[6] Note that SimAA expected at (198,37) performs as well as Walverine at (197,39).

We interpret each prediction generated using randomly sampled clients as a sample scenario, so that a set of such scenarios represents draws from a probability distribution over CE prices. The corresponding vector of predicted prices that is evaluated is actually the average of multiple (40) such predictions; that is, we evaluate an estimate of the mean of this probability distribution. The predictions generated using sets of random clients are not as good as the predictions with expected clients (see Figure 9.1 left), although with more than 40 sets of random clients, the results might improve. Still, the predictions with random clients comprise RoxyBot-06's stochastic model of hotel prices, which is key to its bidding strategy. Moreover, using random clients helps RoxyBot-06 make better interim predictions later in the game as we explain next.

The graphs depicted in Figure 9.1 pertain to hotel price predictions made at the beginning of the game, when all hotel auctions are open. In those CE computations, prices are initialized to 0. As hotel auctions close, RoxyBot-06 updates the predicted prices of the hotel auctions that remain open. We experimented with two ways of constructing interim price predictions. The first is to intialize and lower bound the prices in the hotel markets at their closing (for closed auctions) or current ask (for open auctions) prices while computing competitive equilibrium prices.[7] The second differs in its treatment of closed auctions: we simulate a process of distributing the goods in the closed auctions to the clients who want them most, and then exclude the closed markets (i.e., fix prices at ∞) from further computations of competitive equilibrium prices.

[5] See Wellman *et al.* [108] for details.

[6] With the exception of the RoxyBot-06 data point, this plot was produced by the Walverine team [108].

[7] At first blush, it may seem more sensible to *fix* these prices at their closing prices. But if some hotel closing price were artificially low, and if that price could not increase, then the prices of the hotels complementing the hotel in question would be artificially high.

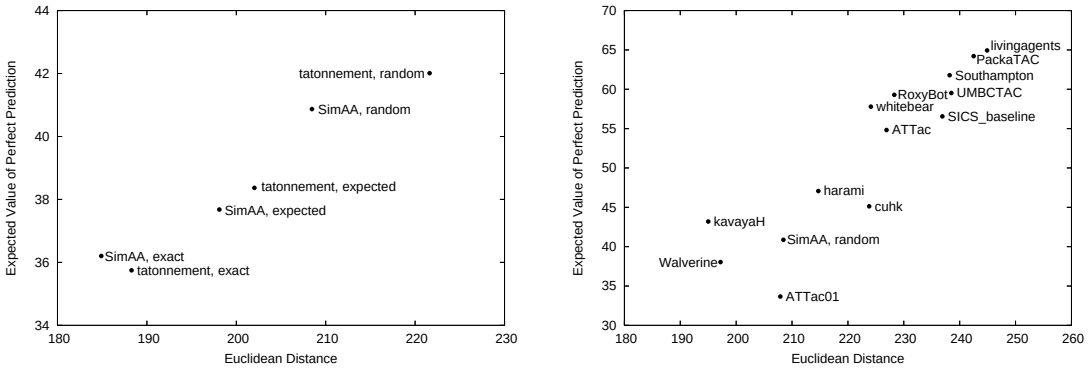


Figure 9.1: EVPP and Euclidean Distance for the CE price prediction methods (tâtonnement and SimAA with $\alpha = \frac{1}{24}$; expected, random, and exact) and the TAC 2002 agents' predictions in the 2002 finals (60 games). The plot on the left shows that SimAA's predictions are better than tâtonnement's and that expected's are better than random's. RoxyBot-06's method of hotel price prediction (SimAA, Random) is plotted again on the right. Note the differences in scales between the two plots.

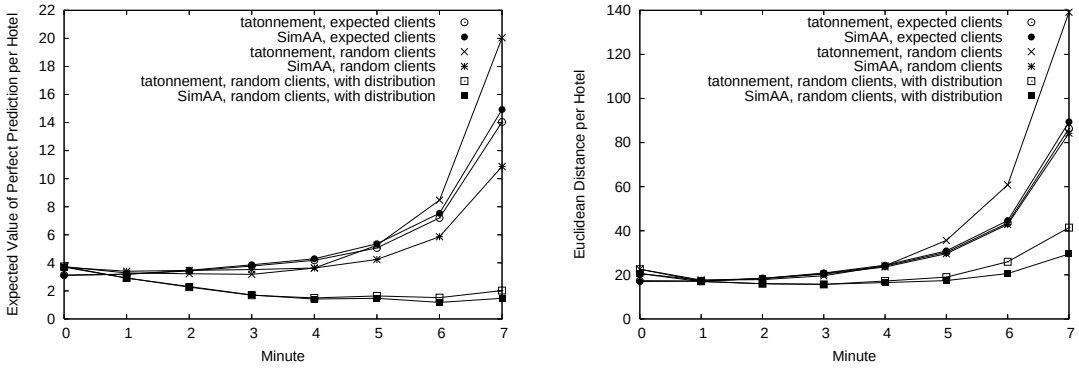


Figure 9.2: EVPP and Euclidean Distance in TAC 2006 finals (165 games) of the CE price prediction methods with and without distribution as the game progresses. Distribution improves prediction quality.

Regarding the second method—the distribution method—we determine how to distribute goods by computing competitive equilibrium prices! As explained in Algorithm 5, all the hotels (in both open and closed auctions) are distributed to *random* clients by determining who is willing to pay the competitive equilibrium prices for what. It is not clear how to distribute goods to expected clients.

Figure 9.2, which depicts prediction quality over time, shows that the prediction methods enhanced with distribution are better than the predictions obtained by merely initializing the prices of closed hotel auctions at their closing prices. Hotels that close early tend to sell for less than hotels that close late; hence, the prediction quality of any method that makes decent initial predictions is bound to deteriorate if those predictions remain relatively constant throughout the game.

Returning to Figure 9.1, SimAA outperforms tâtonnement as a means of hotel price prediction with both expected and random clients on the TAC 2002 finals data set. It remains to show that this performance gain

Algorithm 5 Distribute

```
1: for all hotel auctions h do
2:     initialize price to 0
3:     initialize supply to 16
4: end for
5: compute competitive equilibrium prices {Tâtonnement or SimAA}
6: for all closed hotel auctions h do
7:     distribute units of h to those who demand them at the computed competitive equilibrium prices
8:     distribute any leftover units of h uniformly at random
9: end for
```

does not come at the expense of increased computational efforts. Indeed, it does not; see Table 9.2.

Table 9.2 shows the runtimes of the CE prediction methods on TAC 2002 (60 games) and TAC 2006 (165 games) finals data set at minute 0. Here, we see that SimAA is almost five times faster than tâtonnement with expected clients, and almost ten times faster with random clients. Tâtonnement is not guaranteed to converge, and in general, it does not. Instead, the tâtonnement procedure usually runs for the maximum number of iterations (fixed at 10,000, in our implementation).

Moving to minutes 1–7 on the TAC 2006 finals data set, SimAA is more than five times faster with expected clients and more than ten times faster with random clients. Note that CE prices are bounded below by current ask prices; over time, those prices tend to increase, leading to faster and faster runtimes. Extending the sampling methods to incorporate distribution, runtimes double. This factor of two slowdown is acceptable for SimAA but not for tâtonnement, since the former runs ten times as fast as the latter.

	Exp Tât	Exp SimAA	Sam Tât	Sam SimAA	Dist Tât	Dist SimAA
2002, minute 0	2213	507	1345	157	—	—
2006, minute 0	2252	508	1105	130	1111	128
2006, average 1–7	2248	347	1138	97	2249	212

Table 9.2: Runtimes for the CE price prediction methods, in milliseconds. Experiments were run on AMD Athlon(tm) 64 bit 3800+ dual core processors with 2M of RAM. The machines were not dedicated.

The simulation methods discussed in this section—the tâtonnement process and simultaneous ascending auctions—were employed to predict hotel prices only. (In our simulations, flight prices are fixed at their expected minima, and entertainment prices are fixed at 80.) In principle, competitive equilibrium (CE) prices could serve as predictions in all TAC markets. However, CE prices are unlikely to be good predictors of flight prices, since flight prices are determined exogenously. With regard to entertainment tickets, CE prices might have predictive power; however, incorporating entertainment tickets into the tâtonnement and SimAA calculations would have been expensive. (In our simulations, following Wellman *et al.* [108], client utilities are simply augmented with fixed entertainment bonuses that favor longer trips.) Nonetheless, in future work, it could be of interest to evaluate the success of these or related methods in predicting CDA clearing prices.

Finally, we note that we refer to our methods of computing excess demand as "client-based" because we compute the demands of each client on an individual basis. In contrast, one could employ an "agent-based" method, whereby the demands of agents, not clients, would be calculated. Determining an agent's demands

involves solving so-called *completion*, a deterministic (prices are known) optimization problem at the heart of RoxyBot-00's architecture [46]. As TAC completion is NP-hard, the agent-based method of predicting hotel prices is too expensive to be included in RoxyBot-06's inner loop. In designing RoxyBot-06, we reasoned that an architecture based on a stochastic pricing model generated using the client-based method and randomly sampled clients would outperform one based on a point estimate pricing model generated using the agent-based method and some form of expected clients, but we did not verify this conjecture empirically.

9.5.3 Entertainment

During each bid interval, RoxyBot-06 predicts current and future buy and sell prices for tickets to all entertainment events. These price predictions are optimistic: the agent assumes it can buy (or sell) goods at the least (or most) expensive prices that it expects to see before the end of the game. More specifically, each current price prediction is the best predicted price during the current bid interval.

RoxyBot-06's estimates of entertainment ticket prices are based on historical data from the past 40 games. To generate a scenario, a sample game is drawn at random from this collection, and the sequences of entertainment bid, ask, and transaction prices are extracted. Given such a history, for each auction a, let $trade_{ai}$ denote the price at which the last trade before time i transacted; this value is initialized to 200 for buying and 0 for selling. In addition, let bid_{ai} denote the bid price at time i, and let ask_{ai} denote the ask price at time i.

RoxyBot-06 predicts the future buy price in auction a after time t as follows:

$$future\_buy_{at} = \min_{i=t+1,\ldots,T} \min\{trade_{ai}, ask_{ai}\} \tag{9.15}$$

In words, the future buy price at each time $i = t+1, \ldots, T$ is the minimum of the ask price after time i and the most recent trade price. The future buy price at time t is the minimum across the future buy prices at all later times. The future sell price after time t is predicted analogously:

$$future\_sell_{at} = \max_{i=t+1,\ldots,T} \max\{trade_{ai}, bid_{ai}\} \tag{9.16}$$

9.6 TAC 2006 Competition Results

To further establish the efficacy of RoxyBot's strategy, we include the results of the TAC 2006 tournament.

9.6.1 Summary

Table 9.3 lists the agents entered in TAC-06 and Table 9.4 summarizes the outcome. RoxyBot dominated the seeding round, which consisted of 960 games. The finals comprised 165 games over three days, with the 80 games on the last day weighted 1.5 times as much as the 85 over the first two days. In spite of its glowing performance in the preliminary (qualifying and seeding) rounds, on the first day of the finals, RoxyBot finished third, behind Mertacor and Walverine—the top scorers in 2005. As it happens, RoxyBot's optimization routine, which was designed for stochastic hotel and entertainment price predictions, was accidentally fed deterministic predictions (i.e., point price estimates) for entertainment. Moreover, these predictions were fixed, rather than adapted based on recent game history.

On days 2 and 3, RoxyBot ran properly, basing its bidding in all auctions on stochastic information. Moreover, the agent was upgraded after day 1 to bid on flights not just once, but twice, during each minute. This enabled the agent to delay its bidding somewhat at the end of a game for flights whose prices are decreasing. No doubt this minor modification enabled RoxyBot to emerge victorious in 2006, edging out Walverine by a whisker, below the integer precision reported in Table 9.4. The actual margin was 0.22—a mere 22 parts in 400,000. Adjusting for control variates [94] spreads the top two finishers a bit further.[8] Accounting for RoxyBot's difficulties on day 1 of the finals, the difference between the bidding capabilities of the first and second place agents is perhaps not as close as it seems.

Agent	Affiliation	Reference
006	Swedish Inst Comp Sci	[7]
kin_agent	U Macau	
L-Agent	Carnegie Mellon U	[96]
Mertacor	Aristotle U Thessaloniki	[101, 58]
RoxyBot	Brown U	[43, 45, 46, 68]
UTTA	U Tehran	
Walverine	U Michigan	[25, 111]
WhiteDolphin	U Southampton	[52, 103]

Table 9.3: TAC-06 participants.

Agent	Seeding	Finals	Adjustment Factor
RoxyBot	4148	4032	–5
Walverine	3992	4032	–17
WhiteDolphin	3901	3936	–2
006	3882	3902	–27
Mertacor	3509	3880	–16
L-Agent	3284	3860	7
kin_agent	3897	3725	0
UTTA	1726	2680	–14

Table 9.4: TAC-06 scores, seeding and final rounds, with adjustment factors based on control variates.

9.6.2 Details

Finally, we detail the results of the last day of the TAC-06 finals (80 games). We omit the first two days because agents can vary across days, but cannot vary within. Presumably, the entries on the last day are the teams' preferred versions of the agents. Mean scores, utilities, and costs are plotted in Figure 9.3 and detailed statistics are tabulated in Table 9.5.

There is no single metric such as low hotel or flight costs that is responsible for RoxyBot's success. Rather its success derives from the right balance of contradictory goals. In particular, RoxyBot incurs high hotel and

[8] Kevin Lochner computed these adjustment factors using the method described in Wellman *et al.* [110], Chapter 8.

mid-range flight costs while achieving mid-range trip penalty and high event profit. (An agent suffers trip penalties to the extent that it assings its clients packages that differ from their preferred.)

We compare RoxyBot with two closest rivals: Walverine and WhiteDolphin.

Comparing to Walverine first, Walverine bids lower prices (by 55) on fewer hotels (49 less), yet wins more (0.8) and wastes less (0.42). It would appear that Walverine's hotel bidding strategy outperforms RoxyBot's, except that RoxyBot earns a higher hotel bonus (15 more). RoxyBot also gains an advantage by spending 40 less on flights and earning 24 more in total entertainment profit.

A very different competition takes place between RoxyBot and WhiteDolphin. WhiteDolphin bids lower prices (120 less) on more hotels (by 52) than RoxyBot. RoxyBot spends much more (220) on hotels than WhiteDolphin but makes up for it by earning a higher hotel bonus (by 96) and a lower trip penalty (by 153). It seems that WhiteDolphin's strategy is to minimize costs even if that means sacrificing utility.

	Rox	Wal	Whi	SIC	Mer	L-A	kin	UTT
# of Hotel Bids	130	81	182	33	94	58	15	24
Average of Hotel Bids	170	115	50	513	147	88	356	498
# of Hotels Won	15.99	16.79	23.21	13.68	18.44	14.89	15.05	9.39
Hotel Costs	1102	1065	882	1031	902	987	1185	786
# of Unused Hotels	2.24	1.82	9.48	0.49	4.86	1.89	0.00	0.48
Hotel Bonus	613	598	517	617	590	592	601	424
Trip Penalty	296	281	449	340	380	388	145	213
Flight Costs	4615	4655	4592	4729	4834	4525	4867	3199
Event Profits	110	26	6	-6	123	-93	-162	-4
Event Bonus	1470	1530	1529	1498	1369	1399	1619	996
Total Event Profits	1580	1556	1535	1492	1492	1306	1457	992
Average Utility	9787	9847	9597	9775	9579	9604	10075	6607
Average Cost	5608	5693	5468	5765	5628	5605	6213	3989
Average Score	4179	4154	4130	4010	3951	3999	3862	2618

Table 9.5: 2006 Finals, Last day. Tabulated Statistics.

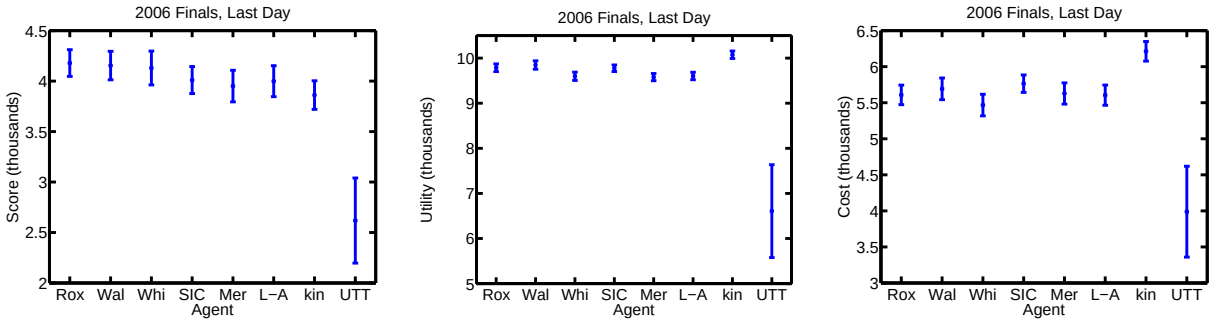


Figure 9.3: 2006 Finals, Last day. Mean scores, utilities, and costs, and 95% confidence interavals.

9.7 Collective Behavior

The hotel price prediction techniques described in Section 9.5.2 are designed to compute (or at least approximate) competitive equilibrium prices without full knowledge of the client population. In this section, we assume this knowledge and view the output of the tâtonnement and SimAA calculations not as predictions but as ground truth. We compare the actual prices in the final games to this ground truth in respective years since 2002 to determine whether TAC market prices resemble CE prices. What we find is depicted in Figure 9.4. Because of the nature of our methods, these calculations pertain to hotel prices only.

The results are highly correlated on both metrics (Euclidean distance and EVPP). We observe that the accuracy of CE price calculations has varied from year to year. 2003 was the year in which TAC Supply Chain Management (SCM) was introduced. Many participants diverted their attention away from Travel towards SCM that year, perhaps leading to degraded performance in Travel. Things seem to improve in 2004 and 2005. We cannot explain the setback in 2006, except by noting that performance is highly dependent on the particular agent pool, and in 2006 there were fewer agents in that pool.

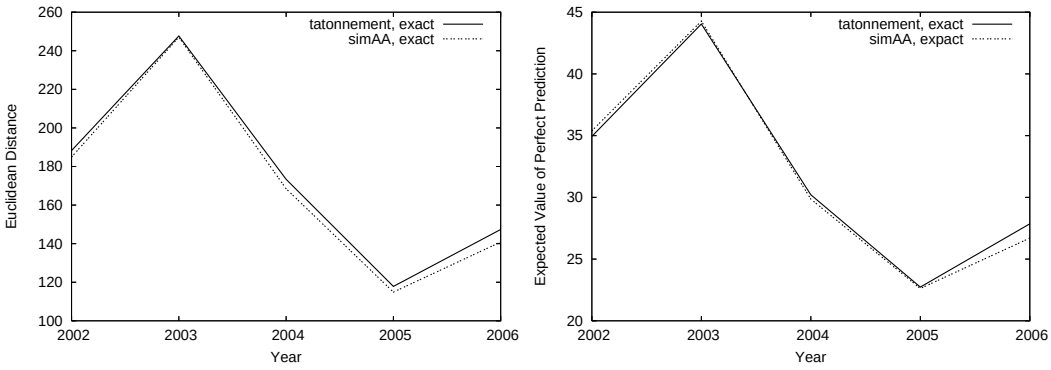


Figure 9.4: A comparison of the actual (hotel) prices to the output of competitive equilibrium price calculations in the final games since 2002. The label "exact" means: full knowledge of the client population.

9.8 Conclusion

The foremost aim of trading agent research is to develop a body of techniques for effective design and analysis of trading agents. Contributions to trading agent design include the invention of trading strategies, together with models and algorithms for realizing their computation and methods to measure and evaluate the performance of agents characterized by those strategies. Researchers seek both specific solutions to particular trading problems and general principles to guide the development of trading agents across market scenarios. This chapter purports to contribute to this research agenda. We described the design and implementation of RoxyBot-06, an able trading agent as demonstrated by its performance in TAC-06.

Although automated trading in electronic markets has not yet fully taken hold, the trend is well underway. Through TAC, the trading agent community is demonstrating the potential for autonomous bidders to make

pivotal trading decisions in a most effective way. Such agents offer the potential to accelerate the automation of trading more broadly, and thus shape the future of commerce.

9.9 Acknowledgments

This chapter extends Lee *et al.* [68]. Excerpts of Sections 9.4.1 and 9.4.2 were extracted from Wellman *et al.* [110]. This research was supported by NSF Career Grant #IIS-0133689.

9.10 TAC Bidding Problem: SAA

The problem of bidding in the simultaneous auctions that characterize TAC can be formulated as a two-stage stochastic program. In this appendix, we present the implementation details of the integer linear program (ILP) encoded in RoxyBot-06 that approximates an optimal solution to this stochastic program.[9]

We formulate this ILP assuming current prices are known, and future prices are uncertain in the first stage but revealed in the second stage. Note that whenever prices are known, it suffices for an agent to make decisions about the quantity of each good to buy, rather than about bid amounts, since choosing to bid an amount that is greater than or equal to the price of a good is equivalent to a decision to buy that good.

Unlike in the main body of the chapter, this ILP formulation of bidding in TAC assumes linear prices. Table 9.6 lists the price constants and decision variables for each auction type. For hotels, the only decisions pertain to buy offers; for flights, the agent decides how many tickets to buy now and how many to buy later; for entertainment events, the agent chooses sell quantities as well as buy quantities.

Hotels	Price	Variable (bid)
bid now	$\mathcal{Y}_{as}$	ϕ_{apq}

Flights and Events	Price	Variable (qty)
buy now	$\mathcal{M}_a$	μ_a
buy later	$\mathcal{Y}_{as}$	υ_{as}

Events	Price	Variable (qty)
sell now	$\mathcal{N}_a$	ν_a
sell later	$\mathcal{Z}_{as}$	ζ_{as}

Table 9.6: Auction types and associated price constants and decision variables.

9.10.1 Index Sets

$a \in A$ indexes the set of goods, or auctions.

$a_f \in A_f$ indexes the set of flight auctions.

[9] The precise formulation of RoxyBot-06's bidding ILP appears in Lee *et al.* [68]. The formulation here is slightly simplified, but we expect it would perform comparably in TAC. The key differences are in flight and entertainment bidding.

$a_h \in A_h$ indexes the set of hotel auctions.

$a_e \in A_e$ indexes the set of event auctions.

$c \in C$ indexes the set of clients.

$p \in P$ indexes the set of prices.

$q \in Q$ indexes the set of quantities
(i.e., the units of each good in each auction).

$s \in S$ indexes the set of scenarios.

$t \in T$ indexes the set of trips.

9.10.2 Constants

$\mathcal{G}_{at}$ indicates the quantity of good a required to complete trip t.

$\mathcal{M}_a$ indicates the current buy price of a_f, a_e.

$\mathcal{N}_a$ indicates the current sell price of a_e.

$\mathcal{Y}_{as}$ indicates the future buy price of a_f, a_h, a_e in scenario s.

$\mathcal{Z}_{as}$ indicates the future sell price of a_e in scenario s.

$\mathcal{H}_a$ indicates the hypothetical quantity won of hotel a_h.

$\mathcal{O}_a$ indicates the quantity of good a the agent owns.

$\mathcal{U}_{ct}$ indicates client c's value for trip t.

9.10.3 Decision Variables

$\Gamma = \{\gamma_{cst}\}$ is a set of boolean variables indicating whether or not client c is allocated trip t in scenario s.

$\Phi = \{\phi_{apq}\}$ is a set of boolean variables indicating whether to bid price p on the qth unit of a_h.

$M = \{\mu_a\}$ is a set of integer variables indicating how many units of a_f, a_e to buy now.

$N = \{\nu_a\}$ is a set of integer variables indicating how many units of a_e to sell now.

$Y = \{\upsilon_{as}\}$ is a set of integer variables indicating how many units of a_f, a_e to buy later in scenario s.

$Z = \{\zeta_{as}\}$ is a set of integer variables indicating how many units of a_e to sell later in scenario s.

9.10.4 Objective Function

$$\max_{\Gamma,\Phi,M,N,Y,Z} \sum_{S} \left(\overbrace{\sum_{C,T} \mathcal{U}_{ct}\gamma_{cts}}^{trip\ value} - \overbrace{\sum_{A_f} \left(\overbrace{\mathcal{M}_a\mu_a}^{current} + \overbrace{\mathcal{Y}_{as}\upsilon_{as}}^{future} \right)}^{flight\ cost} - \overbrace{\sum_{A_h,Q,p\geq\mathcal{Y}_{as}} \mathcal{Y}_{as}\phi_{apq}}^{hotel\ cost} + \right. \tag{9.17}$$

$$\left. \sum_{A_e} \left(\overbrace{\overbrace{\mathcal{N}_a\nu_a}^{current} + \overbrace{\mathcal{Z}_{as}\zeta_{as}}^{future}}^{event\ revenue} - \overbrace{\overbrace{\mathcal{M}_a\mu_a}^{current} - \overbrace{\mathcal{Y}_{as}\upsilon_{as}}^{future}}^{event\ cost} \right) \right)$$

9.10.5 Constraints

$$\sum_{T} \gamma_{cst} \leq 1 \quad \forall c \in C, s \in S \tag{9.18}$$

$$\overbrace{\sum_{C,T} \gamma_{cst}\mathcal{G}_{at}}^{allocation} \leq \overbrace{\mathcal{O}_a}^{own} + \overbrace{(\mu_a + \upsilon_{as})}^{buy} \quad \forall a \in A_f, s \in S \tag{9.19}$$

$$\overbrace{\sum_{C,T} \gamma_{cst}\mathcal{G}_{at}}^{allocation} \leq \overbrace{\mathcal{O}_a}^{own} + \overbrace{\sum_{Q,p\geq\mathcal{Y}_{as}} \phi_{apq}}^{buy} \quad \forall a \in A_h, s \in S \tag{9.20}$$

$$\overbrace{\sum_{C,T} \gamma_{cst}\mathcal{G}_{at}}^{allocation} \leq \overbrace{\mathcal{O}_a}^{own} + \left(\overbrace{\mu_a + \upsilon_{as}}^{buy} \right) - \left(\overbrace{\nu_a + \zeta_{as}}^{sell} \right) \quad \forall a \in A_e, s \in S \tag{9.21}$$

$$\sum_{P,Q} \phi_{apq} \geq \mathcal{H}_a \quad \forall a \in A_h \tag{9.22}$$

$$\sum_{P} \phi_{apq} \leq 1 \quad \forall a \in A_h, q \in Q \tag{9.23}$$

Equation (9.18) limits each client to one trip in each scenario. Equation (9.19) prevents the agent from allocating flights that it does not own or buy. Equation (9.20) prevents the agent from allocating hotels that it does not own or buy. Equation (9.21) prevents the agent from allocating event tickets that it does not own or buy and not sell. Equation (9.22) ensures the agent bids on at least HQW units in each hotel auction. Equation (9.23) prevents the agent from placing more than one buy offer per unit in each hotel auction.

An agent might also be constrained not to place sell offers on more units of each good than it owns, and/or not to place buy (sell) offers for more units of each good than the market supplies (demands).

Note that there is no need to explicitly enforce the bid monotonicity constraints in this ILP formulation:

- "Buy offers must be nonincreasing in k, and sell offers nondecreasing."

The ILP does not need this constraint because prices are assumed to be linear. In effect, the only decisions the ILP makes are how many units of each good to bid on. Hence, the bids (10, 15, 20) and (20, 15, 10) are equivalent.

- "An agent may not offer to sell for less than the price it is willing to buy."

  The ILP would not choose to place both a buy offer and a sell offer on a good if the buy price of that good exceeds the sell price, because that would be unprofitable.

Chapter 10

Robustness to Imperfect Predictions[1]

We undertake an experimental study of heuristics designed for the Travel division of the Trading Agent Competition. Our primary goal is to analyze the performance of the sample average approximation (SAA) heuristic, which is approximately optimal in the decision-theoretic (DT) setting, in this game-theoretic (GT) setting. To this end, we conduct experiments in four settings, three DT and one GT. The relevant distinction between the DT and the GT settings is: in the DT settings, agents' strategies do not affect the distribution of prices. Because of this distinction, the DT experiments are easier to analyze than the GT experiments. Moreover, settings with normally distributed prices, and controlled noise, are easier to analyze than those with competitive equilibrium prices. In the studied domain, analysis of the DT settings with possibly noisy normally distributed prices informs our analysis of the richer DT and GT settings with competitive equilibrium prices. In future work, we plan to investigate whether this experimental methodology—namely, transferring knowledge gained in a DT setting with noisy signals to a GT setting—can be applied to analyze heuristics for playing other complex games.

10.1 Introduction

In the design of autonomous trading agents that buy and sell goods in electronic markets, a variety of interesting computational questions arise. One of the most fundamental is to determine how to bid on goods being auctioned off in separate markets when the agent's valuations for those goods are highly interdependent (i.e., complementary or substitutable). The Trading Agent Competition (TAC) Travel division was designed as a testbed in which to compare and contrast various approaches to this problem [109]. We partake in an empirical investigation of heuristics designed for bidding in the simultaneous auctions that characterize TAC in a simplified TAC-like setting.

At a high-level, the design of many successful TAC agents (for example, Walverine [25], RoxyBot (Greenwald and Boyan 2004 & 2005) and ATTac [98]) can be summarized as: Step 1: *predict*, i.e., build a model

[1] Published as [47].

of the auctions' clearing prices; Step 2: *optimize*, i.e., solve for an (approximately) optimal set of bids, given this model. This chapter is devoted to the study of bidding, that is, the optimization piece of this design. We assume that agents are given price predictions in the form of a black box from which they can sample a vector of predicted prices; such samples are called *scenarios*. Because finding an optimal solution to the bidding problem is not generally tractable, our study centers around a series of *heuristics* that construct bids based on approximations or simplifications. We subject these heuristics to experimental trials within a simplified version of the TAC domain that we find more amenable to experimental study than the full-blown TAC Travel game.

10.2 TAC Travel Game

In this section, we briefly summarize the TAC game. For more details, see `http://www.sics.se/tac/`.

A TAC Travel agent is a simulated travel agent whose task is to organize itineraries for a group of clients to travel to and from TACTown. The agent's objective is to procure "desirable" travel goods as inexpensively as possible. An agent desires goods (i.e., it earns utility for procuring them) to the extent that they comprise itineraries that satisfy its clients' preferences.

Travel goods are sold in simultaneous auctions:

- Flights are sold by the "TAC seller" in dynamic posted-pricing environments. No resale is permitted.
- Hotel reservations are also sold by the "TAC seller," in multi-unit ascending call markets. Specifically, 16 hotel reservations are sold in each hotel auction at the $16th$ highest price. No resale is permitted.
- Agents trade tickets to entertainment events among themselves in continuous double auctions.

Flights and hotel reservations are complementary goods: flights do not garner utility without complementary hotel reservations; nor do hotel reservations garner utility without complementary flights. Tickets to entertainment events, e.g., the Boston Red Sox and the Boston Symphony Orchestra, are substitutable.

Clients have preferred departure and arrival dates, and a penalty is subtracted from the agent's utility for allocating packages that do not match clients' preferences exactly. For example, a penalty of 200 (100 per day) is incurred when a client who wants to depart Monday and arrive on Tuesday is assigned a package with a Monday departure and a Thursday arrival. Clients also have hotel preferences, for the two type of hotels, "good" and "bad." A client's preference for staying at the good rather than the bad hotel is described by a *hotel bonus*, utility the agent accumulates when the client's assigned package includes the good hotel.

10.3 Bidding Heuristics

Our test suite consists of six marginal-utility-based and two sample average approximation heuristics. We present a brief description of these heuristics here. Interested readers are referred to [110] for more detailed explanations.

10.3.1 Marginal-Utility-Based Heuristics

In a second-price auction for a single good, it is optimal for an agent to simply bid its independent value on that good [106]. In simultaneous auctions for multiple goods, however, bidding is not so straightforward because it is unclear how to assign independent values to interdependent goods. Perfectly complementary goods (e.g., an inflight and outflight for a particular client) are worthless in isolation, and perfectly substitutable goods (e.g., rooms in different hotels for the same client on the same day) provide added value only in isolation. Still, an agent might be tempted to bid on each good its *marginal* utility (MU), that is, the incremental value of obtaining that good relative to the collection of goods it already owns or can buy. Many reasonable bidding heuristics (e.g., [45], [46], [98]) incorporate some form of marginal utility bidding.

Definition Given a set of goods X, a valuation function $v : 2^X \rightarrow \mathbb{R}$, and bundle prices $q : 2^X \rightarrow \mathbb{R}$. The *marginal utility* $\mu(x, q)$ of good $x \in X$ is defined as:

$$\mu(x) = \max_{Y \subseteq X \setminus \{x\}} [v(Y \cup \{x\}) - q(Y)] - \max_{Y \subseteq X \setminus \{x\}} [v(Y) - q(Y)]$$

Consistent with TAC Travel, we assume additive prices: that is, in the above equation, the bundle pricing function q returns the sum of the predicted prices of the goods in Y.

Our heuristics actually sample a set of scenarios, not a single vector of predicted prices. We consider two classes of marginal utility heuristics based on how they make use of the information in the scenarios.

Bidding Heuristics that Collapse Available Distributional Information

The following heuristics collapse all scenarios into a single vector of predicted prices, namely the average scenario, and then calculate the marginal utility of each good assuming the other goods can be purchased at the average prices.

StraightMU bids the marginal utility of each good.

TargetMU bids marginal utilities only on the goods in a target set of goods. The target set is one that an agent would optimally purchase at the average prices.

TargetMU* is similar to TargetMU, but calculates marginal utilities assuming only goods from the target set are available. This results in higher bids.

Bidding Heuristics that Exploit Available Distributional Information

The heuristics discussed thus far collapse the distributional information contained in the sample set of scenarios down to a point estimate, thereby operating on approximations of the expected clearing prices. The heuristics described next more fully exploit any available distributional information; they seek bids that are effective across multiple scenarios, not in just the average scenario.

AverageMU calculates the marginal utilities of all goods, once per scenario, and then bids the *average* MU of each good in each auction.

BidEvaluator evaluates K candidate bidding policies on a fixed set of E sample scenarios. The policy that earns the highest total score is selected.

BidEvaluator generates its candidates by making successive calls to the TargetMU heuristic, each time sending it a different scenario to use as its predicted prices.

BidEvaluator* is identical to BidEvaluator, except that its candidate bidding policies are generated by calling TargetMU* instead of TargetMU.

10.3.2 Sample Average Approximation

The problem of bidding under uncertainty—how to bid given a distributional model of predicted prices—is a stochastic optimization problem. The objective is to select bids that maximize the expected value of the difference between the value of the goods the agent wins and the cost of those goods. Formally,

Stochastic Bidding Problem Given a set of goods X, a (combinatorial) valuation function $v : 2^X \to \mathbb{R}$, and a distribution f over clearing prices $p \in \mathbb{R}^X$, the *stochastic bidding problem* is defined as:

$$\max_{\beta \in \mathbb{R}^X} \mathbb{E}_{p \sim f} \left[v(\text{Win}(\beta, p)) - \tilde{p}(\text{Win}(\beta, p)) \right] \tag{10.1}$$

Here, $x \in \text{Win}(\beta, p)$ if and only if $b(x) \geq p(x)$, and $\tilde{p} : 2^X \to \mathbb{R}$ is the *additive* extension of $p \in \mathbb{R}^X$, that is, the real-valued function on bundles defined as follows: $\tilde{p}(Y) = \sum_{x \in Y} p(x)$, for all $Y \subseteq X$.

Sample average approximation (SAA) is a standard way of approximating the solution to a stochastic optimization problem, like bidding under uncertainty. The idea behind SAA is simple: (i) generate a set of sample scenarios, and (ii) solve an approximation of the problem that incorporates only the sample scenarios.

Technically, the TAC Travel bidding problem, in which the goal is to maximize the difference between the value of allocating travel packages to clients and the costs of the goods procured to create those packages, is a stochastic program with integer recourse [68]. Using the theory of large deviations, Ahmed and Shapiro [4] establish the following: the probability that an optimal solution to the sample average approximation of a stochastic program with integer recourse is in fact an optimal solution to the original stochastic program approaches 1 exponentially fast as the number of scenarios $S \to \infty$. Given time and space constraints, however, it is not always possible to sample sufficiently many scenarios to make any reasonable guarantees about the quality of a solution to the sample average approximation.

Our default implementation of SAA which we call SAABottom always bids one of the sampled prices. However, given a set of scenarios, SAA is indifferent between bidding the highest sampled price or any amount above that price: in any case SAA believes it will win in all scenarios. Consequently, we do not know exactly how much SAA is willing to pay when it bids the highest sampled price. In the settings with imperfect price prediction or when SAA is given too few scenarios, it may be desirable to bid above the highest sampled price to increase the chances of winning. For this reason, we introduce a modified SAA

heuristic—SAATop—in which bids equal to the highest sampled price are replaced with the "maximum" bid. In general, this bid is the most the agent is willing to pay. In our domain, this maximum is the sum of the utility bonus (300; see Footnote 3) and, for good hotels, the largest hotel bonus among the agent's clients'.

10.4 Experiments in TAC Travel-like Auctions

We consider four experimental settings: normally distributed prices in two decision-theoretic settings, one with perfect and another with imperfect prediction; and competitive equilibrium (CE) prices in a decision-theoretic setting with perfect prediction and a game-theoretic setting with typically imperfect prediction.

Our experiments were conducted in a TAC Travel-like setting, in which nearly all the standard rules apply.[2] Most notably, we simplified the dynamics of the game. In TAC, flights and entertainment tickets are available continuously at time-varying prices, and hotel auctions close one at a time, providing opportunities for agents to revise their bids on other hotels. In this work, we focus on one-shot auctions. More specifically, we assume all hotels close after one round of bidding.

To reduce variance, we eliminated entertainment trading and simplified flight trading by fixing flight prices at zero.[3,4]

We built a simulator of the TAC server, which can easily be tailored to simulate numerous experimental designs. Our simulator is available for download at `http://www.sics.se/tac/showagents.php`.

Each trial in an experiment (i.e., each simulation run) proceeded in five steps:

1. The agents predict hotel clearing prices in the form of *scenarios* - samples from the predicted distribution of clearing prices.

   - In the settings where prices are normally distributed, the scenarios were sampled from given distributions of predicted prices.
   - In the settings characterized by competitive equilibrium prices, scenarios were generated by simulating simultaneous ascending auctions, as described in Lee *et al.* [68].

2. The agents construct bids using price information contained in the scenarios and submit them.

3. The simulator determines hotel clearing prices, and bids that are equal to or above those clearing prices are deemed winning bids.

   - In the *decision-theoretic* settings, the clearing prices were sampled from given distributions of clearing prices.

[2] For a detailed description of the TAC Travel rules, visit `http://www.sics.se/tac`.

[3] Since we fixed flight prices at zero (instead of roughly 700 for round trip tickets), we adjusted the utility bonus for constructing a valid travel package down from 1000 to 300. That way, our simulation scores fall in the same range as real game scores.

[4] Initially, we ran experiments with flight prices fixed at 350, which is the value close to the average flight price in the TAC Travel game. However the resulting one-shot setting was not interesting as flight tickets represented a very high sunk cost and the dominant hotel bidding strategy was to bid very high on the hotels that would complement the flights in completing travel packages.

Experimental Design

Normally-Distributed Prices	Perfect Prediction	DT
Normally-Distributed Prices	Imperfect Prediction	DT
CE Prices	Perfect Prediction	DT
CE Prices	Imperfect Prediction	GT

Table 10.1: Price predictions were either normally distributed or competitive equilibrium prices. Moreover, they were sometimes perfect and sometimes imperfect. Three of the four experimental setups were decision-theoretic (DT); only the fourth was game-theoretic (GT).

- In the *game-theoretic* setting, each hotels' clearing price was set to the 16th highest bid on that hotel.

4. Agents pay clearing prices for the hotels they win. They use the hotels and free flight tickets to create packages for their clients, based on which they earn the corresponding utilities.

5. Each agent's final score is the difference between its utility and its cost.

The first two steps in the above sequence correspond to the prediction and optimization steps typical of autonomous bidding agents. To carry out step 2, the agents employ heuristics from a test suite that includes the eight bidding heuristics detailed in [110], and summarized above.

Regarding price prediction in step 1, hotel price predictions were perfect in our first and third experimental setups and imperfect in our second and fourth. In the first two, hotel prices were predicted to be normally distributed; in the second two, hotel prices were predicted to be competitive equilibrium prices. Our first three experimental setups were decision-theoretic; the fourth was game-theoretic. In the second setup, we simply tweaked the normal distribution of predicted prices to generate a similar, but distinct, normal distribution of clearing prices. In the fourth setup, the game-theoretic setting, clearing prices were dictated by the outcome of 16th price auctions. Our experimental design is summarized in Table 10.1. All setups, with all settings of the parameters (μ, σ, and λ), were run for 1000 trials.

10.4.1 Heuristic Parameter Settings

The parameter settings we chose for the heuristics are shown in Table 10.2. Breaking down a TAC agent's work into two key steps—price prediction and optimization—the column labeled SG lists the scenario generation (i.e., CE price prediction) times; the column labeled BC lists the bid construction (i.e., optimization) times. The rightmost column lists total runtimes. The goal in choosing these parameter settings was to roughly equalize total runtimes across agents in TAC games.

All experiments were run on AMD Athlon(tm) 64 bit 3800+ dual core processors with 2GB of RAM. All times are reported in seconds, averaged over 1000 games. The machines were not dedicated, which explains why generating 50 scenarios could take anywhere from 8.7 to 9.4 seconds, on average. Presumably, all the heuristics (but most notably, AverageMU, the variants of BidEvaluator, and the SAA heuristics) could benefit from higher settings of their parameters.

Agent	E	S	K	SG	BC	Total
TMU	–	50	–	9.4	1.0	10.4
TMU*	–	50	–	9.0	1.1	10.1
BE	15	–	25	7.0	5.3	12.3
BE*	15	–	25	7.0	4.7	11.7
AMU	–	15	–	2.3	10.2	12.5
SMU	–	50	–	8.7	1.5	10.2
SAABottom	–	50	–	8.8	1.7	10.5
SAATop	–	50	–	9.0	1.6	10.6

Table 10.2: Parameter Settings. E is the number of evaluations, S is the number of scenarios, and K is the number of candidate bidding policies.

We optimized the heuristics that bid only on the goods in a target set to bid ∞ on all flights in that set; they do not bother to calculate the marginal utilities of their desired flights.[5] This helps explain why the bid construction phase within TargetMU and TargetMU* is so fast. StraightMU, and hence AverageMU, are also optimized to stop computing marginal utilities once a good's marginal utility hits zero.

10.4.2 Multiunit Marginal Utility

TAC Travel auctions are multi-unit auctions. For bidding in multi-unit auctions, we extend the definition of marginal utility, originally defined for a single copy of each good, to handle multiple copies of the same good. The marginal utility of the first copy of a good is calculated assuming that no other copies of the good can be had; the marginal utility of the second copy of a good is calculated assuming that the first copy is on hand but that no other copies can be had; and so on.

We assume the set of goods X contains J goods, with K_j copies of each good $1 \leq j \leq J$.

Definition Given a set of goods X, a valuation function $v : 2^X \rightarrow \mathbb{R}$, and a pricing function $q : 2^X \rightarrow \mathbb{R}$. The *marginal utility* $\mu(x_{jk}, X, v, q)$ of the kth copy of good j is given by:

$$\max_{Y \subseteq X \setminus \{x_{j1},\ldots,x_{jK_j}\}} [v(Y \cup \{x_{j1}, \ldots, x_{jk}\}) - q(Y)] \; - \\ \max_{Y \subseteq X \setminus \{x_{j1},\ldots,x_{jK_j}\}} [v(Y \cup \{x_{j1}, \ldots, x_{j,k-1}\}) - q(Y)]$$

In words, the marginal utility of the kth copy of good j is simply the difference between the value of an optimal set of goods to buy, assuming $x_{j1}, \ldots, x_{jk}$ cost 0 and $x_{j,k+1}, \ldots, x_{jN}$ cost ∞, and the value of an optimal set of goods to buy, assuming $x_{j1}, \ldots, x_{j,k-1}$ cost 0 and $x_{jk}, \ldots, x_{jN}$ cost ∞.

Our agent implementations of the marginal-utility-based agents employ this definition.

[5] Note that we ran many more experiments than those reported here. In particular, flight prices were not always zero (e.g., see Footnote 4). Indeed, in many cases it was sensible for the various heuristics to make informed decisions about how to bid on flights.

10.5 Decision-Theoretic Experiments with Perfect Distributional Prediction

Our first experimental setup is decision-theoretic, with prices determined exogenously. Each agent is endowed with perfect distributional information, so that it constructs its bids based on samples drawn from the true price distribution. Under these conditions, it is known that the SAA-based heuristics bid optimally in the limit as $S \to \infty$ [4]. The purpose of conducting experiments in this setting was twofold: (i) to evaluate the performance of the SAA-based heuristics with only finitely many scenarios; and (ii) to evaluate the performance of the MU-based heuristics relative to that of the SAA-based heuristics. We find that both the SAA-based heuristics and certain variants of the MU-based heuristics (primarily, TargetMU* and BidEvaluator*) perform well assuming low variance, but that the SAA-based heuristics and AverageMU outperform all the other heuristics assuming high variance.

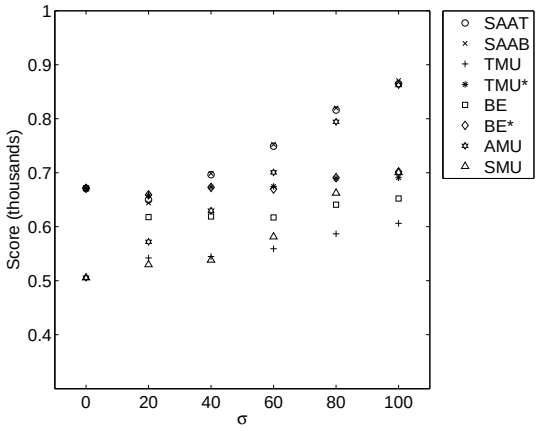


Figure 10.1: Mean Scores. Decision-theoretic setting with perfect distributional prediction.

10.5.1 Setup

Hotel prices were drawn from normal distributions with means[6] $\bar{\mu} = (150, 150, 150, 150, 250, 250, 250, 250)$ constant across experiments and standard deviations $\sigma \in \{0, 20, 40, 60, 80, 100\}$ varying across experiments.

10.5.2 Results

Figure 10.1 depicts the mean scores earned by each agent in each experiment: i.e., for each setting of σ.

The SAA-based agents perform better than most of the agents as variance increases. They gain an advantage by submitting low bids on more goods than necessary in an attempt to win only the goods that are cheap. We refer to this strategy as *hedging*. We see that the SAA agents employ hedging because the number of bids

[6] In this, and all, hotel price vectors, the first four numbers refer to the price of the bad hotel on days 1 through 4, respectively, and the second four numbers refer to the price of the good hotel on days 1 through 4, respectively.

they place increases, their average bids decrease, and the number of hotels they win remains constant as the variance increases. The number of low-priced hotels increases with the variance making hedging especially effective when variance is high.

Recall that target bidders (TargetMU, TargetMU*, BidEvaluator, and BidEvaluator*) bid only on goods in their target set, i.e. they do not hedge. Consequently, failing to win one of the requisite hotels results in not being able to complete a package (most packages are for one-night stays as extending the stay for an extra day is likely to be more expensive than incurring the penalty for deviating from client's preferences). TargetMU and BidEvaluator win fewer and fewer hotels as the variance increases, and hence complete fewer and fewer packages. At the same time the average cost of hotels they win decreases. The agents' scores have a slight upward trend as the benefit from lower cost outweighs the loss from completing fewer packages.

BidEvaluator bids on more hotels than TargetMU when variance is 100. Recall that BidEvaluator chooses the best of K bidding policies. Bidding policies that bid on more hotels score higher because they hedge, implicitly. For example, a policy that bids to reserve two nights for a client may earn a higher score than a policy that bids to reserve one night as the reservation for two nights can be used to create two separate one-night packages if some of the other bids fail.

TargetMU* and BidEvaluator*, the main rivals of the SAA-based agents, do not perform well in this setting. Just like TargetMU and BidEvaluator, TargetMU* and BidEvaluator* bid only on target goods. When variance is low ($\sigma = 20$), bidding high on target good is a good strategy as evidenced by TargetMU*'s and BidEvaluator*'s good performance. As variance increases the agents fail to win some of the target goods. In fact when variance is 100, TargetMU* submits 5.8 bids but wins only 4.8 while BidEvaluator* submits 7.4 bids and wins only 5.2. The average cost of hotels that TargetMU* and BidEvaluator* do win is 50% higher than the prices the SAA-based agents pay per hotel.

Interestingly, AverageMU's strategy happens to be very close to hedging when variance is high. StraightMU submits a lot of bids too but unlike AverageMU does not perform well. StraightMU's bids are higher than AverageMU's resulting in more purchased hotels and higher average hotel cost. The increase in cost that StraightMU incurs compared to AverageMU is not compensated by the increase in utility that extra hotels bring.

In conclusion, the SAA-based agents and AverageMU with their hedging strategy outperform the other agents when variance is high.

10.6 Decision-Theoretic Experiments with Imperfect Distributional Prediction

In our second decision-theoretic experimental setup, the agents construct their bids based on samples drawn from a normal distribution that resembles, but is distinct from, the true distribution. Our intent here is to evaluate the agents' behavior in a controlled setting with imperfect predictions, in order to inform our analysis of their behavior in the game-theoretic setting, where predictions are again imperfect. We find that SAATop performs worse than TargetMU*, and BidEvaluator* at low variance, but outperforms most of the other agents at high variance.

10.6.1 Setup

In these experiments, the *predicted* price distributions were normal with mean values $\bar{\mu}$ = (150, 150, 150, 150, 250, 250, 250, 250), whereas the *clearing* price distributions were normal with mean values $\bar{\mu}+\lambda$. That is, the mean of each predicted distribution differed by λ from the true mean. For example, for $\lambda = -40$, predicted prices were sampled from normal distributions with $\bar{\mu}$ = (150, 150, 150, 150, 250, 250, 250, 250), and clearing prices were sampled from normal distributions with $\bar{\mu}$ = (110, 110, 110, 110, 210, 210, 210, 210). Hence, negative values of λ implied "overprediction." Similarly, positive values of λ implied "underprediction." The λ parameter varied as follows: $\lambda \in \{-40, -30, -20, -10, 0, 10, 20, 30, 40\}$. We chose as standard deviations of the distributions a low setting ($\sigma = 20$) and a high setting ($\sigma = 80$).

In the low (and similarly in the high) deviation experiments the strategies of the agents did not change with λ because the agent received the same predictions for all values of λ. Experiments in this setting evaluate the strategies from the perfect prediction setting with $\sigma = 20$ and $\sigma = 80$ under different distributions of clearing prices as controlled by the values of λ.

10.6.2 Results

Low Variance: $\sigma = 20$ The results assuming low variance are shown in Figure 10.2(a).

Recall from the perfect prediction experiments that the strategy of bidding high on the goods from a target set is as good as hedging when variance is low. In particular, TargetMU* and BidEvaluator* perform as well as the SAA-based agents. We will see that hedging is not a good strategy in the low-variance setting with imperfect prediction while bidding high on the goods in a target set works fairly well.

In an attempt to hedge, the SAA-based agents submit twice as many bids as TargetMU, TargetMU*, BidEvaluator, and BidEvaluator*. The strategy of the SAA-based agents is to bid low hoping to win approximately half the bids. Because predictions are not perfect, the SAA-based agents win too many hotels when prices are lower than expected and too few hotels when prices are higher than expected. Not surprisingly, SAATop, which bids higher than its counterpart, performs worse than SAABottom when prices are lower than expected and better than SAABottom when the opposite it true.

When there is a high degree of overprediction and variance is low, (e.g., when $\lambda = -40$ and $\sigma = 20$), clearing prices are very likely to be below predicted prices. Since TargetMU always bids at least the predicted price, it is likely to win all the hotels it expects to win in this setting, and hence performs well. Consequently, TargetMU*, BidEvaluator, and BidEvaluator* all perform well. In contrast, when prices are often lower than expected, AverageMU and StraightMU win too many goods and thus incur high unnecessary costs.

As λ increases from -40 to -10, AverageMU and StraightMU win fewer unnecessary hotels, which improves their scores. But once λ reaches 0, they fail to win enough hotels, and their utilities decrease as λ increases to 40. TargetMU and BidEvaluator encounter the same difficulty.

TargetMU* and BidEvaluator* bid higher than TargetMU and BidEvaluator; hence, underprediction affects the former pair less than the latter pair.

To summarize, in the low-variance setting TargetMU*'s and BidEvaluator*'s strategy of bidding high on a target set of goods is more robust to imperfect predictions than the strategy of the SAA-based agents that

involves some hedging.

High Variance: $\sigma = 80$ The results assuming high variance are shown in Figure 10.2(b).

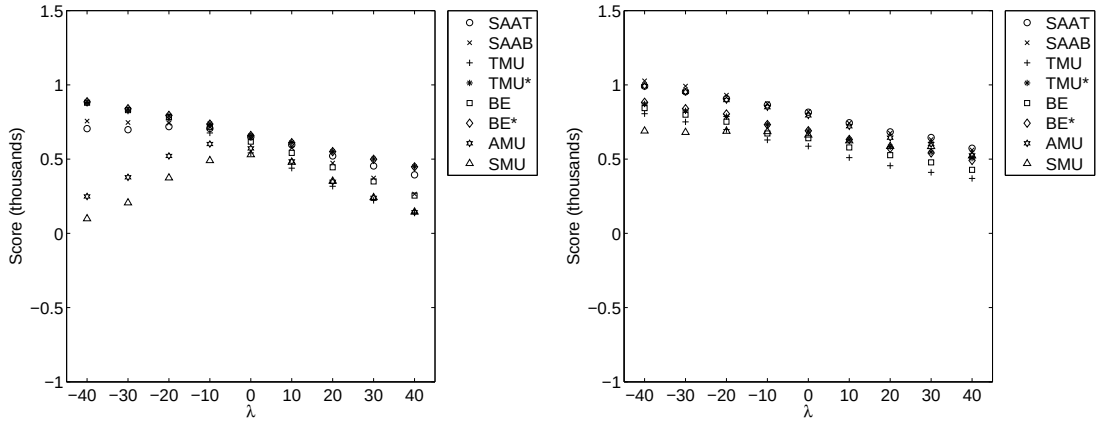
As we observed in the experiments with perfect prediction and $\sigma = 80$, hedging allowed the SAA-based agents to dominate. We are going to see that hedging is effective in the high-variance setting even when predictions are not perfect.

The SAA-based agents submit over four times as many bids as TargetMU, TargetMU*, BidEvaluator, and BidEvaluator*. In contrast to the setting with low variance, high overprediction ($\lambda = -40$) does not cause the SAA-based agents to overspend on hotels. In the high-variance setting the SAA-based agents' bids are 40% lower than in the low-variance setting ($\sigma = 20$) and only one-third of the bids are winning bids.

Similarly, the SAA-based agents perform much better in the high underprediction ($\lambda = 40$) setting when variance is high than when variance is low. In the high-variance setting with underprediction the SAA-based agents win at least as many hotels as the high bidding TargetMU* and BidEvaluator* agents. Although the SAA-based agents bid half the price that TargetMU* and BidEvaluator* bid, a much higher number of bids that the SAA-based agents submit combined with high variance results in a similar number of winning bids.

Performance of the other agents is similar to their performance in the setting with perfect prediction. TargetMU, TargetMU*, BidEvaluator, and BidEvaluator* do not hedge and perform poorly in under and over prediction settings. Target bidders often fail to win some of the target hotels even in the overprediction setting. AverageMU submits a lot of low bids resulting in a well-hedged strategy and the scores that are as high as SAA's for some values of λ. As before, StraightMU wins too many hotels.

In contrast to the setting with low variance and imperfect predictions, the SAA-based agents' hedging strategy works well when there is high variance.



(a) $\sigma = 20$ (b) $\sigma = 80$

Figure 10.2: Mean Scores. Imperfect prediction.

10.7 Experiments with Competitive Equilibrium Prices

In contrast with our first two experimental settings, in which the hotel clearing prices and their corresponding predictions are exogenously determined and hence independent of any game specifics, in our second two experimental settings, both hotel clearing prices and predictions are determined endogenously (i.e., based on features of each game instance). Specifically, following Walverine [25], hotel clearing prices and their corresponding predictions are taken to be approximate *competitive equilibrium* (CE) prices. CE prices are prices at which supply equals demand when all market participants act as price-taking profit maximizers [73]. CE prices need not exist, and likely do not in many of the games studied here. Still, we approximate CE prices as follows: in a market inhabited by its own eight clients and eight randomly sampled clients per competitor, each agent generates a scenario by simulating simultaneous ascending auctions (i.e., increasing prices by some small increment until supply exceeds demand; see Lee *et al.* [68] for details); the resulting prices form a scenario.

10.7.1 Setup

In this context, where hotel price predictions are (roughly) competitive equilibrium prices, we designed two sets of experiments: one decision-theoretic and one game-theoretic. In the former, hotel clearing prices are also the outcome of a simulation of simultaneous ascending auctions, but depend on the actual clients in each game, not some random sampling like the agents' predictions. (Our simulator is more informed than the individual agents.) In the latter, hotel clearing prices are determined by the bids the agents submit. As in TAC Travel, the clearing price is the 16th highest bid (or zero, if fewer than 16 bids are submitted). Note that hotel clearing prices and their respective predictions are not independent of one another in these experiments.

In these experiments games are played with a random number of agents drawn from a binomial distribution with $n = 32$ and $p = 0.5$, with the requisite number of agents sampled uniformly with replacement from the set of eight possible agent types. The agents first sample the number of competitors from the binomial distribution, and then generate scenarios assuming the sampled number of competitors, resampling that number to generate each new scenario.

10.7.2 Decision-Theoretic Experiments

Marginal frequency distributions of CE prices in these experiments have means (109, 126, 126, 107, 212, 227, 227, 210) and standard deviations (47, 37, 37, 46, 50, 41, 41, 49). Standard deviation in this setting is close to 40 making this setting similar to the one with perfect prediction and $\sigma = 40$. The mean hotel prices are approximately 20% lower in this CE setting but we do not expect the difference in mean hotel prices to have a strong effect on the ranking of the agents and attribute the differences in relative results to the different structure of prices: unlike the setting with normally distributed prices, CE prices are not independent.

SAATop, SAABottom, TargetMU*, and BidEvaluator* are among the best agents in this CE setting (see Figure 10.3). However, StraightMU and especially AverageMU perform poorly. AverageMU and StraightMU submit more bids and win more hotels than the other agents, but cannot create as many packages as the top-scoring agents. This is because (i) CE prices of substitutable goods are similar, and (ii) marginal utilities of

substitutable goods are similar. As a result, AverageMU and StraightMU bid almost the same amount on all substitutable goods and either win or lose all of them.

SAA-based agents employ some hedging but do not perform significantly better than the non-hedging heuristics TargetMU* and BidEvaluator*.

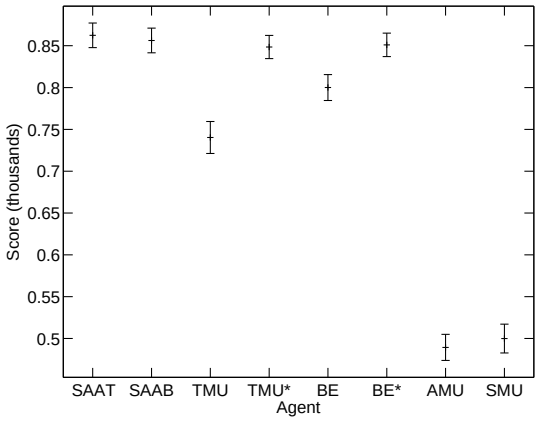


Figure 10.3: Mean scores and confidence intervals. Decision-theoretic setting with CE price prediction.

10.7.3 Game-Theoretic Experiments

The predicted prices are the same as in the decision-theoretic experiments with CE prices and 32 agents: means (109, 126, 126, 107, 212, 227, 227, 210) and standard deviations (47, 37, 37, 46, 50, 41, 41, 49). Marginal frequency distributions of clearing prices have means (91, 98, 100, 91, 198, 186, 187, 197) and standard deviations (41, 33, 32, 40, 50, 56, 54, 50). L1-norm of the difference between mean price vectors is 197. Predicted prices are slightly higher (by about 20) than the clearing prices. This is similar to the decision-theoretic setting with overprediction ($\lambda = -20$) and medium deviation (between 20 and 80).

Indeed, we find that the results in this setting (see Figure 10.4) are similar to the results in the decision-theoretic setting with imperfect prediction and high variance: $\lambda = -20$ and $\sigma = 80$ (see the ranking of agents for $\lambda = -20$ in Figure 10.2(b)). The ranking of non-SAA agents is almost the same in both settings. A notable exception is AverageMU, which performs much worse in the game-theoretic setting for the reasons described above. SAATop and SAABottom are the best agents in this setting, with SAABottom performing slightly better.

10.8 Summary and Discussion of Experimental Results

In our experiments, we evaluated the performance of various bidding heuristics in simultaneous auctions. Based on our findings, we summarize the performance of the heuristics analyzed as follows:

- SAATop and SAABottom perform well in all settings except for the setting with imperfect prediction and low variance. SAATop and SAABottom are especially effective in high-variance settings because

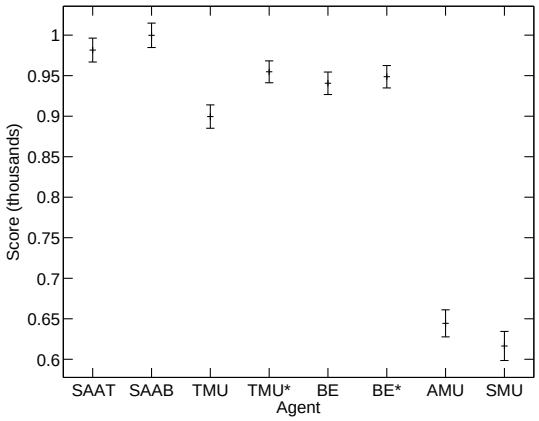


Figure 10.4: Mean scores and confidence intervals. Game-theoretic setting with CE price prediction.

they are able to take advantage of hedging opportunities.

- TargetMU and BidEvaluator are competitive only in the settings with low variance and high overprediction. BidEvaluator outperforms TargetMU in high-variance settings.
- TargetMU* and BidEvaluator* perform well in the settings with low variance.
- AverageMU performs well in the settings with independent prices.
- StraightMU performs worse than the other heuristics.

We can also make the following observations about the various bidding behaviors:

- SAABottom, SAATop, and AMU place low bids on many goods, intending to win whatever sells at cheap prices. These heuristics incur high penalties for not satisfying their clients' precise preferences.
- TargetMU, TargetMU*, BidEvaluator, and BidEvaluator* place higher bids but on fewer goods, namely those for which their clients have clear preferences. These heuristics incur lower penalties, but risk alienating some clients, by not allocating them any travel packages at all.[7]

The performance of SAA is known to approach optimality as the number of scenarios approaches ∞ in decision-theoretic settings. We investigated the viability of two SAA heuristics with only finitely-many scenarios in both decision-theoretic and game-theoretic settings. Our first and third experimental settings (with normally distributed and competitive equilibrium prices, assuming perfect price prediction) established the viability of these heuristics in decision-theoretic settings with only finitely-many scenarios. Our fourth experimental setting established the viability of these heuristics (again, with only finitely-many scenarios, but in addition) in a rich game-theoretic setting.

[7] No penalty is incurred when a client is not allocated any package at all. (Of course, no utility is awarded either.)

10.9 Related Work

The test suite considered here is far from exhaustive. In this section, we mention several heuristics that were not included in our study—some TAC-specific; some more general—and the reasons for their exclusion.

The creators of the ATTac agent [98] propose using AverageMU for TAC hotel bidding. ATTac also employs distributional information about hotel prices to determine the benefit of postponing flight purchases until hotel prices are known; this additional functionality, while certainly of interest, is not applicable to the one-shot auction setting studied here.

WhiteBear's [104] TAC hotel bids are computed by taking a weighted average of the current price and the marginal utility of each hotel. The particular weights, which were fine-tuned based on historical competition data, varied with time. In a one-shot setting, WhiteBear's strategy essentially reduces to TargetMU: it is too risky to bid anything lower.

SouthamptonTAC [53] and Mertacor [101] focus on hotel price prediction, and do not thoroughly analyze bidding. SouthamptonTAC uses fuzzy reasoning to predict how hotel prices change during the game.

Unlike the heuristics studied in this chapter, Walverine's [25] bidding strategy incorporates some game-theoretic reasoning. Specifically, Walverine analytically calculates the distribution of marginal utilities of the other agents' clients and bids a best-response to this distribution. The authors implicitly assume that the other agents bid marginal utilities (i.e., act decision-theoretically) and only their agent bids a best-response (i.e., acts game-theoretically). We learned from the study reported in this paper that SAA can be a successful bidding heuristic in certain markets. Following Walverine's line of thought, we can imagine bidding a best-response to a distribution of SAA bids. However, if this bidding strategy were successful, we would have to assume that other agents would act game-theoretically as well; that is, they would also play a best-response to a distribution of SAA bids. We may then seek a fixed point of this process. This line of inquiry could be fascinating, but any approach based on this insight of Walverine's warrants a detailed study of its own.

Aside from TAC Travel there is a rich literature on bidding in other settings. We reference a few papers here, highlighting some of the settings that have been studied. We are not aware of any papers that address the problem of bidding in multiple one-shot auctions for both complementary and substitutable goods. Gerding *et al.* ([38]) describes a strategy for bidding in simultaneous one-shot second-price auctions selling perfect substitutes. Byde, Priest, & Jennings ([20]) consider the decision-theoretic problem of bidding in multiple auctions with overlapping closing times. Their model treats all goods as indistinguishable (i.e., winning any n goods results in utility $v(n)$). Krishna & Rosenthal ([65]) characterize a symmetric equilibrium for the case of one-shot simultaneous auctions with indistinguishable complementary goods (i.e. $v(n) \geq nv(1)$).

10.10 Conclusion

The primary purpose of this work was to show that using as much distributional information as possible is an effective approach to bidding in TAC Travel-like one-shot simultaneous auctions. Most TAC Travel agents used point price predictions or employed little distributional information about prices in constructing their bids. Some of the difficulties with using distributional price predictions include the inaccuracy of and the

high computational cost of optimizing with respect to distributional predictions. We showed experimentally that the SAA heuristic, which uses more distributional information than the other heuristics in our test suite, is one of the best heuristics in the GT setting.

The underlying research question motivating this line of inquiry was: how can we facilitate the search for heuristics that perform well against a variety of competing agents in complex games? Analyzing the performance of an individual agent in a game-theoretic setting is complicated because each agent's performance is affected by the strategies of the others, and can vary dramatically with the mix of participants. Others tackling this problem in the TAC Travel domain have employed more direct game-theoretic analysis techniques based on equilibrium computations (e.g., Vetsikas *et al.* [105] and Jordan, Kiekintveld, & Wellman [56]). In contrast, we first used systematic decision-theoretic analysis to help us understand some of the intrinsic properties of our bidding heuristics, before attempting any game-theoretic analysis. We found that certain properties of the heuristics that may have been hard to identify in game-theoretic settings, such as how they perform in conditions of over- vs. under-prediction, carried over from our DT to our GT settings.

In summary, the methodology advocated in this chapter for analyzing game-theoretic heuristics is this: first, evaluate the heuristic in DT settings with perfect and imperfect predictions; and second, measure the accuracy of the agent's predictions in GT experiments and use the corresponding DT analysis to inform the analysis of the GT results. It remains to test this methodology in other complex games, such as TAC SCM [5].

10.11 Acknowledgments

This research was funded by NSF Grant #IIS-0133689 and the Samsung Foundation of Culture.

10.12 Appendix

We include game statistics collected from our experiments with CE prices to illustrate the type of data we used in our analyses. Statistics for other settings and further data can be found in Lee ([69]).

Agent	SAAT	SAAB	TMU	TMU*	BE	BE*	AMU	SMU
Score	863	859	742	847	799	850	488	499
Utility	1935	1838	1420	1811	1619	1830	1805	1902
Cost	1071	979	677	964	819	980	1316	1402
Penalty	383	348	251	309	237	331	229	234
# of Clients without a Package	1.60	2.01	3.40	2.20	2.89	2.09	2.48	2.28
# of Hotel Bids	11.0	10.7	6.3	6.3	5.8	6.8	42.2	35.5
Average Hotel Bid	270	187	187	292	233	280	106	125
Total Hotel Bonus	400	389	292	383	324	390	378	422
# of Hotels Won	6.7	6.2	4.6	5.8	5.1	5.9	9.3	9.6
# of Unused Hotels	0.0	0.0	0.0	0.0	0.0	0.0	1.2	1.3
Average Hotel Cost	159.6	159.0	147.4	166.5	160.5	166.0	140.9	145.6

Decision-theoretic setting

Agent	SAAT	SAAB	TMU	TMU*	BE	BE*	AMU	SMU
Score	981	999	899	954	938	948	652	617
Utility	2057	2007	1708	1885	1796	1904	2386	2397
Cost	1075	1007	808	931	857	955	1734	1779
Penalty	437	418	287	345	272	375	338	358
# of Clients without a Package	1.13	1.33	2.53	1.90	2.34	1.75	0.64	0.60
# of Hotel Bids	10.8	10.7	6.3	6.3	5.9	6.8	42.1	35.5
Average Hotel Bid	274	188	187	292	234	281	106	125
Total Hotel Bonus	435	426	354	400	371	404	518	536
# of Hotels Won	7.6	7.1	5.5	6.1	5.7	6.3	12.5	12.5
# of Unused Hotels	0.0	0.0	0.0	0.0	0.0	0.0	1.9	1.8
Average Hotel Cost	142.4	142.4	147.8	152.6	151.5	152.8	138.3	142.8

Game-theoretic setting

Table 10.3: Game statistics for experiments with CE prices

Bibliography

[1] Z. Abrams. Revenue maximization when bidders have budgets. In *SODA*, pages 1074–1082. ACM Press, 2006.

[2] Z. Abrams, O. Mendelevitch, and J. A. Tomlin. Optimal delivery of sponsored search advertisements subject to budget constraints. In *ACM EC*, pages 272–278, 2007.

[3] G. Aggarwal, A. Goel, and R. Motwani. Truthful auctions for pricing search keywords. In *Proc. ACM EC*, pages 1–7, June 2006.

[4] S. Ahmed and A. Shapiro. The sample average approximation method for stochastic programs with integer recourse. Optimization Online, `http://www.optimization-online.org`, 2002.

[5] R. Arunachalam and N. M. Sadeh. The supply chain trading agent competition. *Electronic Commerce Research and Applications*, 4(1):66–84, 2005.

[6] K. Asdemir. Dynamics of bidding in search engine auctions: An analytical investigation. In *Proc. 2nd Workshop on Sponsored Search Auctions*, 2006.

[7] E. Aurell, M. Boman, M. Carlsson, J. Eriksson, N. Finne, S. Janson, P. Kreuger, and L. Rasmusson. A trading agent built on constraint programming. In *Eighth International Conference of the Society for Computational Economics: Computing in Economics and Finance*, Aix-en-Provence, 2002.

[8] M. J. Bailey. The demand revealing process: To distribute the surplus. *Public Choice*, 91(2):107–26, April 1997.

[9] M. Benisch, J. Andrews, and N. Sadeh. Pricing for customers with probabilistic valuations as a continuous knapsack problem. In *ACM International Conference Proceeding Series*, volume 156, pages 38 – 46, 2006.

[10] M. Benisch, A. Greenwald, I. Grypari, R. Lederman, V. Naroditskiy, and M. Tschantz. Botticelli: A supply chain management agent. In *Third International Conference on Autonomous Agents and Multiagent Systems*, volume 3, pages 1174–1181, July 2004.

[11] M. Benisch, A. Greenwald, V. Naroditskiy, and M. Tschantz. A stochastic programming approach to TAC SCM. In *Fifth ACM Conference on Electronic Commerce*, pages 152–160, May 2004.

[12] R. Bent and P. V. Hentenryck. Scenario Based Planning for Partially Dynamic Vehicle Routing Problems with Stochastic Customers. *Operations Research, to appear*, 2001.

[13] R. Bent and P. V. Hentenryck. Dynamic Vehicle Routing with Stochastic requests. In *International Joint Conference on Artificial Intelligence (IJCAI)*, Acapulco, Mexico, 2003.

[14] J. Birge and F. Louveaux. *Introduction to Stochastic Programming*. Springer, New York, 1997.

[15] C. Borgs, J. Chayes, O. Etesami, N. Immorlica, K. Jain, and M. Mahdian. Dynamics of bid optimization in online advertisement auctions. In *Proc. SSA*, pages 531–540, 2007.

[16] C. Borgs, J. T. Chayes, N. Immorlica, M. Mahdian, and A. Saberi. Multi-unit auctions with budget-constrained bidders. In *Proc. ACM EC*, pages 44–51, 2005.

[17] K. M. Bretthauer and B. Shetty. The nonlinear resource allocation problem. *Operations Research*, 43:670–683, 1995.

[18] N. Buchbinder, K. Jain, and J. S. Naor. Online primal-dual algorithms for maximizing ad-auctions revenue. In *Proc. ESA*, pages 253–264, 2007.

[19] D. A. Burke, K. N. Brown, O. Koyuncu, B. Hnich, and A. Tarim. Foreseer: A constraint based agent for tac-scm. In *Trading Agent Competition - Supply Chain Management, IJCAI '05, Edinburgh, Scotland*, August 2005.

[20] A. Byde, C. Preist, and N. R. Jennings. Decision procedures for multiple auctions. In *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 613–620, New York, NY, USA, 2002. ACM.

[21] M. Cary, A. Das, B. Edelman, I. Giotis, K. Heimerl, and A. R. Karlin. Greedy bidding strategies for keyword auctions. In *ACM EC*, 2007.

[22] R. Cavallo. Optimal decision-making with minimal waste: Strategyproof redistribution of vcg payments. In *Proc. of the 5th Int. Joint Conf. on Autonomous Agents and Multi Agent Systems (AAMAS'06)*, Hakodate, Japan, 2006.

[23] D. Chakrabarty, Y. Zhou, and R. Lukose. Budget-constrained bidding in keyword auctions and online knapsack problems. In *Proc. SSA*, May 2007.

[24] H. Chang, R. Givan, and E. Chong. On-line Scheduling Via Sampling. In *Artificial Intelligence Planning and Scheduling (AIPS)*, pages 62–71, Breckenridge, Colorado, 2000.

[25] S. Cheng, E. Leung, K. Lochner, K.O'Malley, D. Reeves, L. Schvartzman, and M. Wellman. Walverine: A Walrasian trading agent. *Decision Support Systems*, 39(2):169–184, 2005.

[26] M. S.-Y. Chwe. The discrete bid first auction. *Economics Letters*, 31(4):303–306, December 1989.

[27] P. Cramton. Simultaneous ascending auctions. In P. Cramton, Y. Shoham, and R. Steinberg, editors, *Combinatorial Auctions*. MIT Press, 2006.

[28] G. de Clippel, V. Naroditskiy, and A. Greenwald. Destroy to save. In *EC 09: Proceedings of the 10th ACM Conference on Electronic Commerce*, 2009.

[29] S. De Vries, R. Vohra, C. for Mathematical Studies in Economics, and M. Science. Combinatorial auctions: A survey. *INFORMS Journal on Computing*, 15(3):284–309, 2003.

[30] B. Edelman and M. Ostrovsky. Strategic bidder behavior in sponsored search auctions. In *Proc. 1st Workshop on Sponsored Search Auctions*, Vancouver, Canada, June 2005.

[31] B. Edelman, M. Ostrovsky, and M. Schwarz. Internet advertising and the generalized second price auction: Selling billions of dollars worth of keywords. *American Economic Review*, 2007.

[32] B. Efron and R. Tibshirani. *An Introduction to the Bootstrap*. Chapman and Hall, 1993.

[33] J. Eriksson, N. Finne, and S. Janson. The sics TAC classic server and agentware. In preparation, 2005.

[34] B. Faltings. A Budget-balanced, Incentive-compatible Scheme for Social Choice. In P. Faratin and J. A. Rodriguez-Aguilar, editors, *Agent-Mediated Electronic Commerce VI*, volume 3435, pages 30–43. Springer, 2005.

[35] J. Feldman, S. Muthukrishnan, M. Pal, and C. Stein. Budget optimization in search-based advertising auctions. In *ACM EC*, pages 40–49, 2007.

[36] C. Fritschi and K. Dorer. Agent-oriented software engineering for successful TAC participation. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 45–46, 2002.

[37] D. Fudenberg and J. Tirole. *Game Theory*. MIT Press, Cambridge, 1991.

[38] E. Gerding, R. Dash, D. Yuen, and N. R. Jennings. Optimal bidding strategies for simultaneous vickrey auctions with perfect substitutes. In *8th Int. Workshop on Game Theoretic and Decision Theoretic Agents*, pages 10–17, ["lib/utils:month_12586" not defined] 2006.

[39] E. Gerding, Z. Rabinovich, A. Byde, E. Elkind, and N. Jennings. Approximating mixed Nash equilibria using smooth fictitious play in simultaneous auctions. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 3*, pages 1577–1580. International Foundation for Autonomous Agents and Multiagent Systems Richland, SC, 2008.

[40] A. Goldberg, J. Hartline, A. Karlin, M. Saks, and A. Wright. Competitive auctions. *Games and Economic Behavior*, 55(2):242–269, 2006.

[41] J. R. Green and J.-J. Laffont. *Incentives in public decision-making*. North Holland, New York, 1979.

[42] A. Greenwald. Bidding marginal utility in simultaneous auctions. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, pages 1463–1464, August 2003.

[43] A. Greenwald. Bidding marginal utility in simultaneous auctions. In *International Joint Conference on Artificial Intelligence Workshop on Trading Agent Design and Analysis*, August 2003.

[44] A. Greenwald. Bid determination in simultaneous auctions. Technical Report 16, Brown University, 2005.

[45] A. Greenwald and J. Boyan. Bidding under uncertainty: Theory and experiments. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, pages 209–216, July 2004.

[46] A. Greenwald and J. Boyan. Bidding algorithms for simultaneous auctions: A case study. *Journal of Autonomous Agents and Multiagent Systems*, 10(1):67–89, 2005.

[47] A. Greenwald, V. Naroditskiy, and S. Lee. Bidding Heuristics for Simultaneous Auctions: Lessons from TAC Travel. In *AAAI-08 Workshop on Trading Agent Design and Analysis*, July 2008.

[48] A. Greenwald, V. Naroditskiy, T. Odean, M. Ramirez, E. Sodomka, J. Zimmerman, and C. Cutler. *Marginal Bidding: An Application of the Equimarginal Principle to Bidding in TAC SCM*, page 217. Springer Berlin Heidelberg, 2009.

[49] M. Guo and V. Conitzer. Worst-case optimal redistribution of vcg payments. In *EC '07: Proceedings of the 8th ACM conference on Electronic commerce*, pages 30–39, New York, NY, USA, 2007. ACM.

[50] M. Guo and V. Conitzer. Better redistribution with inefficient allocation in multi-unit auctions with unit demand. In *EC '08: Proceedings of the 9th ACM conference on Electronic commerce*, pages 210–219, New York, NY, USA, 2008. ACM.

[51] M. Guo and V. Conitzer. Optimal-in-expectation redistribution mechanisms. In *AAMAS '08: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*, pages 1047–1054, Richland, SC, 2008.

[52] M. He and N. Jennings. SouthamptonTAC: Designing a successful trading agent. In *Proceedings of the Fifteenth European Conference on Artificial Intelligence*, pages 8–12, 2002.

[53] M. He and N. R. Jennings. SouthamptonTAC: An adaptive autonomous trading agent. *ACM Trans. Interet Technol.*, 3(3):218–235, 2003.

[54] D. S. Hochbaum. A nonlinear knapsack problem. In *Operations Research Letters*, volume 17, pages p. 103–110(8), April 1995.

[55] D. S. Hochbaum and J. G. Shanthikumar. Convex separable optimization is not much harder than linear optimization. *J. ACM*, 37(4):843–862, 1990.

[56] P. R. Jordan, C. Kiekintveld, and M. P. Wellman. Empirical game-theoretic analysis of the tac supply chain game. In *AAMAS '07: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, pages 1–8, New York, NY, USA, 2007. ACM.

[57] L. Kaelbling, M. Littman, and A. Cassandra. Planning and Acting in Partially Observable Stochastic Domains. *Artificial Intelligence*, 101 (1-2):99–134, 1998.

[58] D. Kehagias, P. Toulis, and P. Mitkas. A long-term profit seeking strategy for continuous double auctions in a trading agent competition. In *Fourth Hellenic Conference on Artificial Intelligence*, Heraklion, 2006.

[59] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, 2004.

[60] C. Kiekintveld, M. Wellman, S. Singh, J. Estelle, Y. Vorobeychik, V. Soni, and M. Rudary. Distributed feedback control for decision making on supply chains. In *Fourteenth International Conference on Automated Planning and Scheduling*, page To Appear, 2004.

[61] B. Kitts and B. Leblanc. Optimal bidding on keyword auctions. *Electronic Markets*, 14(3):186–201, 2004.

[62] A. Kleywegt, A. Shapiro, and T. H. de Mello. The sample average approximation method for stochastic discrete optimization. *SIAM Journal of Optimization*, 12:479–502, 2001.

[63] A. J. Kleywegt and J. D. Papastavrou. The dynamic and stochastic knapsack problem. *Operations Research*, 46(1):17–35, 1998.

[64] V. Krishna. *Auction Theory*. Academic Press, March 2002.

[65] V. Krishna and R. W. Rosenthal. Simultaneous auctions with synergies. *Games and Economic Behavior*, 17(1):1–31, November 1996. available at http://ideas.repec.org/a/eee/gamebe/v17y1996i1p1-31.html.

[66] S. Lahaie. An analysis of alternative slot auction designs for sponsored search. In *Proc. 7th ACM Conference on Electronic Commerce*, pages 218–227, Ann Arbor, MI, June 2006.

[67] J. Ledyard, D. Porter, and A. Rangel. Experiments testing multiobject allocation mechanisms. *Journal of Economics and Management Strategy*, 6:639–665, 1997.

[68] S. Lee, A. Greenwald, and V. Naroditskiy. Roxybot-06: An $(SAA)^2$ TAC travel agent. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 1378–1383, January 2007.

[69] S. J. Lee. Comparison of bidding algorithms in simultaneous auctions. B.S. honors thesis, Brown University, `http://list.cs.brown.edu/publications/theses/ugrad/`, 2007.

[70] B. Lev and P. Zarowin. The boundaries of financial reporting and how to extend them. *Journal of Accounting Research*, 37:353–385, 1999.

[71] G. S. Lueker. Average-case analysis of off-line and on-line knapsack problems. *J. of Algorithms*, 29(2):277–305, 1998.

[72] A. Marchetti-Spaccamela and C. Vercellis. Stochastic on-line knapsack problems. *Mathematical Programming*, 68:73–104, 1995.

[73] A. Mas-Colell, M. Whinston, and J. Green. *Microeconomic Theory*. Oxford University Press, New York, 1995.

[74] T. Mathews and A. Sengupta. Sealed bid second price auctions with discrete bidding. *Applied Economics Research Bulletin*, Special Issue I (Auctions):31–52, 2008.

[75] A. Mehta, A. Saberi, U. V. Vazirani, and V. V. Vazirani. Adwords and generalized on-line matching. In *Proc. FOCS*, pages 264–273, 2005.

[76] S. T. M.K. Kozlov and L. Khachian. Polynomial solvability of convex quadratic programming. *Doklady Akad. Nauk SSR*, 5:1051–1053, 1979.

[77] H. Moulin. *Axioms of Cooperative Decision Making (Econometric Society Monographs)*. Cambridge University Press, July 1991.

[78] H. Moulin. Almost budget-balanced vcg mechanisms to assign multiple objects. *Journal of Economic Theory*, 2008.

[79] V. Naroditskiy and A. R. Greenwald. Using Iterated Best-Response to Find Bayes-Nash Equilibria in Auctions. In *AAAI*, pages 1894–1895. AAAI Press, 2007.

[80] J. Nash. Non-cooperative games. *Annals of Mathematics*, 54:286–295, 1951.

[81] D. Nikovski and M. Branch. Marginalizing Out Future Passengers in Group Elevator Control. *Uncertainity in Artificial Intelligence (UAI)*, 2003.

[82] N. Nisan and A. Ronen. Algorithmic Mechanism Design. *Games and Economic Behavior*, 35:166–196, 2001.

[83] J. Noga and V. Sarbua. An online partially fractional knapsack problem. In *Proc. 8th Sym. Parallel Architectures, Algorithms and Networks*, pages 108–112, 2005.

[84] T. Odean, V. Naroditskiy, A. Greenwald, and J. Donaldson. Marginal Bidding: An Application of the Equimarginal Principle to Bidding in TAC SCM. In *AAA-07 Workshop on Trading Agent Design and Analysis*, July 2007.

[85] M. Osborne and A. Rubinstein. *A Course in Game Theory*. MIT Press, Cambridge, 1994.

[86] J. D. Papastavrou, S. Rajagopalan, and A. J. Kleywegt. The dynamic and stochastic knapsack problem with deadlines. *Management Science*, 42(12):1706–1718, 1996.

[87] D. Pardoe and P. Stone. TacTex-03: A supply chain management agent. *SIGecom Exchanges*, 4(3):In this issue, 2004.

[88] D. Pardoe and P. Stone. Bidding for Customer Orders in TAC SCM. In *Agent Mediated Electronic Commerce*, volume 4, pages 143–157, 2005.

[89] M. Patriksson. A survey on the continuous nonlinear resource allocation problem. *European Journal of Operational Research*, 127(1):1–46, February 2008.

[90] C. Plott. Experiments testing multiobject allocation mechanisms. *Journal of Economics and Management Strategy*, 6:605–638, 1997.

[91] R. Porter, Y. Shoham, and M. Tennenholtz. Fair imposition. *Journal of Economic Theory*, 118(2):209 – 228, 2004.

[92] D. M. Reeves and M. P. Wellman. Computing best-response strategies in infinite games of incomplete information. In *UAI*, 2004.

[93] D. M. Reeves, M. P. Wellman, J. K. Mackie-Mason, and A. Osepayshvili. Exploring bidding strategies for market-based scheduling. *Decis. Support Syst.*, 39(1):67–85, March 2005.

[94] S. M. Ross. *Simulation*. Academic Press, third edition, 2002.

[95] P. Rusmevichientong and D. P. Williamson. An adaptive algorithm for selecting profitable keywords for search-based advertising services. In *Proc. 7th ACM conference on Electronic commerce*, pages 260–269, 2006.

[96] J. A. R. P. Sardinha, R. L. Milidiú, P. M. Paranhos, P. M. Cunha, and C. J. P. de Lucena. An agent based architecture for highly competitive electronic markets. In *Proceedings of the Eighteenth International Florida Artificial Intelligence Research Society Conference, Clearwater Beach, Florida, USA*, pages 326–332, 2005.

[97] A. Shapiro and T. Homen-De-Mello. On rate convergence of monte carlo approximations of stochastic programs. *SIAM Journal on Optimization*, 11:70–86, 2001.

[98] P. Stone, R. Schapire, M. Littman, J. Csirik, and D. McAllester. Decision-theoretic bidding based on learned density models in simultaneous, interacting auctions. *Journal of Artificial Intelligence Research*, 19:209–242, 2003.

[99] A. Sureka and P. R. Wurman. Using tabu best-response search to find pure strategy nash equilibria in normal form games. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 1023–1029, New York, NY, USA, 2005. ACM.

[100] B. K. Szymanski and J.-S. Lee. Impact of ROI on bidding and revenue in sponsored search advertisement auctions. In *Proc. 2nd Workshop on Sponsored Search Auctions*, Ann Arbor, MI, 2006.

[101] P. Toulis, D. Kehagias, and P. Mitkas. Mertacor: A successful autonomous trading agent. In *Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1191–1198, Hakodate, 2006.

[102] H. R. Varian. Position auctions. *International Journal of Industrial Organization*, 2007, to appear.

[103] I. Vetsikas and B. Selman. WhiteBear: An empirical study of design tradeoffs for autonomous trading agents. In *Game Theoretic Decision Theoretic Agents Workshop*, August 2002.

[104] I. A. Vetsikas and B. Selman. A principled study of the design tradeoffs for autonomous trading agents. In *AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 473–480, New York, NY, USA, 2003. ACM.

[105] I. A. Vetsikas and B. Selman. Bayes-nash equilibria for mth price auctions with multiple closing times. *SIGecom Exch.*, 6(2):27–36, 2007.

[106] W. Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *Journal of Finance*, 16:8–37, 1961.

[107] L. Walras. *Eléments d'économie politique pure*. L. Corbaz, Lausanne, 1874.

[108] M. Wellman, D. Reeves, K. Lochner, and Y. Vorobeychik. Price prediction in a Trading Agent Competition. *Artificial Intelligence Research*, 21:19–36, 2004.

[109] M. Wellman, P. Wurman, K. O'Malley, R. Bangera, S. Lin, D. Reeves, and W. Walsh. A Trading Agent Competition. *IEEE Internet Computing*, April 2001.

[110] M. P. Wellman, A. Greenwald, and P. Stone. *Autonomous Bidding Agents: Strategies and Lessons from the Trading Agent Competition*. MIT Press, 2007.

[111] M. P. Wellman, D. M. Reeves, K. M. Lochner, and R. Suri. Searching for Walverine 2005. In *IJCAI-05 Workshop on Trading Agent Design and Analysis*, number 3937 in Lecture Notes on Artificial Intelligence, pages 157–170. Springer, 2005.

[112] J. Yu. *Discrete approximation of continuous allocation mechanism*. PhD thesis, California Institute of Technology, 1999.

[113] Y. Zhou and R. Lukose. Vindictive bidding in keyword auctions. In *Proc. 9th Int. Conf. Electronic Commerce*, pages 141–146, Minneapolis, MN, August 2007.

[114] Y. Zhou and V. Naroditskiy. Algorithm for stochastic multiple-choice knapsack problem and keywords bidding. In *WWW08: Workshop on Targeting and Ranking for Online Advertising*, 2008.