Computational Problems in Multiagent Systems

by

Albert Xin Jiang

B.Sc., The University of British Columbia, 2003

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

The Faculty of Graduate Studies

(Computer Science)

The University Of British Columbia

March 31, 2006

Abstract

There has been recent interest from the computer science community on multiagent systems, where multiple autonomous agents, each with their own utility functions, act according to their own interests. In this thesis, we apply techniques developed in other areas of CS to solve two computational problems in multiagent systems:

Action Graph Games: Action Graph Games (AGGs), first proposed in [Bhat and Leyton-Brown 2004], are a fully expressive game representation which can compactly express strict and context-specific independence and anonymity structure in players' utility functions. We present an efficient algorithm for computing expected payoffs under mixed strategy profiles. This algorithm runs in time polynomial in the size of the AGG representation (which is itself polynomial in the number of players when the in-degree of the action graph is bounded). We also present an extension to the AGG representation which allows us to compactly represent a wider variety of structured utility functions.

Learning and planning in online auction environments: There is much active research into the design of automated bidding agents, particularly for environments that involve multiple auctions. These settings are complex partly because an agent's optimal strategy depends on information about other bidder's preferences. When bidders' valuation distributions are not known ex ante, machine learning techniques can be used to approximate them from historical data. It is a characteristic feature of auctions, however, that information about some bidders' valuations is systematically concealed. This occurs in the sense that some bidders may fail to bid at all because the asking price exceeds their valuations, and also in the sense that a high bidder may not be compelled to reveal her valuation. Ignoring these "hidden bids" can introduce bias into the estimation of valuation distributions. To overcome this problem, we proposed an EM-base algorithm. We validate the algorithm experimentally using agents that react to their environments both decision-theoretically and game-theoretically, using both synthetic and real-world (eBay) datasets. We show that our approach estimates bidders' valuation distributions and the distribution over the true number of bidders significantly more accurately than more straightforward density estimation techniques. Bidding agents using the estimated distributions from our EM approach were able to outperform bidding agents using the straightforward estimates, in both decision-theoretic and game-theoretic settings.

Contents

List of Figures

Glossary

$C_s(m)$	the contribution of action s to node m, 20
D	a configuration, 5
$D(s)$	the number of players that chose action s, 5
$D^{(s)}$	a configuration over $\nu(s)$, 5
G	the action graph, 5
N	the set of players, 5
S	the set of distinct actions; also the set of action nodes in the action graph, 5
S_i	player i's set of actions, 5
$V^i_{s_i}(\sigma_{-i})$	expected utility to agent i for playing pure strategy s_i, given that all other agents play the mixed strategy profile σ_{-i}, 10
Δ	the set of configurations, 5
$\Delta^{(s,i)}$	the set of configurations over $\nu(s)$ given that player i has played s, 5
$\Delta^{(s_i,i)}(\sigma_{-i})$	the set of configurations over $\nu(s_i)$ that have positive probability of occurring under the mixed strategy (s_i, σ_{-i}), 14
$\Delta^{(s)}$	the set of configurations over $\nu(s)$ given that some player has played s, 5
Σ	set of all mixed strategy profiles, 9
Σ_i	set of all mixed strategies of player i, 9
$\mathbf{s}$	an action profile, 5
$\mathcal{I}$	the maximum in-degree of the action graph, 7
σ	mixed strategy profile, 9
σ_i	mixed strategy of player i, 9
$\nabla V^{i,j}_{s_i,s_j}(\overline{\sigma})$	payoff Jacobian's entry at row (i, s_i) and column (j, s_j), 21
ν	the neighbor relation of the action graph. $\nu(s)$ is the set of neighbors of s., 5
n	the number of players, 5
s_i	an action of player i, 5
u	utility function , 5

u^s	the utility function associated with action s; stores utilities given that the current player has played s, 6

Chapter 1

Introduction

1.1 Multiagent Systems

There has been recent interest from the computer science community on (non-cooperative) multiagent systems, where multiple autonomous agents, each with their own utility functions, act according to their own interests. These situations can be modeled as games, and analyzed using game theory. Game theory has received a great deal of study, and is perhaps the dominant paradigm in microeconomics. However, the computational aspects of game theory have received relatively less attention until recently.

The intersection of computer science and game theory includes many exciting topics. One line of research is to apply game theory to resource allocation problems in computing systems. Internet-based systems, due to their distributed and multiagent nature, are a popular area for game-theoretic analysis. Examples include network routing [Roughgarden and Tardos 2004], CPU scheduling [Regev and Nisan 1998; Waldspurger et al. 1992], peer-to-peer file sharing [Golle et al. 2001] and search engines' ranking systems [Altman and Tennenholtz 2005].

Another line of research, which includes the topic of this thesis, is to apply CS techniques to analyze and solve computational problems in multiagent systems. One type of problems is the computation of game-theoretic *solutions* such as Nash equilibria (and restricted versions such as pure-strategy equilibria and social welfare maximizing equilibria) and correlated equilibria, given a description of the game. Recently, complexity analysis from theoretical computer science has been used to analyze these problems [Conitzer and Sandholm 2003; Goldberg and Papadimitriou 2005; Daskalakis et al. 2005; Chen and Deng 2005]. There are also some recent research on practical algorithms for computing Nash equilibria [Koller et al. 1994; Porter et al. 2004].

The representation of games has also received recent interest. Traditionally, simultaneous-move games are represented in normal form, and dynamic games are represented in extensive form. However, for large games, these representations are impractically large, and any non-trivial computation on these representations would be intractable. Fortunately, most large games of any practical interest have highly structured payoff functions, thus it is possible to compactly represent them. One influential approach is to represent structure in payoffs as graphs. AI researchers have used graphs to represent structure in joint probability distributions (as Bayes nets or Markov random fields) with great success. For multiagent systems, several graphical representations have been proposed to represent the (strict) independence between players' payoffs;

this includes multi-agent influence diagrams [Koller and Milch 2001] for dynamic games and graphical games [Kearns et al. 2001] for simultaneous-move games. Another type of graphical representations focus on *context-specific* independencies in agents' utility functions – that is, games in which agents' abilities to affect each other depend on the actions they choose. This includes local effect games [Leyton-Brown and Tennenholtz 2003] and action graph games [Bhat and Leyton-Brown 2004]; the latter is the topic of Chapter 2 of this thesis.

Another type of computational problems often encountered in multiagent systems are learning problems. Game-theoretic solution concepts such as Nash equilibria assume that the players have perfect knowledge about the game being played, and are perfectly rational. In practice these conditions often do not hold. How should agents act in such environments? This is often cast as a multiagent reinforcement learning problem: instead of assuming other agents to be perfectly rational, assume instead that agent tries to learn an optimal policy through repeated interaction with the other players. Many approaches have been proposed [Bowling and Veloso 2001; Bowling 2004; Powers and Shoham 2004]. Most of these multiagent reinforcement learning approaches study settings of repeated games where players know the game being played. However, in many situations such as auctions, each player does not know the complete payoff function, and each player has certain private information about the payoffs. In the case of auctions, the private information are players' valuations on the item(s) being auctioned. If we know the joint probability distribution of the valuations, we can formulate the situation as a Bayesian game. Usually we do not know the distribution of valuations; instead the bidding history of previous auctions are available, and we could try to estimate the distribution from data. This machine learning problem is an essential component of the problem of building automated bidding agents, which has attracted interest partly due to the Trading Agent Competitions (TAC-Classic and TAC Supply Chain Management) [Wellman et al. 2002]. In Chapter 3 of this thesis, we look at another auction setting with practical interest, namely online auction systems such as the popular eBay. Learning the distributions in this setting presents unique challenges due to the fact that not all bids are shown in the bidding history.

1.2 Overview

My thesis contains results from two research projects on computational problems in game theory and multiagent systems. One is a way of compactly representing games in general, and the other is about a learning problem in a specific kind of multiagent systems, namely online auction environments. The two projects may seem to be about vastly different things, but a common theme is that in both cases we apply techniques developed in other areas of CS (graphical models and dynamic programming in the former, and the Expectation-Maximization algorithm in the latter) to solve computational problems in multiagent systems.

Chapter 2 is about Action Graph Games (AGGs). First proposed in [Bhat and Leyton-Brown 2004], AGGs are a graphical representation of games that

exploits the context-specific independence and anonymity structure in many games. Using AGGs we are able to compactly represent large structured games with many players, and furthermore efficiently compute expected payoffs under mixed strategies, which is an essential step in many game-theoretic computations, e.g. best response, Nash equilibria and correlated equilibria. In this thesis, we make several significant improvements to to results in [Bhat and Leyton-Brown 2004]. First we significantly improved the algorithm for computing expected payoffs of AGGs. Now our dynamic programming algorithm is able to compute expected payoffs in polynomial time with respect to the size of the representation, whereas in [Bhat and Leyton-Brown 2004] a polynomial time algorithm was found for only symmetric games with symmetric strategies. Furthermore, we extended the AGG representation to include function nodes in the action graph. This allows us to compactly represent a larger class of context-specific independence structure in games. Results of computational experiments on structured games confirm our theoretical predictions of compactness and computational speedup.

In Chapter 3 we try to solve the problem of learning bidders' valuation and number distributions in online auction environments. There is much active research into the design of automated bidding agents, particularly for environments that involve multiple auctions. These settings are complex partly because an agent's optimal strategy depends on information about other bidder's preferences. When bidders' valuation distributions are not known ex ante, machine learning techniques can be used to approximate them from historical data. It is a characteristic feature of auctions, however, that information about some bidders' valuations is systematically concealed. This occurs in the sense that some bidders may fail to bid at all because the asking price exceeds their valuations, and also in the sense that a high bidder may not be compelled to reveal her valuation. Ignoring these "hidden bids" can introduce bias into the estimation of valuation distributions. To overcome this problem, we proposed an EM-based algorithm. We validate the algorithm experimentally using agents that react to their environments both decision-theoretically and game-theoretically, using both synthetic and real-world (eBay) datasets. We show that our approach estimates bidders' valuation distributions and the distribution over the true number of bidders significantly more accurately than more straightforward density estimation techniques. Bidding agents using the estimated distributions from our EM approach were able to outperform bidding agents using the straightforward estimates, in both decision-theoretic and game-theoretic settings.

Chapter 2

Action Graph Games

2.1 Introduction

Game-theoretic[1] models have recently been very influential in the computer science community. In particular, simultaneous-action games have received considerable study, which is reasonable as these games are in a sense the most fundamental. In order to analyze these models, it is often necessary to compute game-theoretic quantities ranging from expected utility to Nash equilibria.

Most of the game theoretic literature presumes that simultaneous-action games will be represented in normal form. This is problematic because quite often games of interest have a large number of players and a large set of action choices. In the normal form representation, we store the game's payoff function as a matrix with one entry for each player's payoff under each combination of all players' actions. As a result, the size of the representation grows exponentially with the number of players. Even if we had enough space to store such games, most of the computations we'd like to perform on these exponential-sized objects take exponential time.

Fortunately, most large games of any practical interest have highly structured payoff functions, and thus it is possible to represent them compactly. (Intuitively, this is why humans are able to reason about these games in the first place: we understand the payoffs in terms of simple relationships rather than in terms of enormous look-up tables.) One influential class of representations exploit strict independencies between players' utility functions; this class include graphical games [Kearns et al. 2001], multi-agent influence diagrams [Koller and Milch 2001], and game nets [LaMura 2000]. A second approach to compactly representing games focuses on *context-specific* independencies in agents' utility functions – that is, games in which agents' abilities to affect each other depend on the actions they choose. Since the context-specific independencies considered here are conditioned on actions and not agents, it is often natural to also exploit *anonymity* in utility functions, where each agent's utilities depend on the distribution of agents over the set of actions, but not on the identities of the agents. Examples include congestion games [Rosenthal 1973] and local effect games (LEGs) [Leyton-Brown and Tennenholtz 2003]. Both of these representations make assumptions about utility functions, and as a result cannot represent arbitrary games. Bhat and Leyton-Brown [2004] introduced

[1] A version of this chapter has been submitted for publication. Jiang, A.X. and Leyton-Brown, K. (2006) A Polynomial-Time Algorithm for Action-Graph Games. Submitted to AAAI.

action graph games (AGGs). Similar to LEGs, AGGs use graphs to represent the context-specific independencies of agents' utility functions, but unlike LEGs, AGGs can represent arbitrary games. Bhat & Leyton-Brown proposed an algorithm for computing expected payoffs using the AGG representation. For AGGs with bounded in-degree, their algorithm is exponentially faster than normal-form-based algorithms, yet still exponential in the number of players.

In this chapter we make several significant improvements to results in [Bhat and Leyton-Brown 2004]. In Section 2.3, we present an improved algorithm for computing expected payoffs. Our new algorithm is able to better exploit anonymity structure in utility functions. For AGGs with bounded in-degree, our algorithm is polynomial in the number of players. In Section 2.4, we extend the AGG representation by introducing *function nodes*. This feature allows us to compactly represent a wider range of structured utility functions. We also describe computational experiments in Section 2.6 which confirm our theoretical predictions of compactness and computational speedup.

2.2 Action Graph Games

2.2.1 Definition

An action-graph game (AGG) is a tuple $\langle N, \mathbf{S}, \nu, u \rangle$. Let $N = \{1, \ldots, n\}$ denote the set of agents. Denote by $\mathbf{S} = \prod_{i \in N} S_i$ the set of action profiles, where $\prod$ is the Cartesian product and S_i is agent i's set of actions. We denote by $s_i \in S_i$ one of agent i's actions, and $\mathbf{s} \in \mathbf{S}$ an action profile.

Agents may have actions in common. Let $S \equiv \bigcup_{i \in N} S_i$ denote the set of distinct actions choices in the game. Let Δ denote the set of *configurations* of agents over actions. A configuration $D \in \Delta$ is an ordered tuple of $|S|$ integers $(D(s), D(s'), \ldots)$, with one integer for each action in S. For each $s \in S$, $D(s)$ specifies the number of agents that chose action $s \in S$. Let $\mathcal{D} : \mathbf{S} \mapsto \Delta$ be the function that maps from an action profile $\mathbf{s}$ to the corresponding configuration D. These shared actions express the game's anonymity structure: agent i's utility depends only on her action s_i and the configuration $\mathcal{D}(\mathbf{s})$.

Let G be the *action graph*: a directed graph having one node for each action $s \in S$. The neighbor relation is given by $\nu : S \mapsto 2^S$. If $s' \in \nu(s)$ there is an edge from s' to s. Let $D^{(s)}$ denote a configuration over $\nu(s)$, i.e. $D^{(s)}$ is a tuple of $|\nu(s)|$ integers, one for each action in $\nu(s)$. Intuitively, agents are only counted in $D^{(s)}$ if they take an action which is an element of $\nu(s)$. $\Delta^{(s)}$ is the set of configurations over $\nu(s)$ given that some player has played s.[2] Similarly we define $\mathcal{D}^{(s)} : \mathbf{S} \mapsto \Delta^{(s)}$ which maps from an action profile to the corresponding configuration over $\nu(s)$.

The action graph expresses *context-specific independencies* of utilities of the game: $\forall i \in N$, if i chose action $s_i \in S$, then i's utility depends only on the

[2] If action s is in multiple players' action sets (say players i, j), and these action sets do not completely overlap, then it is possible that the set of configurations given that i played s (denoted $\Delta^{(s,i)}$) is different from the set of configurations given that j played s. $\Delta^{(s)}$ is the union of these sets of configurations.

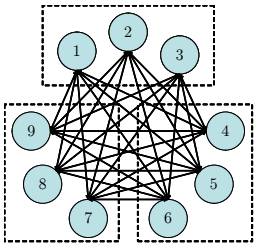

Figure 2.1: AGG representation of an arbitrary 3-player, 3-action game

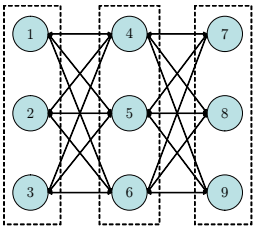

Figure 2.2: AGG representation of a 3-player, 3-action graphical game

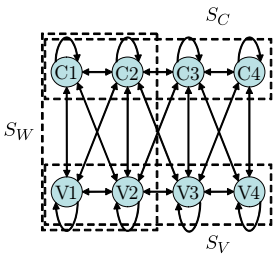

Figure 2.3: AGG representation of the ice cream vendor game

numbers of agents who chose actions connected to s, which is the configuration $\mathcal{D}^{(s_i)}(\mathbf{s})$. In other words, the configuration of actions not in $\nu(s_i)$ does not affect i's utility.

We represent the agents' utilities using a tuple of $|S|$ functions $u \equiv (u^s, u^{s'}, \ldots)$, one for each action $s \in S$. Each u^s is a function $u^s : \Delta^{(s)} \mapsto \mathbb{R}$. So if agent i chose action s, and the configuration over $\nu(s)$ is $D^{(s)}$, then agent i's utility is $u^s(D^{(s)})$. Observe that all agents have the same utility function, i.e. conditioned on choosing the same action s, the utility each agent receives does not depend on the identity of the agent. For notational convenience, we define $u(s, D^{(s)}) \equiv u^s(D^{(s)})$ and $u_i(\mathbf{s}) \equiv u(s_i, \mathcal{D}^{(s_i)}(\mathbf{s}))$.

2.2.2 Examples

Any arbitrary game can be encoded as an AGG as follows. Create a unique node s_i for each action available to each agent i. Thus $\forall s \in S,\ D(s) \in \{0, 1\}$, and $\forall i,\ \sum_{s \in S_i} D(s)$ must equal 1. The distribution simply indicates each agent's action choice, and the representation is no more or less compact than the normal form (see Section 2.2.3 for a detailed analysis).

Example 1. *Figure 2.1 shows an arbitrary 3-player, 3-action game encoded as an AGG. As always, nodes represent actions and directed edges represent membership in a node's neighborhood. The dotted boxes represent the players' action sets: player 1 has actions 1, 2 and 3; etc. Observe that there is always an edge between pairs of nodes belonging to different action sets, and that there is never an edge between nodes in the same action set.*

In a graphical game [Kearns et al. 2001] nodes denote agents and there is an edge connecting each agent i to each other agent whose actions can affect i's utility. Each agent then has a payoff matrix representing his local game with neighboring agents; this representation is more compact than normal form whenever the graph is not a clique. Graphical games can be represented as AGGs by replacing each node i in the graphical game by a distinct cluster of nodes S_i representing the action set of agent i. If the graphical game has an

edge from i to j, create edges so that $\forall s_i \in S_i, \forall s_j \in S_j,\ s_i \in \nu(s_j)$. The resulting AGG representations are as compact as the original graphical game representations.

Example 2. *Figure 2.2 shows the AGG representation of a graphical game having three nodes and two edges between them (i.e., player 1 and player 3 do not directly affect each others' payoffs). The AGG may appear more complex than the graphical game; in fact, this is only because players' actions are made explicit.*

The AGG representation becomes even more compact when agents have actions in common, with utility functions depending only on the *number* of agents taking these actions rather than on the *identities* of the agents.

Example 3. *The action graph in Figure 2.3 represents a setting in which n vendors sell chocolate or vanilla ice creams, and must choose one of four locations along a beach. There are three kinds of vendors: n_C chocolate (C) vendors, n_V vanilla vendors, and n_W vendors that can sell both chocolate and vanilla, but only on the west side. Chocolate (vanilla) vendors are negatively affected by the presence of other chocolate (vanilla) vendors in the same or neighboring locations, and are simultaneously positively affected by the presence of nearby vanilla (chocolate) vendors. Note that this game exhibits context-specific independence without any strict independence, and that the graph structure is independent of n.*

Other examples of compact AGGs that cannot be compactly represented as graphical games include: location games, role formation games, traffic routing games, product placement games and party affiliation games.

2.2.3 Size of an AGG Representation

We have claimed that action graph games provide a way of representing games compactly. But what exactly is the size of an AGG representation? And how does this size grow as the number of agents n grows? From the definition of AGG in Section 2.2.1, we observe that we need the following to completely specify an AGG:

- The set of agents $N = \{1, \ldots, n\}$. This can be specified by the integer n.
- The set of actions S.
- Each agent's action set $S_i \subseteq S$.
- The action graph G. The set of nodes is S, which is already specified. The neighbor relation ν can be straightforwardly represented as neighbor lists: for each node $s \in S$ we specify its list of neighbors $\nu(s) \subseteq S$. The space required is $\sum_{s \in S} |\nu(s)|$, which is bounded by $|S|\mathcal{I}$, where $\mathcal{I} = \max_s |\nu(s)|$, i.e. the maximum in-degree of the action graph.

- For each action s, the utility function $u^s : \Delta^{(s)} \mapsto \mathbb{R}$. We need to specify a utility value for each distinct configuration $D^{(s)} \in \Delta^{(s)}$. The set of configurations $\Delta^{(s)}$ can be derived from the action graph, and can be sorted in lexicographical order. So we do not need to explicitly specify $\Delta^{(s)}$; we can just specify a list of $|\Delta^{(s)}|$ utility values that correspond to the (ordered) set of configurations.[3] $|\Delta^{(s)}|$, the number of distinct configurations over $\nu(s)$, in general does not have a closed-form expression. Instead, we consider the operation of extending all agents' action sets via $\forall i : S_i \mapsto S$. Now the number of configurations over $\nu(s)$ is an upper bound on $|\Delta^{(s)}|$. The bound is the number of (ordered) combinatorial compositions of $n-1$ (since one player has already chosen s) into $|\nu(s)|+1$ nonnegative integers, which is $\frac{(n-1+|\nu(s)|)!}{(n-1)!|\nu(s)|!}$. Then the total space required for the utilities is bounded from above by $|S|\frac{(n-1+\mathcal{I})!}{(n-1)!\mathcal{I}!}$.

Therefore the size of an AGG representation is dominated by the size of its utility functions, which is bounded by $|S|\frac{(n-1+\mathcal{I})!}{(n-1)!\mathcal{I}!}$. If $\mathcal{I}$ is bounded by a constant as n grows, the representation size grows like $O(|S|n^{\mathcal{I}})$, i.e. polynomially with respect to n.

The AGG representation achieves compactness by exploiting two types of structure in the utilities:

1. **Anonymity:** agent i's utility depends only on her action s_i and the configuration (i.e. number of players that play each action), but not on the identities of the players. Since the number of configurations $|\Delta|$ is usually less than the number of action profiles $|\mathbf{S}| = \prod_i |S_i|$ and is never greater, we need fewer numbers to represent the utilities in AGG compared to the normal form.

2. **Context-specific independence:** for each node $s \in S$, the utility function u^s only needs to be defined over $\Delta^{(s)}$. Since $|\Delta^{(s)}|$ is usually less than $|\Delta|$ and is never greater, this further reduces the numbers we need to specify.

For each AGG, there exists a unique *induced normal form* representation with the same set of players and $|S_i|$ actions for each i; its utility function is a matrix that specifies each player i's payoff for each possible action profile $\mathbf{s} \in \mathbf{S}$. This implies a space complexity of $n\prod_{i=1}^{n}|S_i|$. When $S_i \equiv S$ for all i, this becomes $n|S|^n$, which grows exponentially with respect to n. The number of payoff values stored in an AGG representation is always less or equal to the number of payoff values in the induced normal form representation. This is because for each entry in the normal form which represents i's utility under action profile $\mathbf{s}$, there exists a unique action profile $\mathbf{s}$ in the AGG with the

[3] This is the most compact way of representing the utility functions, but does not provide easy random access of the utilities. Therefore, when we want to do computation using AGG, we may convert each utility function u^s to a data structure that efficiently implements a mapping from sequences of integers to (floating-point) numbers, (e.g. tries, hash tables or Red-Black trees), with space complexity in the order of $O(\mathcal{I}|\Delta^{(s)}|)$.

corresponding action for each player. This $\mathbf{s}$ induces a unique configuration $\mathcal{D}(\mathbf{s})$ over the AGG's action nodes. By construction of the AGG utility functions, $\mathcal{D}(\mathbf{s})$ together with s_i determines a unique utility $u^{s_i}(\mathcal{D}^{(s_i)}(\mathbf{s}))$ in the AGG. Furthermore, there are no entries in the AGG utility functions that do not correspond to any action profile (s_i, s_{-i}) in the normal form. This means that there exists a many-to-one mapping from entries of normal form to utilities in the AGG. Of course, the AGG representation has the extra overhead of representing the action graph, which is bounded by $|S|\mathcal{I}$. But asymptotically, AGG's space complexity is never worse than the equivalent normal form.

2.3 Computing with AGGs

One of the main motivations of compactly representing games is to do efficient computation on the games. We have introduced AGG as a compact representation of games; now we would like to exploit the compactness of the AGG representation when we do computation. We focus on the computational task of computing expected payoffs under a mixed strategy profile. Besides being important in itself, this task is an essential component of many game-theoretic applications, e.g. computing best responses, Govindan and Wilson's continuation methods for finding Nash equilibria [Govindan and Wilson 2003; Govindan and Wilson 2004], the simplicial subdivision algorithm for finding Nash equilibria [van der Laan et al. 1987], and finding correlated equilibria using Papadimitriou's algorithm [Papadimitriou 2005].

Besides exploiting the compactness of the representation, we would also like to be able to exploit the fact that quite often the mixed strategy profile given will have small *support*. The support of a mixed strategy σ_i is the set of pure strategies played with positive probability (i.e. $\sigma_i(s_i) > 0$). Quite often games have Nash equilibria with small support. Porter et al. [2004] proposed algorithms that explicitly search for Nash equilibria with small support. In other algorithms for computing Nash equilibria such as Govindan-Wilson and simplicial subdivision, quite often we will also be computing expected payoffs for mixed strategy profiles with small support. Our algorithm appropriately exploits strategy profiles with small supports.

2.3.1 Notation

Let $\varphi(X)$ denote the set of all probability distributions over a set X. Define the set of mixed strategies for i as $\Sigma_i \equiv \varphi(S_i)$, and the set of all mixed strategy profiles as $\Sigma \equiv \prod_{i \in N} \Sigma_i$. We denote an element of Σ_i by σ_i, an element of Σ by σ, and the probability that i plays action s as $\sigma_i(s)$.

Define the expected utility to agent i for playing pure strategy s_i, given that all other agents play the mixed strategy profile σ_{-i}, as

$$V^i_{s_i}(\sigma_{-i}) \equiv \sum_{\mathbf{s}_{-i} \in \mathbf{S}_{-i}} u_i(s_i, \mathbf{s}_{-i}) \Pr(\mathbf{s}_{-i} | \sigma_{-i}). \tag{2.1}$$

where $\Pr(\mathbf{s}_{-i}|\sigma_{-i}) = \prod_{j \neq i} \sigma_j(s_j)$ is the probability of s_{-i} under the mixed strategy σ_{-i}.

The set of i's pure strategy best responses to a mixed strategy profile σ_{-i} is $\arg\max_s V_s^i(\sigma_{-i})$, and hence the full set of i's pure and mixed strategy best responses to σ_{-i} is

$$BR_i(\sigma_{-i}) \equiv \varphi(\arg\max_s V_s^i(\sigma_{-i})). \tag{2.2}$$

A strategy profile σ is a Nash equilibrium iff

$$\forall i \in N,\ \sigma_i \in BR_i(\sigma_{-i}). \tag{2.3}$$

2.3.2 Computing $V_{s_i}^i(\sigma_{-i})$

Equation (2.1) is a sum over the set S_{-i} of action profiles of players other than i. The number of terms is $\prod_{j \neq i} |S_j|$, which grows exponentially in n. Thus Equation (2.1) is an exponential time algorithm for computing $V_{s_i}^i(\sigma_{-i})$. If we were using the normal form representation, there really would be $|S_{-i}|$ different outcomes to consider, each with potentially distinct payoff values, so evaluation Equation (2.1) is the best we could do for computing $V_{s_i}^i$.

Can we do better using the AGG representation? Since AGGs are fully expressive, representing a game without any structure as an AGG would not give us any computational savings compared to the normal form. Instead, we are interested in structured games that have a compact AGG representation. In this section we present an algorithm that given any i, s_i and σ_{-i}, computes the expected payoff $V_{s_i}^i(\sigma_{-i})$ in time polynomial with respect to the size of the AGG representation. In other words, our algorithm is efficient if the AGG is compact, and requires time exponential in n if it is not. In particular, recall that for classes of AGGs whose in-degrees are bounded by a constant, their sizes are polynomial in n. As a result our algorithm will be polynomial in n for such games.

First we consider how to take advantage of the context-specific independence structure of the AGG, i.e. the fact that i's payoff when playing s_i only depends on the configurations in the neighborhood of i. This allows us to *project* the other players' strategies into smaller action spaces that are relevant given s_i. This is illustrated in Figure 2.4, using the ice cream vendor game (Figure 2.3). Intuitively we construct a graph from the point of view of an agent who took a particular action, expressing his indifference between actions that do not affect his chosen action. This can be thought of as inducing a context-specific graphical game. Formally, for every action $s \in S$ define a reduced graph $G^{(s)}$ by including only the nodes $\nu(s)$ and a new node denoted $\emptyset$. The only edges included in $G^{(s)}$ are the directed edges from each of the nodes $\nu(s)$ to the node s. Player j's action s_j is projected to a node $s_j^{(s)}$ in the reduced graph $G^{(s)}$ by the following mapping:

$$s_j^{(s)} \equiv \begin{cases} s_j & s_j \in \nu(s) \\ \emptyset & s_j \notin \nu(s) \end{cases}. \tag{2.4}$$

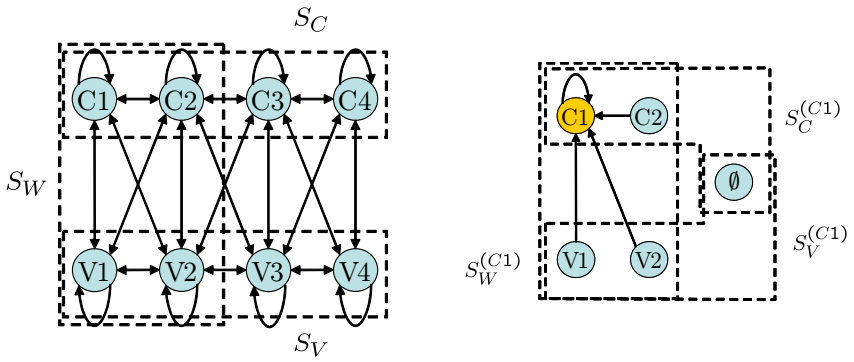

Figure 2.4: Projection of the action graph. Left: action graph of the ice cream vendor game. Right: projected action graph and action sets with respect to the action C1.

In other words, actions that are not in $\nu(s)$ (and therefore do not affect the payoffs of agents playing s) are projected to $\emptyset$. The resulting *projected* action set $S_j^{(s)}$ has cardinality at most $\min(|S_j|, |\nu(s)| + 1)$.

We define the set of mixed strategies on the projected action set $S_j^{(s)}$ by $\Sigma_j^{(s)} \equiv \varphi(S_j^{(s)})$. A mixed strategy σ_j on the original action set S_j is projected to $\sigma_j^{(s)} \in \Sigma_j^{(s)}$ by the following mapping:

$$\sigma_j^{(s)}(s_j^{(s)}) \equiv \begin{cases} \sigma_j(s_j) & s_j \in \nu(s) \\ \sum_{s' \in S_i \setminus \nu(s)} \sigma_j(s') & s_j^{(s)} = \emptyset \end{cases} . \tag{2.5}$$

So given s_i and σ_{-i}, we can compute $\sigma_{-i}^{(s_i)}$ in $O(n|S|)$ time in the worst case. Now we can operate entirely on the projected space, and write the expected payoff as

$$V_{s_i}^i(\sigma_{-i}) = \sum_{s_{-i}^{(s_i)} \in S_{-i}^{(s_i)}} u(s_i, \mathcal{D}^{(s_i)}(s_i, s_{-i})) \Pr(s_{-i}^{(s_i)} | \sigma_{-i}^{(s_i)})$$

where $\Pr(s_{-i}^{(s_i)} | \sigma_{-i}^{(s_i)}) = \prod_{j \neq i} \sigma_j^{(s_i)}(s_j^{(s_i)})$. The summation is over $S_{-i}^{(s_i)}$, which in the worst case has $(|\nu(s_i)| + 1)^{(n-1)}$ terms. So for AGGs with strict or context-specific independence structure, computing $V_{s_i}^i(\sigma_{-i})$ this way is much faster than doing the summation in (2.1) directly. However, the time complexity of this approach is still exponential in n.

Next we want to take advantage of the anonymity structure of the AGG. Recall from our discussion of representation size that the number of distinct configurations is usually smaller than the number of distinct pure action profiles. So ideally, we want to compute the expected payoff $V_{s_i}^i(\sigma_{-i})$ as a sum over the

possible configurations, weighted by their probabilities:

$$V_{s_i}^i(\sigma_{-i}) = \sum_{D^{(s_i)} \in \Delta^{(s_i,i)}} u_i(s_i, D^{(s_i)}) Pr(D^{(s_i)} | \sigma^{(s_i)}) \tag{2.6}$$

where $\sigma^{(s_i)} \equiv (s_i, \sigma_{-i}^{(s_i)})$ and

$$\Pr(D^{(s_i)} | \sigma^{(s_i)}) = \sum_{\mathbf{s}: \mathcal{D}^{(s_i)}(\mathbf{s}) = D^{(s_i)}} \prod_{j=1}^{N} \sigma_j(s_j) \tag{2.7}$$

which is the probability of $D^{(s_i)}$ given the mixed strategy profile $\sigma^{(s_i)}$. Equation (2.6) is a summation of size $|\Delta^{(s_i,i)}|$, the number of configurations given that i played s_i, which is polynomial in n if $\mathcal{I}$ is bounded. The difficult task is to compute $\Pr(D^{(s_i)} | \sigma^{(s_i)})$ for all $D^{(s_i)} \in \Delta^{(s_i,i)}$, i.e. the probability distribution over $\Delta^{(s_i,i)}$ induced by $\sigma^{(s_i)}$. We observe that the sum in Equation (2.7) is over the set of all action profiles corresponding to the configuration $D^{(s_i)}$. The size of this set is exponential in the number of players. Therefore directly computing the probability distribution using Equation (2.7) would take exponential time in n. Indeed this is the approach proposed in [Bhat and Leyton-Brown 2004].

Can we do better? We observe that the players' mixed strategies are independent, i.e. σ is a product probability distribution $\sigma(\mathbf{s}) = \prod_i \sigma_i(s_i)$. Also, each player affects the configuration D independently. This structure allows us to use dynamic programming (DP) to efficiently compute the probability distribution $\Pr(D^{(s_i)} | \sigma^{(s_i)})$. The intuition behind our algorithm is to apply one agent's mixed strategy at a time. In other words, we add one agent at a time to the action graph. Let $\sigma_{1\ldots k}^{(s_i)}$ denote the projected strategy profile of agents $\{1, \ldots, k\}$. Denote by $\Delta_k^{(s_i)}$ the set of configurations induced by actions of agents $\{1, \ldots, k\}$. Similarly denote $D_k^{(s_i)} \in \Delta_k^{(s_i)}$. Denote by P_k the probability distribution on $\Delta_k^{(s_i)}$ induced by $\sigma_{1\ldots k}^{(s_i)}$, and by $P_k[D]$ the probability of configuration D. At iteration k of the algorithm, we compute P_k from P_{k-1} and $\sigma_k^{(s_i)}$. After iteration n, the algorithm stops and returns P_n. The pseudocode of our DP algorithm is shown as Algorithm 1.

Each $D_k^{(s_i)}$ is represented as a sequence of integers, so P_k is a mapping from sequences of integers to real numbers. We need a data structure to manipulate such probability distributions over configurations (sequences of integers) which permits quick lookup, insertion and enumeration. An efficient data structure for this purpose is a *trie*. Tries are commonly used in text processing to store strings of characters, e.g. as dictionaries for spell checkers. Here we use tries to store strings of integers rather than characters. Both lookup and insertion complexity is linear in $|\nu(s_i)|$. To achieve efficient enumeration of all elements of a trie, we store the elements in a list, in the order of their insertions.

Algorithm 1 Computing the induced probability distribution $\Pr(D^{(s_i)}|\sigma^{(s_i)})$.

Algorithm ComputeP
Input: s_i, $\sigma^{(s_i)}$
Output: P_n, which is the distribution $Pr(D^{(s_i)}|\sigma^{(s_i)})$ represented as a trie.
$D_0^{(s_i)} = (0, \ldots, 0)$
$P_0[D_0^{(s_i)}] = 1.0$ // Initialization: $\Delta_0^{(s_i)} = \{D_0^{(s_i)}\}$
for $k = 1$ to n **do**
  Initialize P_k to be an empty trie
  for all $D_{k-1}^{(s_i)}$ from P_{k-1} **do**
    for all $s_k^{(s_i)} \in S_k^{(s_i)}$ such that $\sigma_k^{(s_i)}(s_k^{(s_i)}) > 0$ **do**
      $D_k^{(s_i)} = D_{k-1}^{(s_i)}$
      if $s_k^{(s_i)} \neq \emptyset$ **then**
        $D_k^{(s_i)}(s_k^{(s_i)})$ += 1 // Apply action $s_k^{(s_i)}$
      end if
      if $P_k[D_k^{(s_i)}]$ does not exist yet **then**
        $P_k[D_k^{(s_i)}] = 0.0$
      end if
      $P_k[D_k^{(s_i)}]$ += $P_{k-1}[D_{k-1}^{(s_i)}] \times \sigma_k^{(s_i)}(s_k^{(s_i)})$
    end for
  end for
end for
return P_n

2.3.3 Proof of correctness

It is straightforward to see that Algorithm 1 is computing the following recurrence in iteration k:

$$\forall D_k \in \Delta_k^{(s_i)}, \quad P_k[D_k] = \sum_{D_{k-1}, s_k^{(s_i)} : \mathcal{D}^{(s_i)}(D_{k-1}, s_k^{(s_i)}) = D_k} P_{k-1}[D_{k-1}] \times \sigma_k^{(s_i)}(s_k^{(s_i)}) \tag{2.8}$$

where $\mathcal{D}^{(s_i)}(D_{k-1}, s_k^{(s_i)})$ denotes the configuration resulting from applying k's projected action $s_k^{(s_i)}$ to the configuration $D_{k-1} \in \Delta_k^{(s_i)}$.

On the other hand, the probability distribution on $\Delta_k^{(s_i)}$ induced by $\sigma_{1\ldots k}$ is by definition

$$\Pr(D_k|\sigma_{1\ldots k}) = \sum_{s_{1\ldots k} : \mathcal{D}^{(s_i)}(s_{1\ldots k}) = D_k} \prod_{j=1}^{k} \sigma_j(s_j) \tag{2.9}$$

Now we want to prove that our DP algorithm is indeed computing the correct probability distribution, i.e. $P_k[D_k]$ as defined by Equation 2.8 is equal to $\Pr(D_k|\sigma_{1\ldots k})$.

Theorem 1. *For all k, and for all $D_k \in \Delta_k^{(s_i)}$, $P_k[D_k] = \Pr(D_k|\sigma_{1\ldots k})$.*

Proof by induction on k. **Base case**: Applying Equation (2.8) for $k = 1$, it is straightforward to verify that $P_1[D_1] = \Pr(D_1|\sigma_1)$ for all $D_1 \in \Delta_1^{(s_i)}$.

Inductive case: Now assume $P_{k-1}[D_{k-1}] = \Pr(D_{k-1}|\sigma_{1\ldots k-1})$ for all $D_{k-1} \in \Delta_{k-1}^{(s_i)}$.

$$P_k[D_k] = \sum_{\substack{D_{k-1}, s_k : \\ \mathcal{D}(D_{k-1}, s_k) = D_k}} P_{k-1}[D_{k-1}] \times \sigma_k(s_k) \tag{2.10}$$

$$= \sum_{\substack{D_{k-1}, s_k : \\ \mathcal{D}(D_{k-1}, s_k) = D_k}} \sigma_k(s_k) \times \left(\sum_{s_{1\ldots k-1} : \mathcal{D}(s_{1\ldots k-1}) = D_{k-1}} \prod_{j=1}^{k-1} \sigma_j(s_j) \right) \tag{2.11}$$

$$= \sum_{D_{k-1}, s_k : \mathcal{D}(D_{k-1}, s_k) = D_k} \left(\sum_{s_{1\ldots k-1} : \mathcal{D}(s_{1\ldots k-1}) = D_{k-1}} \prod_{j=1}^{k} \sigma_j(s_j) \right) \tag{2.12}$$

$$= \sum_{s_{1\ldots k-1}} \sum_{s_k} \sum_{D_{k-1}} \mathbf{1}_{[\mathcal{D}(D_{k-1}, s_k) = D_k]} \cdot \mathbf{1}_{[\mathcal{D}(s_{1\ldots k-1}) = D_{k-1}]} \cdot \prod_{j=1}^{k} \sigma_j(s_j) \tag{2.13}$$

$$= \sum_{s_{1\ldots k}} \left(\sum_{D_{k-1}} \mathbf{1}_{[\mathcal{D}(D_{k-1}, s_k) = D_k]} \cdot \mathbf{1}_{[\mathcal{D}(s_{1\ldots k-1}) = D_{k-1}]} \right) \cdot \prod_{j=1}^{k} \sigma_j(s_j) \tag{2.14}$$

$$= \sum_{s_{1\ldots k}} \mathbf{1}_{[\mathcal{D}(s_{1\ldots k}) = D_k]} \prod_{j=1}^{k} \sigma_j(s_j) \tag{2.15}$$

$$= \sum_{s_{1\ldots k} : \mathcal{D}(s_{1\ldots k}) = D_k} \prod_{j=1}^{k} \sigma_j(s_j) \tag{2.16}$$

$$= \Pr(D_k|\sigma_{1\ldots k}) \tag{2.17}$$

Note that from (2.13) to (2.14) we use the fact that given an action profile $s_{1\ldots k-1}$, there is a unique configuration $D_{k-1} \in \Delta_{k-1}^{(s_i)}$ such that $D_{k-1} = \mathcal{D}^{(s_i)}(s_{1\ldots k-1})$. □

2.3.4 Complexity

Our algorithm for computing $V_{s_i}^i(\sigma_{-i})$ consists of first computing the projected strategies using (2.5), then following Algorithm 1, and finally doing the weighted sum given in Equation (2.6). Let $\Delta^{(s_i,i)}(\sigma_{-i})$ denote the set of configurations over $\nu(s_i)$ that have positive probability of occurring under the mixed strategy

(s_i, σ_{-i}). In other words, this is the number of terms we need to add together when doing the weighted sum in Equation (2.6). When σ_{-i} has full support, $\Delta^{(s_i,i)}(\sigma_{-i}) = \Delta^{(s_i,i)}$. Since looking up an entry in a trie takes time linear in the size of the key, which is $|\nu(s_i)|$ in our case, the complexity of doing the weighted sum in Equation (2.6) is $O(|\nu(s_i)||\Delta^{(s_i,i)}(\sigma_{-i})|)$.

Algorithm 1 requires n iterations; in iteration k, we look at all possible combinations of $D_{k-1}^{(s_i)}$ and $s_k^{(s_i)}$, and in each case do a trie look-up which costs $O(|\nu(s_i)|)$. Since $|S_k^{(s_i)}| \leq |\nu(s_i)| + 1$, and $|\Delta_{k-1}^{(s_i)}| \leq |\Delta^{(s_i,i)}|$, the complexity of Algorithm 1 is $O(n|\nu(s_i)|^2|\Delta^{(s_i,i)}(\sigma_{-i})|)$. This dominates the complexity of summing up (2.6). Adding the cost of computing $\sigma_{-i}^{(s)}$, we get the overall complexity of expected payoff computation $O(n|S| + n|\nu(s_i)|^2|\Delta^{(s_i,i)}(\sigma_{-i})|)$.

Since $|\Delta^{(s_i,i)}(\sigma_{-i})|) \leq |\Delta^{(s_i,i)}| \leq |\Delta^{(s_i)}|$, and $|\Delta^{(s_i)}|$ is the number of payoff values stored in payoff function u^{s_i}, this means that expected payoffs can be computed in polynomial time with respect to the size of the AGG. Furthermore, our algorithm is able to exploit strategies with small supports which lead to a small $|\Delta^{(s_i,i)}(\sigma_{-i})|)$. Since $|\Delta^{(s_i)}|$ is bounded by $\frac{(n-1+|\nu(s_i)|)!}{(n-1)!|\nu(s_i)|!}$, this implies that if the in-degree of the graph is bounded by a constant, then the complexity of computing expected payoffs is $O(n|S| + n^{\mathcal{I}+1})$.

Theorem 2. *Given an AGG representation of a game, i's expected payoff $V_{s_i}^i(\sigma_{-i})$ can be computed in polynomial time with respect to the representation size, and if the in-degree of the action graph is bounded by a constant, the complexity is polynomial in n.*

2.3.5 Discussion

Of course it is not necessary to apply the agents' mixed strategies in the order $1 \ldots n$. In fact, we can apply the strategies in any order. Although the number of configurations $|\Delta^{(s_i,i)}(\sigma_{-i})|$ remains the same, the ordering does affect the intermediate configurations $\Delta_k^{(s_i)}$. We can use the following heuristic to try to minimize the number of intermediate configurations: sort the players by the sizes of their projected action sets, in ascending order. This would reduce the amount of work we do in earlier iterations of Algorithm 1, but does not change the overall complexity of our algorithm.

In fact, we do not even have to apply *one* agent's strategy at a time. We could partition the set of players into sub-groups, compute the distributions induced by each of these sub-groups, then combine these distributions together. Algorithm 1 can be straightforwardly extended to deal with such distributions instead of mixed strategies of single agents. In Section 2.5.1 we apply this approach to compute Jacobians efficiently.

Relation to Polynomial Multiplication

We observe that the problem of computing $Pr(D|\sigma^{(s_i)})$ can be expressed as one of multiplication of multivariate polynomials. For each action node $s \in \nu(s_i)$,

let x_s be a variable corresponding to s. Then consider the following expression:

$$\prod_{k=1}^{n} \left(\sigma_k^{(s_i)}(\emptyset) + \sum_{s_k \in S_k \cap \nu(s_i)} \sigma_k(s_k) x_{s_k} \right) \tag{2.18}$$

This is a multiplication of n multivariate polynomials, each corresponding to one player's projected mixed strategy. This expression expands to a sum of $|\Delta^{(s_i,i)}|$ terms. Each term can be identified by the tuple of exponents of the x variables, $(D(s), D(s'), \ldots)$. In other words, the set of terms corresponds to the set of configurations $\Delta^{(s_i,i)}$. The coefficient of the term with exponents $D \in \Delta^{(s_i,i)}$ is

$$\sum_{\mathbf{s}^{(s_i)}:\mathcal{D}^{(s_i)}(\mathbf{s}^{(s_i)})=D} \left(\prod_{k=1}^{n} \sigma^{(s_i)}(s_k^{(s_i)}) \right)$$

which is exactly $\Pr(D|\sigma^{(s_i)})$ by Equation (2.7)! So the whole expression (2.18) evaluates to

$$\sum_{D \in \Delta^{(s_i,i)}} \Pr(D|\sigma^{(s_i)}) \prod_{s \in \nu(s_i)} x_s^{D(s)}$$

Thus the problem of computing $\Pr(D|\sigma^{(s_i)})$ is equivalent to the problem of computing the coefficients in (2.18). Our DP algorithm corresponds to the strategy of multiplying one polynomial at a time, i.e. at iteration k we multiply the polynomial corresponding to player k's strategy with the expanded polynomial of $1 \ldots (k-1)$ that we computed in the previous iteration.

2.4 AGG with Function Nodes

There are games with certain kinds of context-specific independence structures that AGGs are not able to exploit. In Example 4 we show a class of games with one such kind of structure. Our solution is to extend the AGG representation by introducing function nodes, which allows us to exploit a much wider variety of structures.

2.4.1 Motivating Example: Coffee Shop

Example 4. *In the Coffee Shop Game there are n players; each player is planning to open a new coffee shop in an downtown area, but has to decide on the location. The downtown area is represented by a $r \times c$ grid. Each player can choose to open the shop at any of the $B \equiv rc$ blocks, or decide not to enter the market. Conditioned on player i choosing some location s, her utility depends on:*

- *the number of players that chose the same block,*
- *the number of players that chose any of the surrounding blocks, and*

- *the number of players that chose any other location.*

The normal form representation of this game has size $n|S|^n = n(B+1)^n$. Since there are no strict independencies in the utility function, the size of the graphical game representation would be similar. Let us now represent the game as an AGG. We observe that if agent i chooses an action s corresponding to one of the B locations, then her payoff is affected by the configuration over all B locations. Hence, $\nu(s)$ would consist of B action nodes corresponding to the B locations. The action graph has in-degree $\mathcal{I} = B$. Since the action sets completely overlap, the representation size is $O(|S||\Delta^{(s)}|) = O(B\frac{(n-1+B)!}{(n-1)!B!})$. If we hold B constant, this becomes $O(Bn^B)$, which is exponentially more compact than the normal form and the graphical game representation. If we instead hold n constant, the size of the representation is $O(B^n)$, which is only slightly better than the normal form and graphical game representations.

Intuitively, the AGG representation is only able to exploit the anonymity structure in this game. However, this game's payoff function does have context-specific structure. Observe that u^s depends only on three quantities: the number of players that chose the same block, the number of players who chose surrounding blocks, and the number of players who chose other locations. In other words, u^s can be written as a function g of only 3 integers: $u^s(D^{(s)}) = g(D(s), \sum_{s' \in S'} D(s'), \sum_{s'' \in S''} D(s''))$ where S' is the set of actions that surrounds s and S'' the set of actions corresponding to the other locations. Because the AGG representation is not able to exploit this context-specific information, utility values are duplicated in the representation.

2.4.2 Function Nodes

In the above example we showed a kind of context-specific independence structure that AGGs cannot exploit. It is easy to think of similar examples, where u^s could be written as a function of a small number of intermediate parameters. One example is a "parity game" where u^s depends only on whether $\sum_{s' \in \nu(s)} D(s')$ is even or odd. Thus u^s would have just two distinct values, but the AGG representation would have to specify a value for every configuration $D^{(s)}$.

This kind of structure can be exploited within the AGG framework by introducing *function nodes* to the action graph G. Now G's vertices consist of both the set of action nodes S and the set of function nodes P. We require that no function node $p \in P$ can be in any player's action set, i.e. $S \cap P = \{\}$, so the total number of nodes in G is $|S| + |P|$. Each node in G can have action nodes and/or function nodes as neighbors. For each $p \in P$, we introduce a function $f_p : \Delta^{(p)} \mapsto \mathbb{N}$, where $D^{(p)} \in \Delta^{(p)}$ denotes configurations over p's neighbors. The configurations D are extended over the entire set of nodes, by defining $D(p) \equiv f_p(D^{(p)})$. Intuitively, $D(p)$ are the intermediate parameters that players' utilities depend on.

To ensure that the AGG is meaningful, the graph G restricted to nodes in P is required to be a directed acyclic graph (DAG). Furthermore it is required that

every $p \in P$ has at least one neighbor (i.e. incoming edge). These conditions ensure that $D(s)$ for all s and $D(p)$ for all p are well-defined. To ensure that every $p \in P$ is "useful", we also require that p has at least one out-going edge. As before, for each action node s we define a utility function $u^s : \Delta^{(s)} \mapsto \mathbb{R}$. We call this extended representation $(N, \mathbf{S}, P, \nu, \{f_p\}_{p\in P}, u)$ an Action Graph Game with Function Nodes (AGGFN).

2.4.3 Representation Size

Given an AGGFN, we can construct an equivalent AGG with the same players N and actions S and equivalent utility functions, but represented without any function nodes. We put an edge from s' to s in the AGG if either there is an edge from s' to s in the AGGFN, or there is a path from s' to s through a chain of function nodes. The number of utilities stored in an AGGFN is no greater than the number of utilities in the equivalent AGG without function nodes. We can show this by following similar arguments as before, establishing a many-to-one mapping from utilities in the AGG representation to utilities in the AGGFN. On the other hand, AGGFNs have to represent the functions f_p, which can either be implemented using elementary operations, or represented as mappings similar to u^s. We could construct examples with huge number of function nodes, such that the space complexity of representing $\{f_p\}_{p\in P}$ would be greater than that of the utility functions. In other words, blindly adding function nodes will not make the representation more compact. We want to add function nodes only when they represent meaningful intermediate parameters and hence reduce the number of incoming edges on action nodes.

Consider our coffee shop example. For each action node s corresponding to a location, we introduce function nodes p'_s and p''_s. Let $\nu(p'_s)$ consist of actions surrounding s, and $\nu(p''_s)$ consist of actions for the other locations. Then we modify $\nu(s)$ so that it has 3 nodes: $\nu(s) = \{s, p'_s, p''_s\}$, as shown in Figure 2.5. For all function nodes $p \in P$, we define $f_p(D^{(p)}) = \sum_{m\in\nu(p)} D(m)$. Now each $D^{(s)}$ is a configuration over only 3 nodes. Since f_p is a summation operator, $|\Delta^{(s)}|$ is the number of compositions of $n-1$ into 4 nonnegative integers, $\frac{(n+2)!}{(n-1)!3!} = n(n+1)(n+2)/6 = O(n^3)$. We must therefore store $O(Bn^3)$ utility values. This is significantly more compact than the AGG representation without function nodes, which had a representation size of $O(B\frac{(n-1+B)!}{(n-1)!B!})$.

Remark 1. One property of the AGG representation as defined in Section 2.2.1 is that utility function u^s is shared by all players that have s in their action sets. What if we want to represent games with agent-specific utility functions, where utilities depend not only on s and $D^{(s)}$, but also on the identity of the player playing s? We could split s into individual player's actions s_i, s_j etc., so that each action node has its own utility function, however the resulting AGG would not be able to take advantage of the fact that the actions s_i, s_j affect the other players' utilities in the same way. Using function nodes, we are able to compactly represent this kind of structure. We again split s into separate action nodes s_i, s_j, but also introduce a function node p with s_i, s_j as its neighbors,

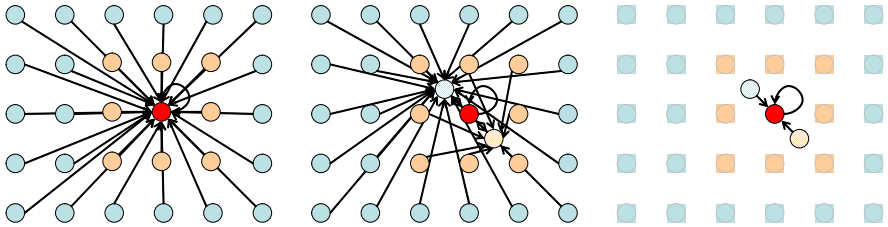

Figure 2.5: A 5×6 Coffee Shop Game: Left: the AGG representation without function nodes (looking at only the neighborhood of the a node s). Middle: we introduce two function nodes. Right: s now has only 3 incoming edges.

and define f_p to be the summation operator $f_p(D^{(p)}) = \sum_{m \in \nu p} D(m)$. This way the function node p with its configuration $D(p)$ acts as if s_i and s_j had been merged into one node. Action nodes could then include p instead of both s_i and s_j as a neighbor. This way agents can have different utility functions, without sacrificing representational compactness.

2.4.4 Computing with AGGFNs

Our expected-payoff algorithm cannot be directly applied to AGGFNs with arbitrary f_p. First of all, projection of strategies does not work directly, because a player j playing an action $s_j \notin \nu(s)$ could still affect $D^{(s)}$ via function nodes. Furthermore, our DP algorithm for computing the probabilities does not work because for an arbitrary function node $p \in \nu(s)$, each player would not be guaranteed to affect $D(p)$ independently. Therefore in the worst case we need to convert the AGGFN to an AGG without function nodes in order to apply our algorithm. This means that we are not always able to translate the extra compactness of AGGFNs over AGGs into more efficient computation.

Definition 1. *An AGGFN is* contribution-independent (CI) *if*

- *For all $p \in P$, $\nu(p) \subseteq S$, i.e. the neighbors of function nodes are action nodes.*
- *There exists a commutative and associative operator $*$, and for each node $s \in S$ an integer w_s, such that given an action profile $\mathbf{s}$, for all $p \in P$, $D(p) = *_{i \in N : s_i \in \nu(p)} w_{s_i}$.*

Note that this definition entails that $D(p)$ can be written as a function of $D^{(p)}$ by collecting terms: $D(p) \equiv f_p(D^{(p)}) = *_{s \in \nu(p)}(*_{k=1}^{D(s)} w_s)$.

The coffee shop game is an example of a contribution-independent AGGFN, with the summation operator serving as $*$, and $w_s = 1$ for all s. For the parity game mentioned earlier, $*$ is instead addition mod 2. If we are modeling an auction, and want $D(p)$ to represent the amount of the winning bid, we would let w_s be the bid amount corresponding to action s, and $*$ be the max operator.

For contribution-independent AGGFNs, it is the case that for all function nodes p, each player's strategy affects $D(p)$ independently. This fact allows us to adapt our algorithm to efficiently compute the expected payoff $V_{s_i}^i(\sigma_{-i})$. For simplicity we present the algorithm for the case where we have one operator $*$ for all $p \in P$, but our approach can be directly applied to games with different operators associated with different function nodes, and likewise with a different set of w_s for each operator.

We define the *contribution* of action s to node $m \in S \cup P$, denoted $C_s(m)$, as 1 if $m = s$, 0 if $m \in S \setminus \{s\}$, and $*_{m' \in \nu(m)}(*_{k=1}^{C_s(m')} w_s)$ if $m \in P$. Then it is easy to verify that given an action profile $\mathbf{s}$, $D(s) = \sum_{j=1}^{n} C_{s_j}(s)$ for all $s \in S$ and $D(p) = *_{j=1}^{n} C_{s_j}(p)$ for all $p \in P$.

Given that player i played s_i, we define the projected contribution of action s, denoted $C_s^{(s_i)}$, as the tuple $(C_s(m))_{m \in \nu(s_i)}$. Note that different actions may have identical projected contributions. Player j's mixed strategy σ_j induces a probability distribution over j's projected contributions, $\Pr(C^{(s_i)}|\sigma_j) = \sum_{s_j : C_{s_j}^{(s_i)} = C^{(s_i)}} \sigma_j(s_j)$. Now we can operate entirely using the probabilities on projected contributions instead of the mixed strategy probabilities. This is analogous to the projection of σ_j to $\sigma_j^{(s_i)}$ in our algorithm for AGGs without function nodes.

Algorithm 1 for computing the distribution $Pr(D^{(s_i)}|\sigma)$ can be straightforwardly adopted to work with contribution-independent AGGFNs: whenever we apply player k's contribution $C_{s_k}^{(s_i)}$ to $D_{k-1}^{(s_i)}$, the resulting configuration $D_k^{(s_i)}$ is computed componentwise as follows: $D_k^{(s_i)}(m) = C_{s_k}^{(s_i)}(m) + D_{k-1}^{(s_i)}(m)$ if $m \in S$, and $D_k^{(s_i)}(m) = C_{s_k}^{(s_i)}(m) * D_{k-1}^{(s_i)}(m)$ if $m \in P$. Following similar complexity analysis, if an AGGFN is contribution-independent, expected payoffs can be computed in polynomial time with respect to the representation size. Applied to the coffee shop example, since $|\Delta^{(s)}| = O(n^3)$, our algorithm takes $O(n|S| + n^4)$ time, which grows *linearly* in $|S|$.

Remark 2. We note that similar ideas are emloyed in the variable elimination algorithms that exploit *causal independence* in Bayes nets [Zhang and Poole 1996]. Bayes nets are compact representations of probability distributions that graphically represent independencies between random variables. A Bayes net is a DAG where nodes represent random variables and edges represent direct probabilistic dependence between random variables. Efficient algorithms have been developed to compute conditional probabilities in Bayes nets, such as clique tree propagation and variable elimination. Causal independence refers to the situation where a node's parents (which may represent causes) affect the node independently. The conditional probabilities of the node can be defined using a binary operator that can be applied to values from each of the parent variables. Zhang and Poole [1996] proposed a variable elimination algorithm that exploits causal independence by *factoring* the conditional probability distribution into factors corresponding to the causes. The way factors are combined together is similar in spirit to our DP algorithm that combines the independent contributions of the players' strategies to the configuration $D^{(s_i)}$.

This parallel between Bayes nets and action graphs are not surprising. In AGGFNs, we are trying to compute the probability distribution over configurations $\Pr(D^{(s_i)}|\sigma^{(s_i)})$. If we see each node m in the action graph as a random variable $D(m)$, this is the joint distribution of variables $\nu(s_i)$. However, whereas edges in Bayes nets represent probabilistic dependence, edges in the action graph have different semantics depending on the target. Incoming edges of action nodes specifies the neighborhood $\nu(s)$ that we are interested in computing the probabilities of. Incoming edges of a function node represents the deterministic dependence between the random variable of the function node $D(p)$ and its parents. The only probabilistic components of action graphs are the players' mixed strategies. These are probability distributions of random variables associated with players, but are not explicitly represented in the action graph. Whereas AGGFNs in general are not DAGs, given an action s, we can construct an induced Bayes net consisting of $\nu(s)$, the neighbors of function nodes in $\nu(s)$, and the neighbors of any new function nodes included, and so on until no more function nodes are included, and finally augmented with n nodes representing the players' mixed strategies. Whereas for CI AGGFNs, the Bayes net formulation has a simple structure and does not yield a more efficient algorithm compared to Algorithm 1, this formulation could be useful for non-CI AGGFNs with a complex network of function nodes, as standard Bayes net algorithms can be used to exploit the independencies in the induced Bayes net.

2.5 Applications

2.5.1 Application: Computing Payoff Jacobian

A game's payoff Jacobian under a mixed strategy σ is defined as a $\sum_i |S_i|$ by $\sum_i |S_i|$ matrix with entries defined as follows:

$$\frac{\partial V^i_{s_i}(\sigma_{-i})}{\partial \sigma_{i'}(s_{i'})} \equiv \nabla V^{i,i'}_{s_i,s_{i'}}(\overline{\sigma}) \tag{2.19}$$

$$= \sum_{\overline{\mathbf{s}} \in \overline{\mathbf{S}}} u\left(s_i, \mathcal{D}(s_i, s_{i'}, \overline{\mathbf{s}})\right) Pr(\overline{\mathbf{s}}|\overline{\sigma}) \tag{2.20}$$

Here whenever we use an overbar in our notation, it is shorthand for the subscript $-\{i, i'\}$. For example, $\overline{\mathbf{s}} \equiv \mathbf{s}_{-\{i,i'\}}$. The rows of the matrix are indexed by i and s_i while the columns are indexed by i' and $s_{i'}$. Given entry $\nabla V^{i,i'}_{s_i,s_{i'}}(\overline{\sigma})$, we call s_i its *primary action node*, and $s_{i'}$ its *secondary action node*.

One of the main reasons we are interested in computing Jacobians is that it is the computational bottleneck in Govindan and Wilson's continuation method for finding mixed-strategy Nash equilibria in multi-player games [Govindan and Wilson 2003]. The Govindan-Wilson algorithm starts by perturbing the payoffs to obtain a game with a known equilibrium. It then follows a path that is guaranteed to give us one or more equilibria of the unperturbed game. In each step, we need to compute the payoff Jacobian under the current mixed strategy

in order to get the direction of the path; we then take a small step along the path and repeat.

Efficient computation of the payoff Jacobian is important for more than this continuation method. For example, the iterated polymatrix approximation (IPA) method [Govindan and Wilson 2004] has the same computational problem at its core. At each step the IPA method constructs a polymatrix game that is a linearization of the current game with respect to the mixed strategy profile, the Lemke-Howson algorithm is used to solve this game, and the result updates the mixed strategy profile used in the next iteration. Though theoretically it offers no convergence guarantee, IPA is typically much faster than the continuation method. Also, it is often used to give the continuation method a quick start. The payoff Jacobian may also be useful to multiagent reinforcement learning algorithms that perform policy search.

Equation (2.20) shows that the $\nabla V^{i,i'}_{s_i,s_{i'}}(\overline{\sigma})$ element of the Jacobian can be interpreted as the expected utility of agent i when she takes action s_i, agent i' takes action $s_{i'}$, and all other agents use mixed strategies according to $\overline{\sigma}$. So a straightforward approach is to use our DP algorithm to compute each entry of the Jacobian. However, the Jacobian matrix has certain extra structure that allows us to achieve further speedup.

First, we observe that some entries of the Jacobian are identical. If two entries have same primary action node s, then they are expected payoffs on the same utility function u^s, i.e. they have the same value if their induced probability distributions over $\Delta^{(s)}$ are the same. We need to consider two cases:

1. Suppose the two entries come from the same row of the Jacobian, say player i's action s_i. There are two sub-cases to consider:

   (a) Suppose the columns of the two entries belong to the same player j, but different actions s_j and s'_j. If $s_j^{(s_i)} = {s'}_j^{(s_i)}$, i.e. s_j and s'_j both project to the same projected action in s_i's projected action graph, then $\nabla V^{i,j}_{s_i,s_j} = \nabla V^{i,j}_{s_i,s'_j}$.

   (b) Suppose the columns of the entries correspond to actions of different players. We observe that for all j and s_j such that $\sigma^{(s_i)}(s_j^{(s_i)}) = 1$, $\nabla V^{i,j}_{s_i,s_j}(\overline{\sigma}) = V^i_{s_i}(\sigma_{-i})$. As a special case, if $S_j^{(s_i)} = \{\emptyset\}$, i.e. agent j does not affect i's payoff when i plays s_i, then for all $s_j \in S_j$, $\nabla V^{i,j}_{s_i,s_j}(\overline{\sigma}) = V^i_{s_i}(\sigma_{-i})$.

2. If s_i and s_j correspond to the same action node s (but owned by agents i and j respectively), thus sharing the same payoff function u^s, then $\nabla V^{i,j}_{s_i,s_j} = \nabla V^{j,i}_{s_j,s_i}$. Furthermore, if there exist $s'_i \in S_i, s'_j \in S_j$ such that ${s'_i}^{(s)} = {s'_j}^{(s)}$, then $\nabla V^{i,j}_{s_i,s'_j} = \nabla V^{j,i}_{s_j,s'_i}$.

Even if the entries are not equal, we can exploit the similarity of the projected strategy profiles (and thus the similarity of the induced distributions) between entries, and re-use intermediate results when computing the induced

distributions of different entries. Since computing the induced probability distributions is the bottleneck of our expected payoff algorithm, this provides significant speedup.

First we observe that if we fix the row (i, s_i) and the column's player j, then $\overline{\sigma}$ is the same for all secondary actions $s_j \in S_j$. We can compute the probability distribution $\Pr(D_{n-1}|s_i, \overline{\sigma}^{(s_i)})$, then for all $s_j \in S_j$, we just need to apply the action s_j to get the induced probability distribution for the entry $\nabla V^{i,j}_{s_i,s_j}$.

Now suppose we fix the row (i, s_i). For two column players j and j', their corresponding strategy profiles $\sigma_{-\{i,j\}}$ and $\sigma_{-\{i,j'\}}$ are very similar, in fact they are identical in $n-3$ of the $n-2$ components. For AGGs without function nodes, we can exploit this similarity by computing the distribution $\Pr(D_{n-1}|\sigma^{(s_i)}_{-i})$, then for each $j \neq i$, we "undo" j's mixed strategy to get the distribution induced by $\sigma_{-\{i,j\}}$. Recall from Section 2.3.5 that the distributions are coefficients of the multiplication of certain polynomials. So we can undo j's strategy by computing the long division of the polynomial for σ_{-i} by the polynomial for σ_j.

This method does not work for contribution-independent AGGFNs, because in general a player's contribution to the configurations are not reversible, i.e. given $Pr(D_{n-1}|\sigma^{(s_i)}_{-i})$ and σ_j, it is not always possible to undo the contributions of σ_j. Instead, we can efficiently compute the distributions by recursively bisecting the set of players in to sub-groups, computing probability distributions induced by the strategies of these sub-groups and combining them. For example, suppose $n = 9$ and $i = 9$, so $\sigma_{-i} = \sigma_{1\ldots8}$. We need to compute the distributions induced by $\sigma_{-\{1,9\}}, \ldots, \sigma_{-\{8,9\}}$, respectively. Now we bisect σ_{-i} into $\sigma_{1\ldots4}$ and $\sigma_{5\ldots8}$. Suppose we have computed the distributions induced by $\sigma_{1\ldots4}$ as well as $\sigma_{234}, \sigma_{134}, \sigma_{124}, \sigma_{123}$, and similarly for the other group of $5\ldots8$. Then we can compute $\Pr(\cdot|\sigma^{(s_i)}_{-\{1,9\}})$ by combining $\Pr(\cdot|\sigma^{(s_i)}_{234})$ and $\Pr(\cdot|\sigma^{(s_i)}_{5678})$, compute $\Pr(\cdot|\sigma^{(s_i)}_{-\{2,9\}})$ by combining $\Pr(\cdot|\sigma^{(s_i)}_{134})$ and $\Pr(\cdot|\sigma^{(s_i)}_{5678})$, etc. We have reduced the problem into two smaller problems over the sub-groups $1\ldots4$ and $5\ldots8$, which can then be solved recursively by further bisecting the sub-groups. This method saves the re-computation of sub-groups of strategies when computing the induced distributions for each row of the Jacobian, and it works with any contribution-independent AGGFNs because it does not use long division to undo strategies.

2.6 Experiments

We implemented the AGG representation and our algorithm for computing expected payoffs and payoff Jacobians in C++. We ran several experiments to compare the performance of our implementation against the (heavily optimized) GameTracer implementation [Blum et al. 2002] which performs the same computation for a normal form representation. We used the Coffee Shop game (with randomly-chosen payoff values) as a benchmark. We varied both the number of players and the number of actions.

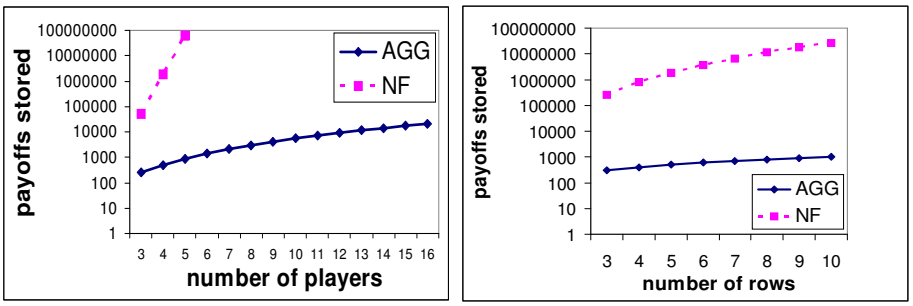

Figure 2.6: Comparing Representation Sizes of the Coffee Shop Game (log-scale). Left: 5×5 grid with 3 to 16 players. Right: 4-player $r \times 5$ grid with r varying from 3 to 10.

2.6.1 Representation Size

For each game instance we counted the number of payoff values that need to be stored in each representation. Since for both normal form and AGG, the size of the representation is dominated by the number of payoff values stored, the number of payoff values is a good indication of the size of the representation.

We first looked at Coffee Shop games with 5×5 blocks, with varying number of players. Figure 2.6 has a log-scale plot of the number of payoff values in each representation versus the number of players. The normal form representation grew exponentially with respect to the number of players, and quickly becomes impractical for large number of players. The size of the AGG representation grew polynomially with respect to n.

We then fixed the number of players at 4, and varied the number of blocks. For ease of comparison we fixed the number of colums at 5, and only changed the number of rows. Figure 2.6 has a log-scale plot of the number of payoff values versus the number of rows. The size of the AGG representation grew linearly with the number of rows, whereas the size of the normal form representation grew like a higher-order polynomial. This was consistent with our theoretical prediction that AGGFNs store $O(|S|n^3)$ payoff values for Coffee Shop games while normal form representations store $n|S|^n$ payoff values.

2.6.2 Expected Payoff Computation

Second, we tested the performance of our dynamic programming algorithm against GameTracer's normal form based algorithm for computing expected payoffs, on Coffee Shop games of different sizes. For each game instance, we generated 1000 random strategy profiles with full support, and measured the CPU (user) time spent computing the expected payoffs under these strategy profiles. We fixed the size of blocks at 5×5 and varied the number of players. Figure 2.7 shows plots of the results. For very small games the normal form based algorithm is faster due to its smaller bookkeeping overhead; as the num-

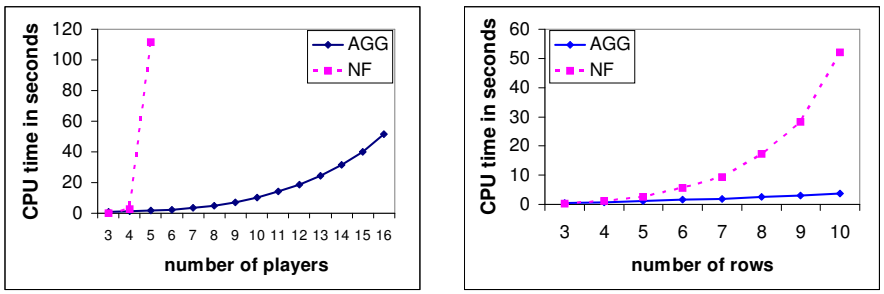

Figure 2.7: Running times for payoff computation in the Coffee Shop Game. Left: 5×5 grid with 3 to 16 players. Right: 4-player $r \times 5$ grid with r varying from 3 to 10.

ber of players grows larger, our AGGFN-based algorithm's running time grows polynomially, while the normal form based algorithm scales exponentially. For more than five players, we were not able to store the normal form representation in memory.

Next, we fixed the number of players at 4 and number of columns at 5, and varied the number of rows. Our algorithm's running time grew roughly linearly in the number of rows, while the normal form based algorithm grew like a higher-order polynomial. This was consistent with our theoretical prediction that our algorithm take $O(n|S|+n^4)$ time for this class of games while normal-form based algorithms take $O(|S|^{n-1})$ time.

Last, we considered strategy profiles having partial support. While ensuring that each player's support included at least one action, we generated strategy profiles with each action included in the support with probability 0.4. GameTracer took about 60% of its full-support running times to compute expected payoffs in this domain, while our algorithm required about 20% of its full-support running times.

2.6.3 Computing Payoff Jacobians

We have also run similar experiments on computing payoff Jacobians. As discussed in Section 2.5.1, the entries of a Jacobian can be formulated as expected payoffs, so a Jacobian can be computed by doing an expected payoff computation for each of its entry. In Section 2.5.1 we discussed methods that exploits the structure of the Jacobian to further speedup the computation. GameTracer's normal-form based implementation also exploits the structure of the Jacobian by re-using partial results of expected-payoff computations. When comparing our AGG-based Jacobian algorithm as described in Section 2.5.1 against GameTracer's implementation, the results are very similar to the above results for computing expected payoffs, i.e. our implementation scales polynomially in n while GameTracer scales exponentially in n. We instead focus on the question of how much speedup does the methods in Section 2.5.1 provide, by comparing

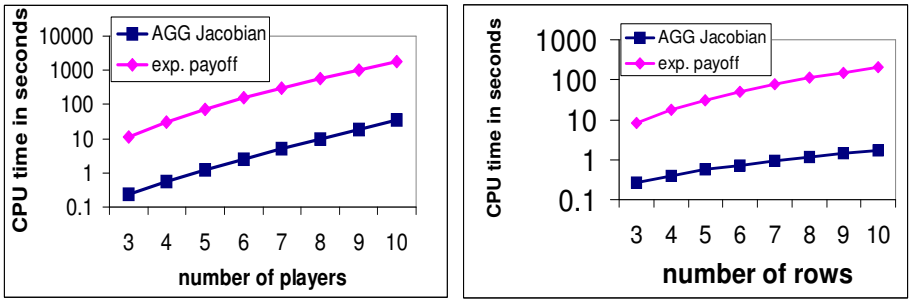

Figure 2.8: Running times for Jacobian computation in the Coffee Shop Game. Left: 5×5 grid with 3 to 10 players. Right: 4-player $r \times 5$ grid with r varying from 3 to 10.

our algorithm in Section 2.5.1 against the algorithm that computes expected payoffs (using our AGG-based algorithm described in Section 2.3) for each of the Jacobian's entries. The results are shown in Figure 2.8. Our algorithm is about 50 times faster. This confirms that the methods discussed in Seciton 2.5.1 provide significant speedup for computing payoff Jacobians.

2.6.4 Finding Nash Equilibria using the Govindan-Wilson algorithm

Govindan and Wilson's algorithm [Govindan and Wilson 2003] is one of the most competitive algorithms for finding Nash equilibria for multi-player games. The computational bottleneck of the algorithm is repeated computation of payoff Jacobians as defined in Section 2.5.1. Now we show experimentally that the speedup we achieved for computing Jacobians using the AGG representation leads to a speedup in the Govidan-Wilson algorithm.

We compared two versions of the Govindan-Wilson algorithm: one is the implementation in GameTracer, where the Jacobian computation is based on the normal form representation; the other is identical to the GameTracer implementation, except that the Jacobians are computed using our algorithm for the AGG representation. Both techniques compute the Jacobians exactly. As a result, given an initial perturbation to the original game, these two implementations would follow the same path and return exactly the same answers. So the difference in their running times would be due to the different speeds of computing Jacobians.

Again, we tested the two algorithms on Coffee Shop games of varying sizes: first we fixed the size of blocks at 4×4 and varied the number of players; then we fixed the number of players at 4 and number of columns at 4, and varied the number of rows. For each game instance, we randomly generated 10 initial perturbation vectors, and for each initial perturbation we run the two versions of the Govindan-Wilson algorithm. Since the running time of the Govindan-Wilson algorithm highly depends on the initial perturbation, it is not meaningful

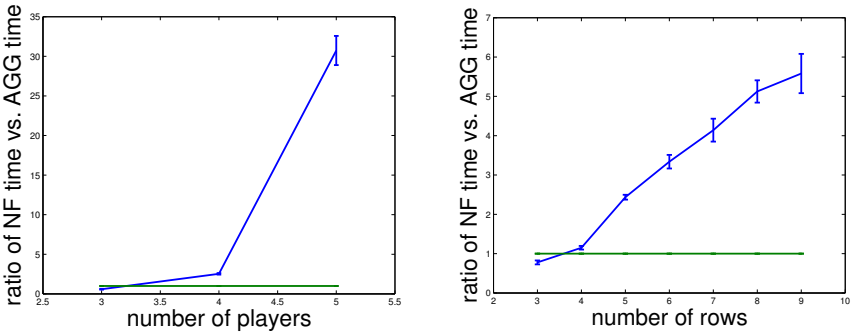

Figure 2.9: Ratios of Running times for the Govindan-Wilson algorithm in the Coffee Shop Game. Left: 4×4 grid with 3 to 5 players. Right: 4-player $r \times 4$ grid with r varying from 3 to 9. The error bars indicate standard deviation over 10 random initial perturbations. The constant lines at 1.0 indicating equal running times are also shown.

to compare the running times with different initial perturbations. Instead, we look at the *ratio* of running times between the normal form implementation and the AGG implementation. Thus a ratio greater than 1 means the AGG implementation spent less time than the normal form implementation. We plotted the results in Figure 2.9. The results confirmed our theoretical prediction that as the size of the games grows (either in the number of players or in the number of actions), the speedup of the AGG implementation compared to the normal from implementation increases.

2.7 Conclusions

We presented a polynomial-time algorithm for computing expected payoffs in action-graph games. For AGGs with bounded in-degree, our algorithm achieves an exponential speed-up compared to normal-form based algorithms and Bhat and Leyton-Brown [2004]'s algorithm. We also extended the AGG representation by introducing function nodes, which allows us to compactly represent a wider range of structured utility functions. We showed that if an AGGFN is contribution-independent, expected payoffs can be computed in polynomial time.

Our current and future research includes two directions: Computationally, we plan to apply our expected payoff algorithm to speed up other game-theoretic computations, such as computing best responses and the simplicial subdivision algorithm for finding Nash equilibria. Also, as a direct corollary of our Theorem 2 and Papadimitriou [2005]'s result, correlated equilibria can be computed in time polynomial in the size of the AGG.

Representationally, we plan to extend the AGG framework to represent more types of structure such as additivity of payoffs. In particular, we intend to study

is Bayesian games. In a Bayesian game, players are uncertain about which game (i.e. payoff function) they are playing, and each receives certain private information about the underlying game. Bayesian games are heavily used in economics for modeling competitive scenarios involving information asymmetries, e.g. for modeling auctions and other kinds of markets. A Bayesian game can be seen as a compact representation, since it is much more compact than its induced normal form. We plan to use the AGG framework to represent not only the structure inherent in Bayesian games, but also context-specific independence structures such as the ones we have considered here.

Chapter 3

Bidding Agents for Online Auctions with Hidden Bids

3.1 Introduction

There[1] has been much research on the study of automated bidding agents for auctions and other market-based environments. The Trading Agent Competitions (TAC-Classic and TAC Supply Chain Management) have attracted much interest [Wellman et al. 2002]. There have also been research efforts on bidding agents and bidding strategies in other auction environments [Byde 2002; Boutilier et al. 1999; Greenwald and Boyan 2004; Arora et al. 2003; Cai and Wurman 2003; Anthony et al. 2001]. Although this body of work considers many different auction environments, bidding agents always face a similar task: given a valuation function, the bidding agent needs to compute an optimal bidding strategy that maximizes expected surplus. (Some environments such as TAC-SCM also require agents to solve additional, e.g., scheduling tasks.)

The "Wilson Doctrine" in mechanism design argues that mechanisms should be constructed so that they are "detail-free"—that is, so that agents can behave rationally in these mechanisms even without information about the distribution of other agents' valuations. For example, the VCG mechanism is detail-free because under this mechanism it is a weakly dominant strategy to bid exactly one's valuation, regardless of other agents' beliefs, valuations or actions. Under common assumptions (in particular, independent private values) single-item English auctions are similar: an agent should remain in the auction until the bidding reaches the amount of her valuation.

While detail-free mechanisms are desirable, they are not ubiquitous. Very often, agents are faced with the problem of deciding how to behave in games that do not have dominant strategies and where other agents' preferences are strategically relevant. For example, we may want to participate in a series of auctions run by different sellers at different times. This is a common scenario at online auction sites such as eBay. In Section 3.4 we consider a sequential auction model of this scenario, and show that information about other bidders' preferences is very relevant in constructing a bidding strategy.

[1] A version of this chapter has been submitted for publication. Jiang, A.X. and Leyton-Brown, K. (2005) Bidding Agents for Online Auctions with Hidden Bids. Submitted to the Machine Learning Journal.

3.1.1 Game-Theoretic and Decision-Theoretic Approaches

How should a bidding agent be constructed? Depending on the assumptions we choose to make about other bidders, two broad approaches to computing bidding strategies suggest themselves: a game-theoretic approach and a decision-theoretic approach. The game theoretic approach assumes that all agents are perfectly rational and that this rationality is common knowledge; the auction is modeled as a Bayesian game. Under this approach, a bidding agent would compute a Bayes-Nash equilibrium of the auction game, and play the equilibrium bidding strategy. Much of the extensive economics literature on auctions follows this approach (see, e.g., the survey in [Klemperer 2000]). For example, in environments with multiple, sequential auctions for identical items and in which each bidder wants only a single item, the Bayes-Nash equilibrium is well-known [Milgrom and Weber 2000; Weber 1983]. Such equilibria very often depend on the distribution of agents' valuation functions and the number of bidders. Although this information is rarely available in practice, it *is* usually possible to estimate these distributions from the bidding history of previous auctions of similar items. Note that this involves making the assumption that past and future bidders will share the same valuation distribution.

The game-theoretic approach has received a great deal of study, and is perhaps the dominant paradigm in microeconomics. In particular, there are very good reasons for seeking strategy profiles that are resistant to unilateral deviation. However, this approach is not always useful to agents who need to decide what to do in a particular setting, especially when the rationality of other bidders is in doubt, when the computation of equilibria is intractable, or when the game has multiple equilibria. In such settings, it is sometimes more appropriate to rely on decision theory. A decision-theoretic approach effectively treats other bidders as part of the environment, and ignores the possibility that they may change their behavior in response to the agent's actions. We again make the assumption that the other bidders come from a stationary population; however, in this case we model agents' bid amounts directly rather than modeling their valuations and then seeking an equilibrium strategy. We then solve the resulting single-agent decision problem to find a bidding strategy that maximizes expected payoff. We could also use a reinforcement-learning approach, where we continue to learn the bidding behavior of other bidders while participating in the auctions. Much recent literature on bidding agent design follows the decision-theoretic approach, e.g. [Boutilier et al. 1999; Byde 2002; Greenwald and Boyan 2004; Stone et al. 2002; Mackie-Mason et al. 2004; Osepayshvili et al. 2005].

This chapter does not attempt to choose between these two approaches; it is our opinion that each has domains for which it is the most appropriate. The important point is that regardless of which approach we elect to take, we are faced with the subproblem of estimating two distributions from the bidding history of past auctions: the distribution on the number of bidders, and the distribution of bid amounts (for decision-theoretic approaches) or of valuations

(for game-theoretic approaches). Consequently, this problem has received much discussion in the literature. For example, Athey and Haile [2002] and various others in econometrics study the estimation of valuation distributions in various standard auction types given observed bids, assuming that bidders are perfectly rational and follow equilibrium strategies. On the decision-theoretic front, a popular approach is to estimate the distribution of the final prices in auctions based on observed selling prices, and then to use this distribution to compute the optimal bidding strategy. Examples of this include Stone et al.'s [2002] ATTac agent for the Trading Agent Competition, Mackie-Mason et al.'s [2004] study of bidding in simultaneous ascending auctions and the follow-up work in [Osepayshvili et al. 2005], Byde's [2002] study of bidding in simultaneous online English auctions, and Greenwald and Boyan's [2004] analysis of sequential English auctions. A paper by Boutilier et al. [1999] takes a different decision-theoretic approach which is relevant to the approach we propose in this chapter. We defer discussion of this work until Section 3.6, after we have presented our approach.

3.1.2 Overview

In this chapter we consider sequential English auctions in which a full bidding history is revealed, such as the online auctions run by eBay. It might seem that there is very little left to say: we learn the distributions of interest from the bidding history, then compute a bidding strategy based on that information for the current and future auctions. However, we show that under realistic bidding dynamics (described in Section 3.2) the observed bidding histories omit some relevant information. First, some bidders may come to the auction when it is already in progress, find that the current price already exceeds their valuation, and leave without placing a bid. Second, the amount the winner was willing to pay is never revealed. Ignoring these sources of bias can lead to poor estimates of the underlying valuation distribution and distribution of number of bidders. In Section 3.3 we propose a learning approach based on the Expectation Maximization (EM) algorithm, which iteratively generates hidden bids consistent with the observed bids, and then computes maximum-likelihood estimations of the valuation distribution based on the completed set of bids. The learned distributions can then be used in computing decision-theoretic or game-theoretic bidding strategies. Section 3.4 discusses the computation of the optimal strategy for an agent in a repeated English auction setting under the decision-theoretic approach, and in a similar (non-repeated) setting under the game-theoretic approach. In Section 3.5 we present experimental results on synthetic data sets as well as on data collected from eBay, which show that our EM learning approach makes better estimates of the distributions, gets more payoff under the decision-theoretic model, and makes a better approximation to the Bayes-Nash equilibrium under the game-theoretic model, as compared to the straightforward approach which ignores hidden bids.

3.2 A Model of Online Auctions

Consider a (possibly repeated) auction held online, e.g., on eBay. There are m potential bidders interested in a certain auction for a single item. We assume that bidders are risk-neutral, have independent private values (IPV), and that utilities are quasilinear. The number of bidders m is drawn from a discrete distribution $g(m)$ with support $[2, \infty)$. Bidders' potential valuations (in the game-theoretic context) or bids (in the decision-theoretic context) are independently drawn from a continuous distribution $f(x)$.

3.2.1 Bidding Dynamics

The m potential bidders arrive at the auction site sequentially. When each bidder arrives, she observes the bids that have been accepted in the auction so far, places a single proxy bid[2] and then leaves the system. The auctioneer processes new bids as follows:

1. When a proxy bid is submitted, the auctioneer compares it to the current price level, which is the second-highest proxy bid so far plus a small bid increment.[3]
   (a) If the submitted bid is not greater than the current price level the bid is dropped and no record of the bid is recorded in the auction's history.
   (b) If the submitted bid is higher than the current price level but lower than the highest proxy bid so far, then the submitted bid is accepted, the bid amount of the currently winning bid is increased to equal the submitted bid (i.e., this bid remains the winning bid), and the submitted bid loses.
   (c) If the submitted bid is higher than the previously winning bid then the price level is increased to equal the previously winning bid and the submitted bid is made the current winner.
2. At the end of the auction, the item is awarded to the bidder who placed the highest bid, and the final price level is the amount of the second highest bid.

3.2.2 Hidden Bids

According to our model, some bidders' proxy bid amounts will be revealed. This includes any bid that was higher than the current price level at the time it was placed, even if that bid was immediately outbid by another proxy bid. However, other bids will not be observed. There are two types of hidden bids:

[2] A proxy bidding system asks bidders for their maximum willingness to pay, and then bids up to this amount on the bidder's behalf. Such systems are common on online auction sites such as eBay; see e.g., `http://pages.ebay.com/help/buy/proxy-bidding.html`

[3] For simplicity in this chapter we ignore this small increment and assume that the current price level is the second-highest proxy bid so far.

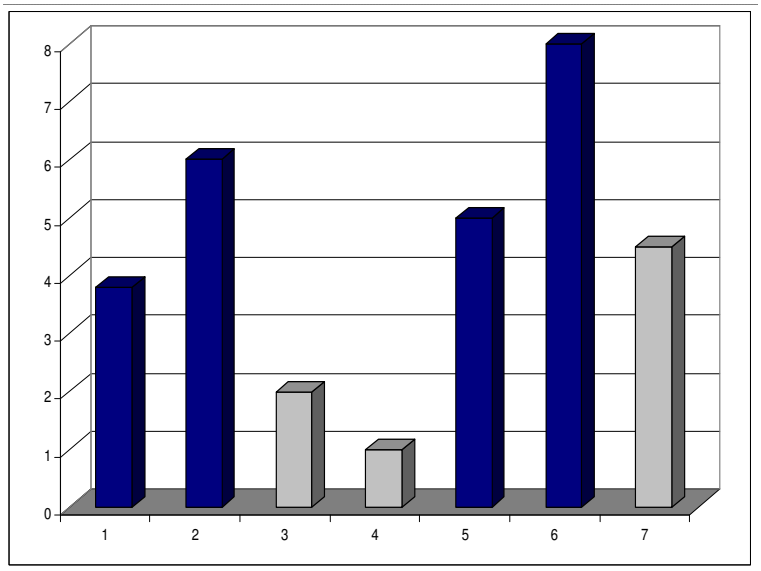

Figure 3.1: An example of the bidding process of an auction with 7 potential bidders.

1. The highest bid of each auction x_{hi}.
2. Dropped bids x_d that were lower than the current price when they were placed.

Let us denote the set of visible bids as x_v and the set of hidden bids as x_h. Let n denote the number of bidders that appears in the bidding history. This means that x_v will always contain $(n-1)$ bids, since the winning bidder's proxy bid is never observed. Since there are m potential bidders in total, $(n-1)$ bids are visible, and one bid is the highest bid x_{hi}, there are $(m-n)$ dropped bids in x_d.

Figure 3.1 shows an example of the bidding process for an auction according to our online auction model, and illustrates how bids are dropped. Bids arrive sequentially from left to right. Bids 3, 4 and 7 (the grey bars) will be dropped because they are lower than the current bid amount at their respective time steps. The amount of the winning bid (6) will not be revealed, although an observer would know that it was at least the amount of bid 2, which would be the amount paid by bidder 6.

3.2.3 Discussion

Our model of the bidding process is quite general. Notice that when a bidder observes that the price level is higher than her potential bid, she may decide not to bid in this auction. This is equivalent to our model in which she always submits

the bid, because dropped bids do not appear in the bidding history (Indeed, this is the motivation for our model). Also our model covers the case of last-minute bidding, which happens quite often in eBay auctions [Roth and Ockenfels 2002]: even though last-minute bids may be submitted almost simultaneously, eBay processes the bids in sequence.

Observe that with a proxy bidding system, and when agents have independent private values, bidders would not benefit from bidding more than once in an auction. However, in practice eBay bidders quite often make multiple bids in one auction. One possible motivation for these bids is a desire to learn more information about the proxy bid of the current high bidder [Shah et al. 2003]. However, only the last bid of the bidder represents her willingness to pay. Given a bidder's last bid, her earlier bids carry no extra information. Therefore, we will be interested in only the last bid from each bidder.[4] We can preprocess the bidding histories by removing all bids except the last bids from each bidder, without losing much information.

3.3 Learning the Distributions

Given the model of the bidding process, the first task of our bidding agent is to estimate the distributions $f(x)$ and $g(m)$ from the bidding history. Suppose we have access to the bidding history of K auctions for identical items.

3.3.1 The Simple Approach

A simple approach is to ignore the hidden bids, and to directly estimate $f(x)$ and $g(m)$ from observed data. The observed number of bidders, n, is used to estimate $g(m)$. To estimate $f(x)$ we use the observed bids x_v, which consists of $(n-1)$ bids for each auction. Any standard density estimation technique may be used.

Because it ignores hidden bids, this approach can be expected to produce biased estimates of f and g:

- $g(m)$ will be skewed towards small values because $n \leq m$.
- $f(v)$ may be skewed towards small values because it ignores the winning bid x_{hi}, or it may be skewed towards large values because it ignores the dropped bids x_d.

A popular variation of this approach (mentioned in Section 3.1) is to directly estimate the distribution of final prices from the selling prices of previous auctions, then to use this distribution to compute a decision-theoretic bidding strategy. The problem with this approach is that for English auctions, the selling prices are the second-highest bids. As we will show in Section 3.4, to compute a decision-theoretic strategy we really need the distribution of the other bidders'

[4] In a common value model, the earlier bids do carry some information, and we would not be able to simply ignore those bids.

highest bids. Using the distribution of final prices introduces bias, as this distribution is skewed towards small values compared to the distribution of highest bids.

3.3.2 EM Learning Approach

We would like to have an estimation strategy that accounts for the hidden bids x_h and any bias introduced by their absence. Suppose $f(x)$ belongs to a class of distributions parameterized by θ, $f(x|\theta)$. Further suppose that $g(m)$ belongs to a class of distributions parameterized by λ, $g(m|\lambda)$. For example, $f(x)$ could be a Normal distribution parameterized by its mean μ and variance σ^2, whereas $g(m)$ could be a Poisson distribution parameterized by its mean, λ. We want to find the maximum likelihood estimates of θ and λ given the observed data x_v.

Suppose that we could actually observe the hidden bids x_h in addition to x_v. Then estimating θ and λ from the completed data set (x_v, x_h) would be easy. Unfortunately we do not have x_h. Given x_v, and with the knowledge of the bidding process, we could generate x_h if we knew θ and λ. Unfortunately we do not know θ and λ.

A popular strategy for learning this kind of model with missing data is the Expectation Maximization (EM) algorithm [Dempster et al. 1977]. EM is an iterative procedure that alternates between E steps which generate the missing data given current estimates for the parameters and M steps which compute the maximum likelihood (or maximum a posteriori) estimates for the parameters based on the completed data, which consists of the observed data and current estimates for the missing data. Formally, the E step computes

$$Q(\theta) = \int \log(p(x_h, x_v|\theta))p(x_h|x_v, \theta^{(old)}, \lambda^{(old)})dx_h, \tag{3.1}$$

and the M step solves the optimization

$$\theta^{(new)} = \arg\max_{\theta}(Q(\theta)). \tag{3.2}$$

Analogous computations are done to estimate λ, the parameter for $g(m|\lambda)$. The EM algorithm terminates when λ and θ converge. It can be shown that EM will converge to a local maximum of the observed data likelihood function $p(x_v|\theta, \lambda)$.

EM is a particularly appropriate algorithm in our auction setting with hidden bids, because

1. EM was designed for finding maximum likelihood (ML) estimates for probabilistic models with unobserved latent variables.

2. EM is especially helpful when the observed data likelihood function $p(x_v|\theta, \lambda)$ (which we want to maximize) is hard to evaluate, but the ML estimation given complete data (x_v, x_h) is relatively easy (because the M step corresponds to maximizing the expected log-likelihood of (x_v, x_h)). This is exactly the case in our auction setting.

The E Step

The integral in Equation (3.1) is generally intractable for our complex bidding process. However, we *can* compute a Monte Carlo approximation of this integral by drawing N samples from the distribution $p(x_h|x_v, \theta^{(old)}, \lambda^{(old)})$, and approximating the integral by a small sum over the samples (see e.g. [Andrieu et al. 2003]). Applied to our model, in each E step our task is therefore to generate samples from the distribution $p(x_h|x_v, \theta^{(old)}, \lambda^{(old)})$. Recall that x_h consists of the highest bid x_{hi} and the dropped bids x_d.

Given $\theta^{(old)}$ and the second highest bid (which is observed), the highest bid x_{hi} can easily be sampled. According to the bidding process described in Section 3.2, the highest bid was generated by repeatedly drawing from $f(x|\theta^{(old)})$ until we get a bid higher than the previously highest (now second-highest) bid. This is exactly the rejection-sampling procedure for the distribution $f(x|\theta^{(old)})$ truncated at the second highest bid and renormalized. For distributions with a simple functional form (e.g. normal distributions), it may be easier to sample directly from the truncated distribution by reversing the CDF (see e.g. [West 1994]).

Sampling the dropped bids x_d is a more difficult task. We use the following procedure, which is based on simulating the bidding process:

1. Sample m from $g(m|\lambda^{(old)})$.
2. If $m < n$, reject the sample and go back to step 1.
3. Simulate the bidding process using x_v and $m - n$ dropped bids:
   (a) Repeatedly draw a sample bid from $f(x|\theta^{(old)})$, and compare it to the current price level. If it is lower than the price level, add the bid to the set of dropped bids x_d. Otherwise, the current price level is increased to the next visible bid from x_v.
   (b) If the number of bids in x_d exceeds $m - n$, or if we used up all the bids in x_v before we have $m - n$ dropped bids in x_d, we reject this sample and go back to step 1. Only when we used up all bids in x_v and we have $m - n$ bids in x_d, do we accept the sample of x_d.
4. Repeat until we have generated N samples of x_d.

The M Step

Our task at each M step is to compute the maximum likelihood (ML) estimates of λ and θ from x_v and the generated samples of x_h. For many standard parametric families of distributions, there are analytical solutions for the ML estimates. For example, if $f(x)$ is a normal distribution $N(\mu, \sigma)$, then given the complete set of bids (x_v, x_h), the ML estimate of μ is the sample mean, and the ML estimate of σ is the sample standard deviation. If $g(m)$ is a Poisson distribution, then the ML estimate of the mean parameter λ is the mean of the number of bidders per auction. If analytical solutions do not exist we can use

numerical optimization methods such as simulated annealing. If prior distributions on λ and θ are available, we may instead compute maximum a posteriori (MAP) estimates, which are point estimates of λ and θ that maximize their posterior probabilities.

3.3.3 Learning Distributions in a Game-Theoretic Setting

The approach we just described is decision-theoretic because we estimate the distribution of bid amounts without considering how other agents would react to our behavior. What if we want to take a game-theoretic approach? Athey and Haile [2002] discussed estimation in the game-theoretic setting, however they generally assume that the number of bidders is known (there is a brief discussion of unknown number of bidders, but it is not relevant to our online auction setting).

Let $f(v)$ be the distribution of bidder's *valuations* (instead of bid amounts), and let $g(m)$ denote the distribution of number of bidders, as before. Given a bidder's valuation v, her bid amount in the game-theoretic setting is given by the Bayes-Nash equilibrium of the auction game. Many (but not all) auction games have symmetric pure strategy equilibria. Assume there is a symmetric pure strategy equilibrium given by the bid function $b(v|f, g)$. Our task is to estimate $f(v)$ and $g(m)$ given the observed bids.

We can use an EM learning algorithm similar to the one in Section 3.3.2 to estimate $f(v|\theta)$ and $g(m|\lambda)$, where θ and λ are parameters for f and g:

- E step: for each auction with observed bids x_v:
  - Compute observed bidders' valuations v_v from x_v by inverting the bid function.[5]
  - Generate bidders with valuations v_h who place hidden bids $x_h = b(v_h|f^{(old)}, g^{(old)})$. This is done by a similar procedure to the one in Section 3.3.2 that simulates the auction's bidding process.
- M step: update θ and λ to maximize the likelihood of the valuations (v_v, v_h).

3.4 Constructing a Bidding Agent

In Sections 3.2 and 3.3 we presented a model of the bidding process for a single auction, and proposed methods to estimate the distributions of bids and number of bidders in an auction. But our work is not done yet: how do we make use of these estimated distributions to compute a bidding strategy?

[5] If the bidding function does not have a single-valued inverse function, or if the equilibrium has mixed strategies, we just generate bidders with valuation v_v that would likely bid x_v under the equilibrium.

If we only participate in one English auction, bidding is simple: under the IPV model it is a dominant strategy to bid up to our valuation for the item, and we do not even need to estimate the distributions. However if we are interested in more complex environments, good estimates of the distributions $f(x)$ and $g(m)$ are essential for computing a good bidding strategy. In this section we discuss two such environments: finitely-repeated online auctions and unrepeated online auctions without proxy bidding. We consider the first problem from a decision-theoretic point of view, and take a game-theoretic approach to the second problem.

3.4.1 A Decision-Theoretic Approach to Repeated Auctions

In this section we develop a decision-theoretic bidding agent for finitely repeated auctions. We choose this setting because it is a reasonable model of the decision-theoretic problem we would face if we wanted to buy one item from an online auction site. Our estimation algorithm can straightforwardly be applied to more complex decision-theoretic models such as infinite horizon models with discount factors and combinatorial valuation models.

The Auction Environment

Suppose we only want to buy one item (say a Playstation 2) in an environment (say eBay) where multiple auctions for similar, but not necessarily identical, Playstation 2 systems are held regularly. Recall that we have assumed that utility is quasilinear: thus if we successfully win one item, our utility will be equal to our valuation for the item minus the price we paid. So our bidding agent's task is to compute a bidding strategy that will maximize this utility. Assume that we are only interested in the first k auctions that will be held after we arrived at the auction site. One motivation for such a restriction is that we prefer to have the item within a bounded amount of time. If we fail to win an item from the k auctions, we lose interest in the item and leave the auction site, and our utility is 0. (Alternatively, we could say that if we fail to win an auction then we could buy the item from a store after leaving the auction site, in which case we would get some other known and constant amount of utility.)

Some of the k auctions may overlap in time, but since eBay auctions have strict closing times, this can be modeled as a sequential decision problem, where our agent makes bidding decisions right before each auction closes. Number the auctions $1 \dots k$ according to their closing times. Let v_j denote our valuation for the item from auction j. Note that this allows the items in the auctions to be non-identical. Let b_j denote our agent's bid for auction j. Let U_j denote our agent's expected payoff from participating in auctions $j \dots k$, assuming we did not win before auction j. Let U_{k+1} be our payoff if we fail to win any of the auctions. For simplicity we define $U_{k+1} = 0$, though the analysis would be similar for any constant value. Suppose that for each auction j the number of other bidders is drawn from $g_j(m)$ and each bidder's bid is drawn from $f_j(x)$.

Since each auction j is an English auction, only the highest bid from other bidders affects our payoff. Let $f_j^1(x)$ and $F_j^1(x)$ respectively denote the probability density function and cumulative density function (CDF) of the highest bid from other bidders in the j^{th} auction. Then

$$F_j^1(x) = \sum_{m=2}^{\infty} g_j(m)(F_j(x))^m,$$

where $F_j(x)$ is the CDF of $f_j(x)$. Now U_j can be expressed as the following function of the future bids $b_{j:k} = (b_j, \ldots, b_k)$ and valuations $v_{j:k} = (v_j, \ldots, v_k)$:

$$U_j(b_{j:k}, v_{j:k}) = \int_{-\infty}^{b_j} (v_j - x) f_j^1(x) dx + (1 - F_j^1(b_j)) U_{j+1}(b_{j+1:k}, v_{j+1:k}) \quad (3.3)$$

The first term in Equation (3.3) is the expected payoff from the j^{th} auction; the second term is the expected payoff from the later auctions.

The Optimal Strategy

Greenwald and Boyan [2004] and Arora et al. [2003] have analyzed similar auction models. Following similar reasoning, we can derive the optimal bidding strategy for our auction model. Let $b_{j:k}^*$ be the optimal bidding strategy for auctions $j, \ldots, k$. Let $U_j^*(v_{j:k})$ denote the expected payoff under optimal strategy, i.e. $U_j^*(v_{j:k}) = U_j(b_{j:k}^*, v_{j:k})$. We can optimize U_j by working from the k^{th} auction to the first one in a manner similar to backward induction. By solving the first-order conditions of U_j, we obtain the optimal bidding strategy:

$$b_j^* = v_j - U_{j+1}^*(v_{j+1:k}) \quad (3.4)$$

In other words, our agent should shade her bids by the "option value", i.e. the expected payoff of participating in future auctions. The exception is of course the k^{th} auction; in this case there are no future auctions and the optimal bid is $b_k^* = v_k$. Thus we see that the standard result that honest bidding is optimal in an unrepeated auction is recovered as the special case $k = 1$.

The computation of the optimal bidding strategies requires the computation of the expected payoffs $U_j^* = U_j(b_{j:k}^*, v_{j:k})$, which involves an integral over the distribution $f_j^1(x)$ (Equation 3.3). In general this integral cannot be solved analytically, but we can compute its Monte Carlo approximation if we can sample from $f_j^1(x)$. If we can sample from $f_j(x)$ and $g_j(m)$, we can use the following straightforward procedure to generate a sample from $f_j^1(x)$:

1. draw m from $g_j(m)$;
2. draw m samples from $f_j(x)$;
3. keep the maximum of these m samples.

The bidding strategy $b^*_{1:k}$ computed using Equations (3.4) and (3.3) is optimal, provided that the distributions $f_j(x)$ and $g_j(m)$ are the correct distributions of bids and number of bidders for all $j \in 1 \ldots k$. Of course in general we do not know the true $f_j(x)$ and $g_j(m)$ and the focus of this chapter is to estimate the distributions from the bidding history and use the estimated distributions to compute the bidding strategy. As a result, the computed bidding strategy should be expected to achieve less than the optimal expected payoff. However, it is reasonable to think that better estimates of $f(x)$ and $g(m)$ should give bidding strategies with higher expected payoffs. This is confirmed in our experiments across a wide range of data sets, which we discuss in Section 3.5.

Auctions that Overlap in Time: Exploiting Early Bidding

We observe that while the optimal bid in auction j does not depend on f^1_j, it does depend on f^1_l for $l > j$. So far we have been estimating $f^1_l(x)$ using $f_l(x)$ and $g_l(m)$. In practice, auctions overlap in time, and we often observe some early bidding activity by other bidders in auctions $j+1, \ldots, k$ before we have to make a bid on auction j. This extra information allows us to make even more informed (posterior) estimates on $f^1_l(x)$, $l > j$, based on $f_l(x)$, $g_l(m)$ and the observed bids for auction l, which leads to a better bid for auction j.

Suppose we have observed $n-1$ early bids, denoted by x_v; the current highest bid x_{hi} is not revealed (but can be sampled from $f(x)$ truncated at the current price). Since the auction is not over, there will be some set of future bids x_{future} (possibly empty). When the auction closes, the highest bid from the other bidders will be $\max\{x_{hi}, x_{future}\}$. We can generate x_{future} if we know the number of future bids. We know the total number of bids m is drawn from $g(m)$, and the number of bids made so far is $n + |x_d|$, where x_d are the dropped bids so far, so the number of future bids is $m - n - |x_d|$. Now we have a procedure that samples from $f^1(x)$:

1. Simulate the auction using our model in Section 3.2 to generate x_d, the dropped bids so far.
2. Sample the total number of bids m from $g(m)$.
3. Compute the number of future bids, $m - n - |x_d|$.
   (a) If this quantity is negative, reject the sample.
   (b) Otherwise generate x_{future} as $(m - n - |x_d|)$ bids drawn from $f(x)$.
4. Generate x_{hi} and take the maximum of x_{future} and x_{hi}.

3.4.2 A Game-Theoretic Approach to Bidding in Online Auctions without Proxies

In Section 3.4.1 we discussed building a decision-theoretic bidding agent for repeated English auctions. What happens if we try to use the game-theoretic

approach on our auction model to build an agent that plays a Bayes-Nash equilibrium of the auction game?

The difficulty turns out to be identifying the equilibrium. If each bidder other than our agent only participates in one auction, then the situation is easy. In this case the other bidders' dominant strategies will be to bid truthfully up to the amount of their valuations. Assuming all bidders follow this dominant strategy as a game-theoretic approach would have us do, we find that the distribution of bids is the same as the distribution of valuations. Thus, our agent's strategy should be the same as the decision-theoretic optimal strategy described in Section 3.4.1. It is easy to verify that this strategy together with the other players' truthful bidding forms a Bayes-Nash equilibrium.

If the other bidders participate in more than one auction then the equilibrium strategy gets more complex, both strategically and computationally. Milgrom and Weber [2000] have derived equilibrium strategies for sequential auctions under some strong simplifying assumptions, such as identical items, a fixed number of bidders, and all bidders entering the auctions at the same time. In an online auction setting, these assumptions are not reasonable: the items are often heterogenous, the number of bidders is unknown *ex ante*, and bidders have different entry times and exit policies. The equilibrium strategy in the general case is not known. Therefore, it is an open problem to find a game-theoretic model for repeated online auctions that is realistic and yet still tractable.

Online Auctions without Proxies

Because of the difficulties described above, we turn to a slightly different auction setting in order to show how our distribution learning techniques can be used to build a game-theoretic bidding agent. Specifically, we will consider an online auction setting which differs in one crucial respect from the setting defined in Section 3.2. Bidders still participate in an online ascending auction against an unknown number of opponents and still arrive and bid sequentially; however, now we add the restriction that bidders cannot place proxy bids. Instead, bidders can now place only a single bid before they leave the auction forever; the game now takes on a flavor similar to a first-price auction, but with different information disclosure rules. We chose this game because it has hidden-bid characteristics similar to the online auctions we discussed earlier, but at the same time it also has a known, computationally tractable—and yet non-trivial—Bayes-Nash equilibrium.

More formally, suppose that there are m potential bidders interested in a single item. The number m is drawn from a distribution $g(m)$, and is observed by the auctioneer but not the bidders. Bidders have independent private valuations, drawn from the distribution $f(v)$. A bid history records every bid which is placed along with the bid amount, and is observable by all bidders. The auction has the following rules:

- The auctioneer determines a random sequential order to use in approaching bidders.

- Each bidder is approached and asked to make a bid. Each bidder is asked once and only once.
- When asked, a bidder has to either make a bid which is at least the current highest bid plus a constant minimum increment δ, or decide not to bid.
  1. If the bidder places a bid, it is recorded in the bidding history.
  2. Otherwise, no record is made of the fact that the auctioneer approached the bidder.
- The auction ends after the last bidder makes her decision. The highest submitted bid wins, and the winner pays her bid amount. The winner's utility is her valuation minus her payment; other bidders get zero utility.

Computing Equilibrium Strategies

We now turn to the construction of a game-theoretic bidding agent for the auction described in Section 3.4.2. We begin by supposing that the distributions $f(v)$ and $g(m)$ are common knowledge, which allows us to treat the auction as a Bayesian game and find its Bayes-Nash equilibrium.

Suppose there exists a pure-strategy equilibrium of this game. Consider bidder i with valuation v, who is asked to make the next bid when the bidding history is x_v. Denote $b(v|x_v)$ the equilibrium bidding strategy. Before we compute $b(v|x_v)$, let us first eliminate dominated strategies. As in classical sealed-bid first-price auctions, it is obvious that we should never bid higher than our valuation v. Let b_o denote the highest submitted bid so far, thus $b_o + \delta$ is the minimum bid amount. It immediately follows that if $v < b_o + \delta$ then we should not bid. On the other hand, if $v \geq b_o + \delta$, then we have non-negative expected payoffs by making a bid (as long as we bid below v).

Define as P the class of strategies which take the following form:

- if $v < b_o + \delta$, then do not bid
- if $v \geq b_o + \delta$, then bid an amount in the interval $[b_o + \delta, v]$

Following the above reasoning, any strategy not in P is weakly dominated. Thus, in equilibrium all bidders will play some strategy in P. Suppose bidder i's valuation is $v \geq b_o + \delta$. Thus she places a bid $b \in [b_o + \delta, v]$. Note that bidder i's expected payoff depends on her valuation, her bid b, and what future bidders do. Since future bidders play strategies in P, we know i will be outbid if and only if at least one future bidder has valuation greater than or equal to $b + \delta$. In other words, as long as everyone plays a strategy in P, a bidder's expected payoff depends on future bidders' valuations but not their exact bidding functions.

Denote by m_f the number of future bidders, and denote by $p_o = \Pr(m_f = 0|x_v)$ the probability that there will be no future bidders. Let $F^1(v)$ be the CDF of the (posterior) distribution of the highest future bid given x_v, conditioned on having at least one future bidder. In other words,

$$F^1(v) = E_{m_f|m_f>0}[F(v)^{m_f}] \tag{3.5}$$

where $F(v)$ is the CDF of the value distribution. F^1 and p_o depend on the posterior distribution of the number of future bidders, $\Pr(m_f|x_v)$. If we can generate samples of m_f, then F^1 and p_o can be straightforwardly approximated. Note that the set of bidders consists of past bidders (observed and hidden), the current bidder, and future bidders. Thus we can generate m_f using essentially the same procedure as described in Section 3.4.1, by drawing m from $g(m)$, simulating the auction so far to get the number of already dropped bids $|x_d|$, then letting $m_f = m - |x_v| - |x_d| - 1$. Note that we do not need to know the exact equilibrium strategy to generate the hidden bidders; the knowledge that the equilibrium strategy is in P is enough to determine whether a bidder should bid or not.

Now we can write the expected payoff $U(b, v)$ as:

$$U(b, v) = (1 - p_o)F^1(b + \delta)(v - b) + p_o(v - b) \tag{3.6}$$

Since v is fixed for each bidder, $U(b, v)$ is then a function of b. We can then use any standard numerical technique for function maximization in order to find the optimal $b^* \in [b_o + \delta, v]$ that maximizes U.

Theorem 3. *The following strategy (played by all bidders) is a Bayes-Nash equilibrium of our auction setting:*

- *if $v < b_o + \delta$, do not bid.*
- *if $v \geq b_o + \delta$, bid $b^* = \arg\max_{b \in [b_o + \delta, v]} U(b, v)$.*

The proof is straightforward: first observe that this strategy is in P. We have showed above that if everyone else is playing a strategy in P, then b^* as defined will maximize the bidder's utility. It follows that if everyone is playing the above strategy, then each bidder is playing a best response to the other bidders' strategies and so we have an equilibrium.

Learning the Distributions

At the beginning of Section 3.4.2 we assumed that the distributions $f(v)$ and $g(m)$ were commonly-known. Now we relax this assumption. We assume that we have access to the bidding histories of K past auctions, and consider the problem of learning the distributions.

As in our online English auctions, some information is hidden from the bidding history: the number and valuations of the bidders who decided not to bid. As discussed in Section 3.3.1, the simple approach of ignoring these hidden bidders could be expected to introduce bias to the estimates of f and g. Furthermore, the observed bids are not equal to the bidders' valuations; as shown above the equilibrium bid is always lower than the valuation.

To correctly account for the hidden information, we use our EM algorithm for game-theoretic settings, as described in Section 3.3.3. To implement the EM algorithm, an important subproblem is reversing the equilibrium bidding strategy. Formally, given a complete bidding history x_v of an auction and the

current estimates $f(v|\theta)$ and $g(m|\lambda)$, compute the corresponding valuations v_v such that bidders with valuations v_v playing the equilibrium strategy would have placed the bids x_v.

Consider each bid $b_i \in x_v$. Denote by b_o the previous observed bid. From Section 3.4.2 we know that $b_i = \arg\max_{b \in [b_o+\delta, v]} U(b, v)$. There are two cases:

1. If $b_i > b_o + \delta$, then b_i must be a local maximum in $[b_o + \delta, \quad v]$ satisfying the first-order condition $\frac{\partial U(b,v)}{\partial b} = 0$. Solving for v, we get

$$v = b_i + \frac{F^1(b_i + \delta) + p_o/(1 - p_o)}{f^1(b_i + \delta)} \tag{3.7}$$

   where f^1 is the derivative of F^1. We can numerically approximate F^1, f^1 and p_o by generating samples of m_f in the same way as in Section 3.4.2.

2. Otherwise, $b_i = b_o + \delta$. Intuitively, this means that the bidder's valuation is high enough to make a bid (i.e. $v \geq b_o + \delta$), but not high enough to fall into Case 1. Any bidder with a valuation in the interval

$$\left[b_o + \delta, \quad b_o + \delta + \frac{F^1(b_o + \delta) + p_o/(1 - p_o)}{f^1(b_o + \delta)}\right]$$

   would bid in this way. Thus we generate samples of v by drawing from $f(v|\theta)$ truncated to the above interval.

Once the EM algorithm has estimated the distributions f and g, our agent can use these distributions to compute the Bayes-Nash equilibrium strategy as described in Section 3.4.2.

3.5 Experiments

We evaluated both our EM learning approach and the simple approach[6] on several synthetic data sets and on real world data collected from eBay. We compared the approaches in three ways:

1. Which approach gives better estimates of the distributions $f(x)$, $g(m)$ and $f^1(x)$? This is important because better estimation of these distributions should give better results, regardless of whether agents take a decision-theoretic approach or a game-theoretic approach to bidding. We measure the closeness of an estimated distribution to the true distribution using the Kullback-Leibler (KL) Divergence from the true distribution to the estimated distribution. The smaller the KL Divergence, the closer the estimated distribution to the true one.

[6] We also looked at the variant of the simple approach that directly estimates the distribution $f^1(x)$ using the selling prices. Our results show that this approach consistently underestimates $f^1(x)$ as expected, and its performance is much worse than both the EM approach and the simple approach. For clarity, detailed results on this approach are not shown.

2. For repeated online auction data, which approach gives better expected payoff under the decision-theoretic bidding model, as described in Section 3.4.1?

3. For data from online auctions without proxies, which approach gives closer estimates to the Bayes-Nash equilibrium under the game-theoretic bidding model (i.e., computes ϵ-equilibria for smaller values of ϵ), as described in Section 3.4.2?

Our experiments show that the EM learning approach outperforms the simple approach in all three ways, across a wide range of data sets.

3.5.1 Repeated Online Auctions

In this section we consider repeated online auctions, and thus attempt to answer the first two questions above. We present results from four data sets:

- Data Set 1 has auctions of identical items, and we know the family of distributions that $f(x)$ and $g(m)$ belong to.
- Data Set 2 has auctions of non-identical items, but we know the bid distribution $f(x)$ is influenced linearly by an attribute a.
- Data Set 3 has auctions of identical items, but we do not know what kind of distributions $f(x)$ and $g(m)$ are. We use nonparametric estimation techniques to estimate the distributions.
- Data Set 4 is real-world auction data on identical items, collected from eBay.

Synthetic Data Set 1: Identical Items

In this data set, the items for sale in all auctions are identical, so the number of bidders and bid amounts come from stationary distributions $g(m)$ and $f(x)$. $f(x)$ is a Normal distribution $N(4, 3.5)$. $g(m)$ is a Poisson distribution shifted to the right: $g(m-2) = P(40)$, i.e. the number of bidders is always at least 2. The bidding history is generated using our model of the bidding process as described in Section 3.2. Each instance of the data set consists of bidding history from 40 auctions. We generated 15 instances of the data set.

Estimating the Distributions Both estimation approaches are informed of the parametric families from which $f(x)$ and $g(m)$ are drawn; their task is to estimate the parameters of the distributions, (μ, σ) for $f(x)$ and λ for $g(m)$. At the M step of the EM algorithm, standard ML estimates for μ, σ, and λ are computed, i.e. sample mean of the bid amounts for μ, standard deviation of the bid amounts for σ, and the mean of the number of bidders minus 2 (due to the shifting) for the Poisson parameter λ.

Our results show that the EM approach outperforms the simple approach in the quality of its estimates for the distributions $f(x)$, $g(m)$ and $f^1(x)$. Figure 3.2 shows typical estimated distributions[7] and the true distributions. We observe that the plot of the estimated $f(x)$ by the simple approach is significantly shifted to the right of the true distribution, i.e. the simple approach overestimated $f(x)$. We have also calculated KL Divergences from the true distributions to the estimated distributions, and the EM estimations have consistently lower KL Divergences. This difference was verified to be significant, using the non-parametric Wilcoxon sign-rank test.

Bidding in Repeated Auctions Estimates from both approaches were used to compute bidding strategies for an auction environment with 8 sequentially held auctions of the same kind of items, using the decision-theoretic model introduced in Section 3.4.1. The agent's "actual" expected payoffs $U_1(b, v)$ under these bidding strategies were then computed, using the true distributions. The optimal bidding strategy and its expected payoff were also computed.

Our results show that the EM approach gives rise to bidding strategies closer to the optimal strategy, and achieves higher expected payoffs, as compared to the simple approach. Figure 3.3 shows a plot of the bidding strategies in the first auction, and a box plot of the mean regrets, which is the differences between optimal expected payoffs and actual expected payoffs. Formally, let b^* denote the optimal strategy and $\hat{b}$ the strategy computed using estimated distributions, then the regret given our agent's valuation v is

$$R(v) = U_1(b^*, v) - U_1(\hat{b}, v)$$

and the mean regret is the expected value of $R(v)$ over the distribution of v, which we set to be the same as the other bidders' bid distribution f:

$$\bar{R} = \int_{-\infty}^{\infty} R(v) f(v) dv$$

From the box plot we observe that the mean regret of the EM approach is much smaller than that of the simple approach. The ratio between the mean regrets of the EM and simple approaches is $1 : 56$.

We also used the estimated distributions on the decision-theoretic model with overlapping auctions, as described in Section 3.4.1. We again tested both approaches on 8 sequentially held auctions, but now some early bidding activity on each auction was generated. These results are shown in Figure 3.4. Again we see that the EM approach achieves higher expected payoffs (and thus less regret) compared to the simple approach. The EM approach seemed to benefit more from the extra information of early bids than the simple approach: the ratio between the mean regrets of the EM and simple approaches increased to $1 : 390$.

[7] The distributions shown were randomly chosen from the 15 instances of the data set. We have verified that the plots of the other distributions are qualitatively similar.

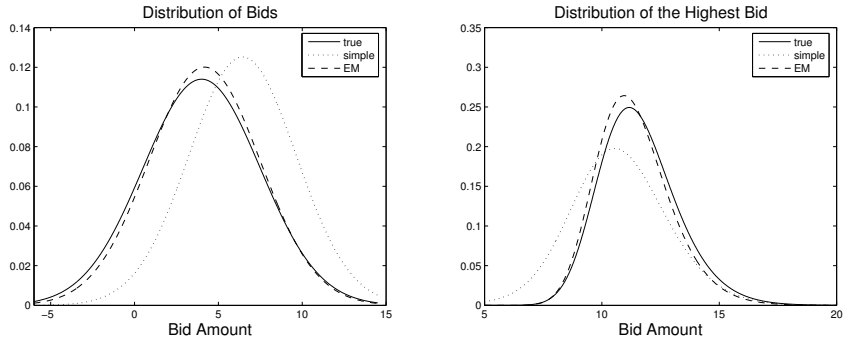

Figure 3.2: Results for Data Set 1, Distribution Estimation: distribution of bids $f(x)$ (left); distribution of highest bids $f^1(x)$ (right).

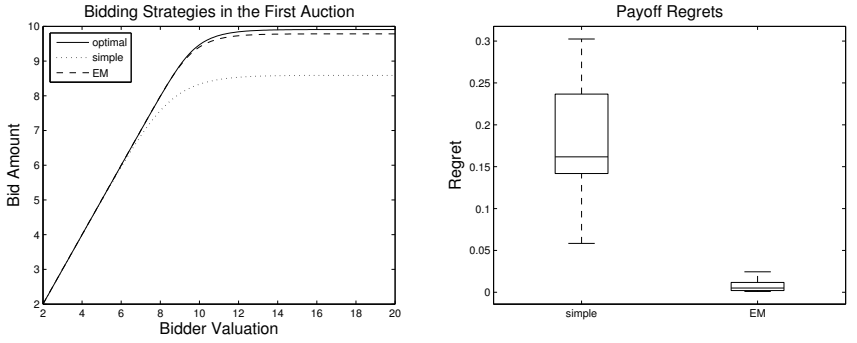

Figure 3.3: Results for Data Set 1, Bidding: bidding strategies in the first auction (left); box plot of payoff regrets of the two approaches (right).

Synthetic Data Set 2: Non-identical Items

In our second data set, the items on sale are not identical; instead the distribution of valuations are influenced by an observable attribute a. In this data set the dependence is linear: $f(x|a) = N(1.1a + 1.0, 3.5)$. $g(m)$ is a Poisson distribution as before: $g(m-2) = P(35)$. For each auction, a is sampled uniformly from the interval $[3, 9]$. In other words, this data set is similar to Data Set 1, except that the bid distribution $f(x)$ is drawn from a different parametric family. Both approaches now use linear regression to estimate the linear coefficients.

Estimating the Distributions Again, our results show that the EM approach outperforms the simple approach for this data set, in terms of its estimates for $f(x)$ and $g(m)$. Figure 3.5 (left) shows the estimated linear relation between the mean of $f(x|a)$ and a. From the figure we can see that the EM approach gives a much better estimate to the linear function. The simple approach again significantly overestimates the bid amounts. In fact the simple

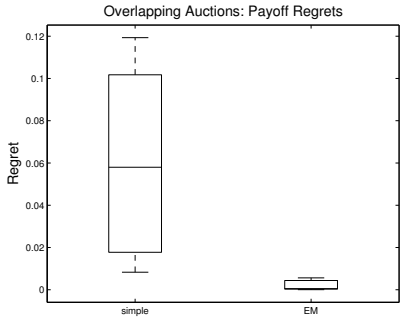

Figure 3.4: Box plot of expected payoff regrets for overlapping auctions

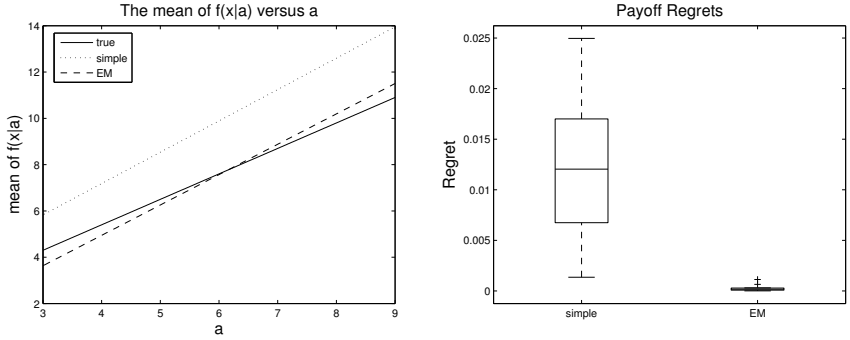

Figure 3.5: Results for Data Set 2: Linear relationship between the mean of $f(x|a)$ and a (left). Box plot of payoff regrets (right).

approach has consistently overestimated $f(x)$ for all the synthetic data sets we tested.

Bidding in Repeated Auctions We then used the estimated distributions to compute a decision-theoretic agent's bidding strategies and expected payoffs of an auction environment with 8 sequential auctions, where the attribute a of each item is observed. The EM approach also gives better expected payoff, the statistical significance of which is confirmed by Wilcoxon's sign-rank test. Figure 3.5 (right) shows a box plot of regrets from different instances of data sets, which shows that the EM approach achieved consistently higher payoffs.

Synthetic Data Set 3: Unknown Distributions

We go back to the identical items model with stationary distributions $f(x)$ and $g(m)$. For this data set, $f(x)$ is a Gamma distribution with shape parameter 2 and scale parameter 3. $g(m)$ is a mixture of two Poisson distributions: $P(4)$ with probability 0.6 and $P(60)$ with probability 0.4. But now the estimation

approaches does not know the types of the true distributions. Instead, both use kernel density estimation (kernel smoothing), a nonparametric estimation strategy. Essentially, given N samples from a distribution $p(x)$, we estimate $p(x)$ by a mixture of N kernel functions centered at the N samples.

Estimating the Distributions A Gaussian kernel is used for estimating $f(x)$ and a uniform kernel is used for estimating $g(m)$. At each M step of the EM algorithm, the bandwidth parameters of the two kernel estimations need to be selected. We use the simple "rule of thumb" strategy [Silverman 1986] for bandwidth selection. We used the same kernel estimation and bandwidth selection technique for the simple approach.

Our results show that the EM approach gives better estimates than the simple approach. Figure 3.6 shows typical estimated distributions and true distributions. From the figure we can observe that the EM estimates of $f(x)$, $g(m)$ and $f^1(x)$ are much closer to the true distributions that the simple estimates. The EM estimates have significantly smaller KL Divergences compared to the simple estimates, verified by Wilcoxon's sign-rank test.

Bidding in Repeated Auctions We then computed the expected payoffs under the decision-theoretic model with 8 sequential auctions. The expected payoffs of the EM approach were not significantly better than that of the simple approach, as shown by the box plot in Figure 3.6. One possible explanation is that although KL divergence is a good measure of similarity of distributions in general, under this particular sequential auction decision-theoretic model KL divergence might not be the best measure of quality. The bidding strategy as defined in (3.4) and (3.3) is optimal if the distributions f and g are the true underlying distributions. Using our estimated distributions, the resulting bidding strategy may be higher or lower than the optimal bids. However from our experience in these experiments, bidding too high is more costly than bidding too low. This is not taken into account in the KL divergence computation as well as our ML estimation procedure. This suggests that a possible future research direction is to identify a loss function suited for the sequential auction bidding model, and compute estimated distributions by minimizing that loss function. Another possible approach is to compute a posterior distribution instead of point estimates of the parameters in each iteration. For example, West [1994] used Gibbs sampling techniques to compute the posterior distribution of parameters in a relatively simple model of incomplete data. Bidding strategies computed using the distribution of parameters should be free of the problem mentioned above. However, this kind of approach would be more computationally expensive.

eBay Data on Sony Playstation 2 Systems

Our experiments on synthetic data sets showed that our EM approach gave good estimates of the true distributions in several different settings. However, the synthetic data sets are generated using our model for the bidding process.

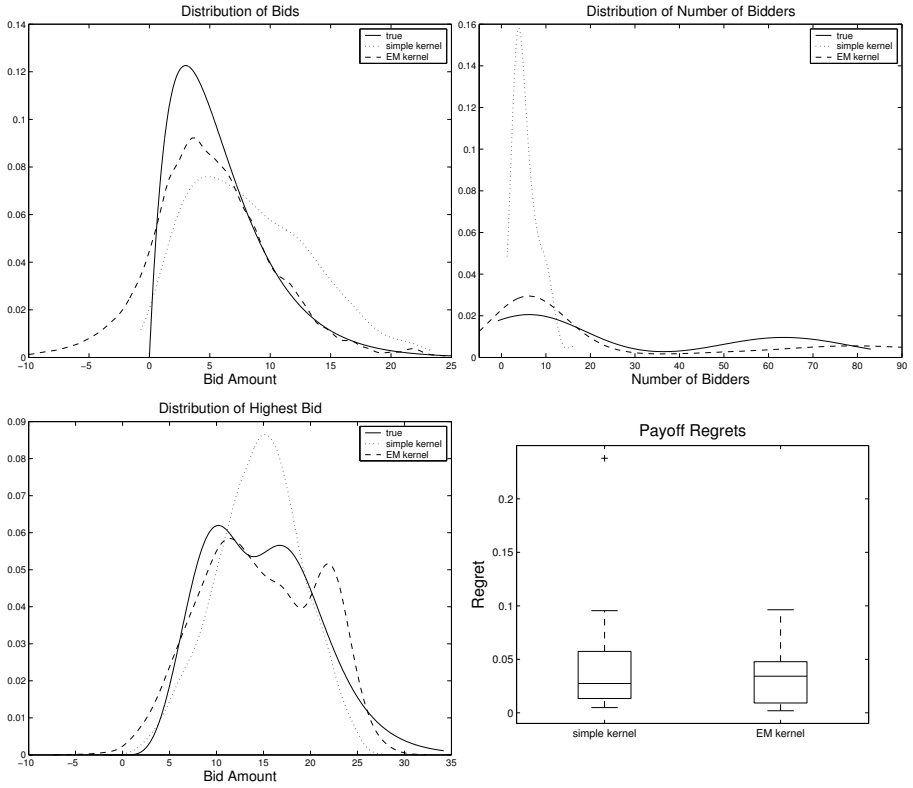

Figure 3.6: Results for Data Set 3: Distribution $f(x)$ (top-left). Distribution $g(m)$ (top-right). Distribution $f^1(x)$ (bottom-left). Box plot of payoff regrets (bottom-right).

Thus, the above experiments do not tell us whether our model for the bidding process is an accurate description of what happens in real world online auctions. To answer this question, we wanted to test our approach on real world bid data. On eBay, the bidding histories of completed auctions are available for 30 days. Unfortunately, information on the hidden bids, especially the proxy bids of the winners of the auctions, is not publicly available. So unlike in the synthetic data experiments, we cannot compare our estimated distributions with the "true" distributions.[8]

To get around this problem, we used the following approach. First we collected bidding histories from a set of eBay auctions. Next we pretended that those highest bids were not placed, and the previously second highest bids were the highest bids. We then "hid" these new highest bids of each auction. This gave us a "ground truth" for the hidden bids which allowed us to evaluate our

[8]Of course we could have used our techniques to generate values for the missing bids; however, this would have been unfair when the goal was to test these techniques!

different approaches. It is true that this approach of hiding bids changed the underlying distribution of bids; however, this did not worry us as our goal was not to learn the true distribution of bids in our eBay auctions. Instead, our goal was to evaluate our algorithms' performance on a realistic, non-synthetic data set. We believe that despite the removal of the high bid our data set preserves qualitative characteristics of the original bidding history data. If our model of the bidding process is correct, then our EM approach should be able to correctly account for the hidden bids in this data set and produce good estimates of $f^1(x)$.

We collected bidding history of eBay auctions on brand new Sony Playstation 2 (Slim Model) consoles, over the month of March 2005. We chose to study these auctions because they had been previously studied by Shah et al. [2003], who argued that bidders' valuations on Playstations tend to be close to the private value model. We considered only auctions that lasted one day and had at least 3 bidders.[9] We found 60 auctions that satisfied these requirements. We then randomly divided our data into a training set and a testing set.

Estimating the Distributions We tested four learning approaches: the EM and simple approaches that estimate a Normal distribution for $f(x)$ and a Poisson distribution for $g(m)$, and the EM and simple approaches that use kernel density estimation to estimate $f(x)$ and $g(m)$. Of course we did not have a ground truth for these distributions, so it was not possible to compare the accuracy of the predictions. However, we could use both approaches' estimates to estimate $f^1(x)$ based on the training set, and compare these estimates against the highest bids from the test set. We did 8 runs of this experiment with different random partitions of training set and testing set, and aggregated the results. The KL Divergences of $f^1(x)$ of the approaches were similar, and no one approach was significantly better than the others.

Bidding in Repeated Auctions We then computed the expected payoffs under the decision-theoretic model. The EM approaches achieved significantly higher payoffs than the simple approaches, as shown in Figure 3.7. The approaches using parametric models achieved similar payoffs to the corresponding approaches with kernels. The good performance of the parametric estimation EM approach for the eBay data set indicates that the Normal and Poisson models for $f(x)$ and $g(m)$ may be adequate models for modeling bidding on eBay.

The EM approaches did not have better KL divergence than the simple approaches, but nevertheless outperformed the simple approaches in the repeated auction experiments. This is similar to our situation in Data Set 3. Our experiments have shown that there is a positive correlation between better KL divergence and better performance in repeated auctions, but that this correlation is not perfect.

[9] We needed at least 3 bidders so that we could drop one and still have one observed bid.

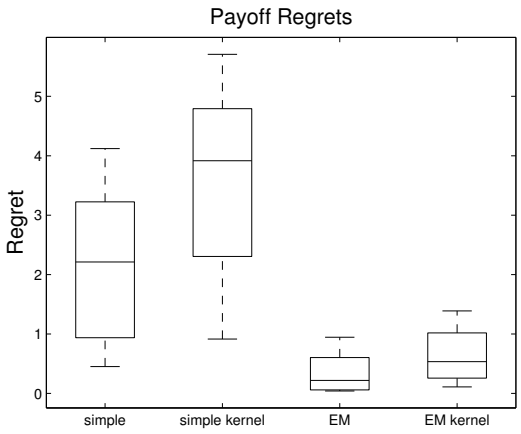

Figure 3.7: Box plot of payoff regrets on the eBay Data Set

3.5.2 Online Auctions without Proxies

We built a data set modeling online auctions without proxies in order to compare the EM approach against the simple approach in the game-theoretic setting described in Section 3.4.2. Thus, in this section we consider questions 1 and 3 from the list at the beginning of Section 3.5.

Similar to Section 3.5.1, we use a normal distribution $N(5.0, 2.0)$ for $f(v)$ and a shifted Poisson distribution (with $\lambda = 10$) for $g(m)$. Each instance of the data set consists of bidding histories from 30 auctions.

Estimating the Distributions As might be expected from the results in the previous section, we found that the EM approach gave much better estimates of $f(v)$ and $g(m)$ than the simple approach. Figure 3.8 shows typical estimated distributions and the true distributions.

Estimating the Bayes-Nash Equilibrium We then compared the game-theoretic bidding agents that would have been built using the distributions estimated by the two learning approaches. Unlike in Section 3.5.1, we cannot simply use expected payoff as a measure of performance, since in a game-theoretic setting our payoff depends other agents' strategies; for example, a bad approximation to an equilibrium could actually lead to increased payoffs for all agents. (Consider e.g., the prisoner's dilemma.) Instead, what we can measure is the amount that each player could gain by unilaterally deviating from the current strategy at the "equilibrium" strategy profile computed using each learning approach. One way of thinking of this value relates to the concept of ϵ-equilibrium. (Recall that a strategy profile is an ϵ-equilibrium if each agent can gain at most ϵ by unilaterally deviating from the strategy profile.) *Every* strategy profile is

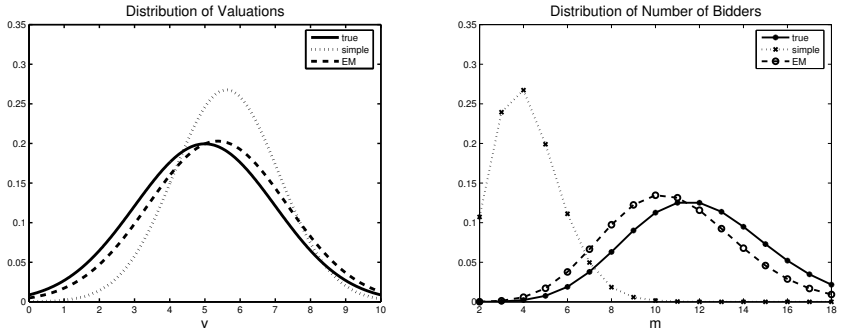

Figure 3.8: Results for Online Auctions without Proxies: the value distributions $f(v)$ (left); the distributions of number of bidders $g(m)$ (right).

an ϵ-equilibrium for some ϵ; what we compute is the smallest ϵ for which the learned strategies form an ϵ-equilibrium.

For our auction setting, we observe (from Theorem 3) that no matter what distributions we use for f and g, our agents will play a strategy in P. As a result, the true equilibrium strategy (computed from Theorem 3 using the true distributions) is always a best response to the strategies our agents play. Given our agent's valuation v and observed bidding history so far x_v, we can compute the difference between the expected payoff of the best response and the expected payoff of our agent's strategy. The ϵ for the game is then the expectation of this difference over v, m, and bidding history x_v. We approximate this expectation by sampling v, m, x_v and taking the mean of the payoff differences. We generated bidding histories for 20 auctions using each approach, and for each bidder in these auctions we computed expected payoff differences for 50 valuations v drawn from the true $f(v)$.

Our results show that strategy profiles computed using the EM approach are *epsilon*-equilibria for much smaller values of ϵ than the strategy profiles computed using the simple approach. This means that the EM bidding agent's strategy is much closer to the true equilibrium. We computed ϵ for 15 instances of the training data set, and Figure 3.9 gives a box plot of the resulting ϵ's.

3.6 Related Work

Our EM approach is similar in spirit to an approach by Boutilier et al. [1999]. This work concerns a decision-theoretic MDP approach to bidding in sequential first-price auctions for complementary goods. For the case when these sequential auctions are repeated, this paper discusses learning a distribution of other agents' highest bids for each good, based on the winning bids in past auctions. If the agent's own bid wins in an auction, the highest bid by the other agents is hidden because only the winning bid is revealed. To overcome this problem of hidden bids, the paper uses an EM approach to learn the distribution of the

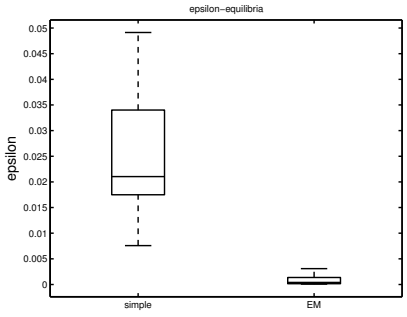

Figure 3.9: Results for Online Auctions without Proxies: box plot of epsilon-equilibria using the simple and EM approaches

highest bid.

In our English auction setting the highest bids are *always* hidden, and thus cannot be directly estimated. Instead we have to estimate the distributions of bids $f(x)$ and of number of bidders $g(m)$—which requires us to take into account bids other than the highest—and then compute the distribution of highest bids using $f(x)$ and $g(m)$. As a result the learning task in our domain is more complicated than the problem considered by Boutilier et al. [1999].

Several other papers have tried to solve the hidden bid problem in various auction settings. Rogers et al. [2005] studied English auctions with discrete bid levels. They discussed estimating the distributions of valuations and numbers of bidders (f and g) from data, then using the estimated distributions to compute an optimal set of bid levels (including the reserve price) for the auctioneer. Their learning approach is to look at only the final prices of the auctions, and then to use Bayesian inference to compute posterior distributions of the parameters of f and g given the final prices of previous auctions. We note that this approach ignores all the earlier bids, which carry information about f and g, while our approach uses all bids observed. Furthermore, Rogers et al.'s [2005] approach works only for parametric distributions. If the true underlying distributions are not in the chosen parametric family of distributions, parametric estimation tends to give poor results. In Section 3.5.1 we showed that our approach can use nonparametric estimation techniques (kernel density estimation). Finally, the method of [Rogers et al. 2005] needs to compute the joint distribution of multiple parameters, which takes exponential time in the number of parameters. Our EM approach only tries to compute ML/MAP estimates of the parameters, so each iteration of EM scales roughly linearly with the number of parameters.

Another relevant paper is [Song 2004]. Like us, Song studies the estimation problem in online English auctions in eBay-like environments. However she used a different approach, based on the theoretical result that the second- and third-highest valuations uniquely identify the underlying value distribution. However, applying this fact to the eBay model presents a problem: while the highest

observed bid corresponds to the second-highest valuations, the second-highest observed bid is not necessarily the third-highest valuation. This is because the first- or second- highest bidders may have submitted bids higher than the third-highest value before the third-highest valued bidder had a chance to submit her final bid, which means her bid can be hidden from the bidding history. Thus the second-highest observed bid is sometimes less than the third-highest valuation, and ignoring this fact would introduce bias to the estimation. Song recognizes this problem, and her solution is to use data from auctions in which the first- or second-highest bidder submitted bids higher than the second-highest observed bid *late* in the auction, because these auctions have relatively higher probability that their second-highest observed bids are third-highest valuations. However, this approach merely reduces expected bias rather than avoiding it entirely. Indeed, we note that there is empirical evidence [Roth and Ockenfels 2002] that bidding activity late in auctions has much higher density than earlier bidding activity. This suggests that even for the selected auctions of Song's approach, there may be significant probability that second-highest observed bids are less than third-highest valuations. Our approach models the hidden bidders, so does not suffer from the bias introduced in her approach, and does not need to drop any auctions from consideration. Finally, Song [2004] did not discuss how to estimate the distribution of the number of bidders, which is essential for computing optimal bidding strategies in our repeated auctions setting (see Section 3.4).

Haile and Tamer [2003] analyzed a different form of the hidden bid problem: a bidder may have submitted an earlier bid below her valuation, then the current price rises above her valuation, so she does not bid again. As a result, bidders' final observed bids may be below their valuations. Haile and Tamer [2003] proposed a method to compute bounds on the value distributions given the observed bids. In our model, bidders' final observed bids coincide with their willingness to pay, so this issue is not addressed in our current model. On the other hand, Haile and Tamer's [2003] approach was intended for physical (as opposed to online) auctions, where there are no fixed closing times, and the number of potential bidders are assumed to be known. Thus their approach cannot be directly applied to online auctions where the number of bidders are inherently unknown. An interesting future research direction is to combine our approach that deal with hidden bidders with Haile and Tamer [2003]'s technique for bounding the valuation distribution.

3.7 Conclusion

In this chapter we have described techniques for building bidding agents in online auction settings that include hidden bids. In particular, we have addressed the issue of estimating the distributions of the number of bidders and bid amounts from incomplete auction data. We proposed a learning approach based on the EM algorithm that takes into account the missing bids by iteratively generating missing bids and doing maximum likelihood estimates on the completed set of

bids. We applied our approach to both decision-theoretic and game-theoretic settings, and conducted experiments on both synthetic data as well as on eBay data. Our results show that our approach never did worse and often did much better than the the straightforward approach of ignoring the missing data, both in terms of the quality of the estimates and in terms of expected payoffs under a decision theoretic bidding model.

Bibliography

Altman, Alon and Moshe Tennenholtz (2005). Ranking systems: The PageRank axioms. *ACM Conference on Electronic Commerce*.

Andrieu, C., N. de Freitas, A. Doucet and M.I. Jordan (2003). An introduction to MCMC for machine learning. *Machine Learning*.

Anthony, P., W. Hall, V.D. Dang and N. Jennings (2001). Autonomous agents for participating in multiple online auctions. *IJCAI Workshop on EBusiness and the Intelligent Web*.

Arora, A., H. Xu, R. Padman and W. Vogt (2003). Optimal bidding in sequential online auctions. Working Paper.

Athey, S. and P. Haile (2002). Identification in standard auction models. *Econometrica*, *70*(6), 2107–2140.

Bhat, N. and K. Leyton-Brown (2004). Computing Nash equilibria of action-graph games. *UAI*.

Blum, B., C. Shelton and D. Koller (2002). Gametracer. http://dags.stanford.edu/Games/gametracer.html.

Boutilier, C., M. Goldszmidt and B. Sabata (1999). Sequential auctions for the allocation of resources with complementarities. *IJCAI*.

Bowling, M. (2004). Convergence and no-regret in multiagent learning. *NIPS*.

Bowling, M. and M. Veloso (2001). Convergence of gradient dynamics with a variable learning rate. *ICML*.

Byde, A (2002). A comparison among bidding algorithms for multiple auctions. *Agent-Mediated Electronic Commerce IV*.

Cai, G. and P.R. Wurman (2003). *Monte Carlo approximation in incomplete-information, sequential-auction games* (Technical Report). North Carolina State University.

Chen, X. and X. Deng (2005). *Settling the complexity of 2-player Nash equilibrium* (Technical Report TR05-150). ECCC.

Conitzer, V. and T. Sandholm (2003). Complexity results about Nash equilibria. *IJCAI*.

Daskalakis, C., P. W. Goldberg and C. H. Papadimitriou (2005). *The complexity of computing a Nash equilibrium* (Technical Report TR05-115). ECCC.

Dempster, A., N. Laird and D. Rubin (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, *39*(1), 1–38.

Goldberg, P.W. and C.H. Papadimitriou (2005). *Reducibility among equilibrium problems* (Technical Report TR05-090). ECCC.

Golle, Philippe, Kevin Leyton-Brown, Ilya Mironov and Mark Lillibridge (2001). Incentives for sharing in peer-to-peer networks. *International Workshop on Electronic Commerce (WELCOM)*.

Govindan, S. and R. Wilson (2003). A global Newton method to compute Nash equilibria. *Journal of Economic Theory*.

Govindan, S. and R. Wilson (2004). Computing Nash equilibria by iterated polymatrix approximation. *Journal of Economic Dynamics and Control*, *28*, 1229–1241.

Greenwald, A. and J. Boyan (2004). Bidding under uncertainty: Theory and experiments. *UAI*.

Haile, Philip A. and Elie Tamer (2003). Inferences with an incomplete model of English auctions. *Journal of Political Economy*, *111*(1), 1–51.

Kearns, M.J., M.L. Littman and S.P. Singh (2001). Graphical models for game theory. *UAI*.

Klemperer, P. (2000). Auction theory: A guide to the literature. In P. Klemperer (Ed.), *The economic theory of auctions*. Edward Elgar.

Koller, D., N. Megiddo and B. von Stengel (1994). Fast algorithms for finding randomized strategies in game trees. *STOC*.

Koller, D. and B. Milch (2001). Multi-agent influence diagrams for representing and solving games. *IJCAI*.

LaMura, P. (2000). Game networks. *UAI*.

Leyton-Brown, K. and M. Tennenholtz (2003). Local-effect games. *IJCAI*.

Mackie-Mason, J.K., A. Osepayshvili, D.M. Reeves and M.P. Wellman (2004). Price prediction strategies for market-based scheduling. *Fourteenth International Conference on Automated Planning and Scheduling* (pp. 244–252).

Milgrom, P. and R. Weber (2000). A theory of auctions and competitive bidding, II. In P. Klemperer (Ed.), *The economic theory of auctions*. Edward Elgar.

Osepayshvili, A., M.P. Wellman, D.M. Reeves and J.K. Mackie-Mason (2005). Self-confirming price prediction for simultaneous ascending auctions. *UAI*.

Papadimitriou, C.H. (2005). Computing correlated equilibria in multiplayer games. *STOC*.

Porter, R., E. Nudelman and Y. Shoham (2004). Simple search methods for finding a Nash equilibrium. *AAAI* (pp. 664–669).

Powers, R. and Y. Shoham (2004). New criteria and a new algorithm for learning in multiagent systems. *NIPS*.

Regev, O. and N. Nisan (1998). The popcorn market: Online markets for computational resources. *International Conference on Information and Computation Economies*.

Rogers, A., E. David, J. Schiff, S. Kraus and N. R. Jennings (2005). Learning environmental parameters for the design of optimal english auctions with discrete bid levels. *International Workshop on Agent-Mediated E-Commerce*. Utrecht, Netherlands.

Rosenthal, R.W. (1973). A class of games possessing pure-strategy Nash equilibria. *Int. J. Game Theory*, *2*, 65–67.

Roth, A.E. and A. Ockenfels (2002). Last-minute bidding and the rules for ending second-price auctions: Evidence from eBay and Amazon auctions on the internet. *American Economic Review*.

Roughgarden, T. and E. Tardos (2004). Bounding the inefficiency of equilibria in nonatomic congestion games. *Games and Economic Behavior*, *47*(2), 389–403.

Shah, H.S., N.R. Joshi, A. Sureka and P.R. Wurman (2003). Mining for bidding strategies on eBay. *Lecture Notes on Artificial Intelligence*.

Silverman, B.W. (1986). *Density estimation*. London: Chapman and Hall.

Song, Unjy (2004). Nonparametric estimation of an eBay auction model with an unknown number of bidders. University of British Columbia.

Stone, P., R.E. Schapire, J.A. Csirik, M.L. Littman and D. McAllester (2002). Attac-2001: A learning, autonomous bidding agent. *Agent-Mediated Electronic Commerce IV* (pp. 143–160).

van der Laan, G., A.J.J. Talman and L. van der Heyden (1987). Simplicial variable dimension algorithms for solving the nonlinear complementarity problem on a product of unit simplices using a general labelling. *Mathematics of OR*, *12*(3), 377–397.

Waldspurger, C.A., T. Hogg, B. A. Huberman, J. O. Kephart and W. S. Stornetta (1992). Spawn: A distributed computational economy. *IEEE Transactions on Software Engineering*.

Weber, R. (1983). Multi-object auctions. In R. Engelbercht-Wiggans, M. Shubik and R. Stark (Eds.), *Auctions, bidding, and contracting: Uses and theory*, 165–191. New York University Press.

Wellman, M.P., A. Greenwald, P. Stone and P.R. Wurman (2002). The 2001 Trading Agent Competition. *IAAI*.

West, M. (1994). Discovery sampling and selection models. In J.O. Berger and S.S. Gupta (Eds.), *Decision theory and related topics IV*. New York: Springer Verlag.

Zhang, N.L. and D. Poole (1996). Exploiting causal independence in bayesian network inference. *JAIR*, *5*, 301–328.